



man pages section 2: System Calls

Sun Microsystems, Inc.
901 San Antonio Road
Palo Alto, CA 94303-4900
U.S.A.

Part No: 816-1056-10
November 2001

Copyright 2001 Sun Microsystems, Inc. 901 San Antonio Road Palo Alto, CA 94303-4900 U.S.A. All rights reserved.

This product or document is protected by copyright and distributed under licenses restricting its use, copying, distribution, and decompilation. No part of this product or document may be reproduced in any form by any means without prior written authorization of Sun and its licensors, if any. Third-party software, including font technology, is copyrighted and licensed from Sun suppliers.

Parts of the product may be derived from Berkeley BSD systems, licensed from the University of California. UNIX is a registered trademark in the U.S. and other countries, exclusively licensed through X/Open Company, Ltd.

Sun, Sun Microsystems, the Sun logo, docs.sun.com, AnswerBook, AnswerBook2, and Solaris are trademarks, registered trademarks, or service marks of Sun Microsystems, Inc. in the U.S. and other countries. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. in the U.S. and other countries. Products bearing SPARC trademarks are based upon an architecture developed by Sun Microsystems, Inc.

The OPEN LOOK and Sun™ Graphical User Interface was developed by Sun Microsystems, Inc. for its users and licensees. Sun acknowledges the pioneering efforts of Xerox in researching and developing the concept of visual or graphical user interfaces for the computer industry. Sun holds a non-exclusive license from Xerox to the Xerox Graphical User Interface, which license also covers Sun's licensees who implement OPEN LOOK GUIs and otherwise comply with Sun's written license agreements.

Federal Acquisitions: Commercial Software—Government Users Subject to Standard License Terms and Conditions.

DOCUMENTATION IS PROVIDED "AS IS" AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

Copyright 2001 Sun Microsystems, Inc. 901 San Antonio Road Palo Alto, CA 94303-4900 U.S.A. Tous droits réservés

Ce produit ou document est protégé par un copyright et distribué avec des licences qui en restreignent l'utilisation, la copie, la distribution, et la décompilation. Aucune partie de ce produit ou document ne peut être reproduite sous aucune forme, par quelque moyen que ce soit, sans l'autorisation préalable et écrite de Sun et de ses bailleurs de licence, s'il y en a. Le logiciel détenu par des tiers, et qui comprend la technologie relative aux polices de caractères, est protégé par un copyright et licencié par des fournisseurs de Sun.

Des parties de ce produit pourront être dérivées du système Berkeley BSD licenciés par l'Université de Californie. UNIX est une marque déposée aux Etats-Unis et dans d'autres pays et licenciée exclusivement par X/Open Company, Ltd.

Sun, Sun Microsystems, le logo Sun, docs.sun.com, AnswerBook, AnswerBook2, et Solaris sont des marques de fabrique ou des marques déposées, ou marques de service, de Sun Microsystems, Inc. aux Etats-Unis et dans d'autres pays. Toutes les marques SPARC sont utilisées sous licence et sont des marques de fabrique ou des marques déposées de SPARC International, Inc. aux Etats-Unis et dans d'autres pays. Les produits portant les marques SPARC sont basés sur une architecture développée par Sun Microsystems, Inc.

L'interface d'utilisation graphique OPEN LOOK et Sun™ a été développée par Sun Microsystems, Inc. pour ses utilisateurs et licenciés. Sun reconnaît les efforts de pionniers de Xerox pour la recherche et le développement du concept des interfaces d'utilisation visuelle ou graphique pour l'industrie de l'informatique. Sun détient une licence non exclusive de Xerox sur l'interface d'utilisation graphique Xerox, cette licence couvrant également les licenciés de Sun qui mettent en place l'interface d'utilisation graphique OPEN LOOK et qui en outre se conforment aux licences écrites de Sun.

CETTE PUBLICATION EST FOURNIE "EN L'ETAT" ET AUCUNE GARANTIE, EXPRESSE OU IMPLICITE, N'EST ACCORDEE, Y COMPRIS DES GARANTIES CONCERNANT LA VALEUR MARCHANDE, L'APTITUDE DE LA PUBLICATION A REPOUDRE A UNE UTILISATION PARTICULIERE, OU LE FAIT QU'ELLE NE SOIT PAS CONTREFAISANTE DE PRODUIT DE TIERS. CE DENI DE GARANTIE NE S'APPLIQUERAIT PAS, DANS LA MESURE OU IL SERAIT TENU JURIDIQUEMENT NUL ET NON AVENU.



011029 @2471



Contents

Preface 9

Introduction 15

Intro(2) 16

System Calls 47

access(2) 48

acct(2) 50

acl(2) 51

adjtime(2) 53

audit(2) 55

auditon(2) 57

auditsvc(2) 62

chdir(2) 64

chmod(2) 66

chown(2) 70

chroot(2) 73

chstate(2) 75

creat(2) 76

devpolicy(2) 79

exec(2) 80

execl(2) 87

execle(2) 94

execlp(2) 101

execv(2) 108

execve(2) 115
 execvp(2) 122
 fac(2) 129
 fchdir(2) 131
 fchmod(2) 133
 fchown(2) 137
 fchroot(2) 140
 fcntl(2) 142
 fgetcmwfsrange(2) 151
 fgetcmwlabel(2) 153
 fgetfattrflag(2) 155
 fgetfpriv(2) 159
 fgetfsattr(2) 162
 fgetmldadorn(2) 164
 fgetslname(2) 166
 fork1(2) 169
 fork(2) 173
 fpathconf(2) 177
 fsetcmwlabel(2) 181
 fsetfattrflag(2) 185
 fsetfpriv(2) 189
 fstat(2) 192
 fstatvfs(2) 196
 getaudit(2) 199
 getaudit_addr(2) 201
 getauid(2) 203
 getclearance(2) 204
 getcmwfsrange(2) 205
 getcmwlabel(2) 207
 getcmwplabel(2) 209
 getdents(2) 210
 getfattrflag(2) 212
 getfpriv(2) 216
 getfsattr(2) 219
 getgroups(2) 221
 getmldadorn(2) 223
 getmsgqcmwlabel(2) 225

getpattr(2)	226
getpgid(2)	229
getpgrp(2)	231
getpid(2)	233
getppid(2)	235
getppriv(2)	237
getrlimit(2)	239
getsemcmwlabel(2)	243
getshmcmwlabel(2)	244
getsid(2)	245
getsldname(2)	246
kill(2)	249
lchown(2)	251
lgetcmwlabel(2)	254
link(2)	256
llseek(2)	258
lseek(2)	259
lsetcmwlabel(2)	261
lstat(2)	265
mkdir(2)	269
mknod(2)	272
mldgetfattrflag(2)	276
mldsetfattrflag(2)	280
mount(2)	284
msgctl(2)	288
msgget(2)	290
msggetl(2)	292
msgrcv(2)	294
msgsnd(2)	296
nice(2)	298
open(2)	299
pathconf(2)	305
p_online(2)	309
pread(2)	312
preadl(2)	318
priocntl(2)	324
priocntlset(2)	333

processor_bind(2) 335
pwrite(2) 337
p writel(2) 344
read(2) 351
readl(2) 357
readlink(2) 363
readv(2) 365
readvl(2) 371
rename(2) 377
rmdir(2) 380
secconf(2) 382
semctl(2) 383
semget(2) 386
semgetl(2) 388
semop(2) 390
semtimedop(2) 394
setaudit(2) 398
setaudit_addr(2) 400
setauid(2) 402
setclearance(2) 403
setcmwlabel(2) 404
setcmwplabel(2) 408
setegid(2) 409
seteuid(2) 411
setfattrflag(2) 413
setfpriv(2) 417
setgid(2) 420
setgroups(2) 422
setpatrr(2) 424
setppriv(2) 427
setregid(2) 429
setreuid(2) 430
setrlimit(2) 431
setuid(2) 435
shmat(2) 437
shmctl(2) 440
shmdt(2) 443

shmget(2)	446
shmgetl(2)	448
shmop(2)	450
sigsend(2)	453
sigsendset(2)	455
stat(2)	457
statvfs(2)	461
stime(2)	464
swapctl(2)	465
symlink(2)	469
sysinfo(2)	471
tokmapper(2)	475
uadmin(2)	476
ulimit(2)	478
umount(2)	479
umount2(2)	481
unlink(2)	483
utimes(2)	485
vfork(2)	487
write(2)	489
writel(2)	496
writev(2)	503
writevl(2)	510

Index	517
--------------	------------

Preface

Overview

A man page is provided for both the naive user and the sophisticated user who is familiar with the Trusted Solaris operating environment and is in need of online information. A man page is intended to answer concisely the question “What does it do?” The man pages in general comprise a reference manual. They are not intended to be a tutorial.

Trusted Solaris Reference Manual

In the AnswerBook2™ and online man command forms of the man pages, all man pages are available:

- Trusted Solaris man pages that are unique for the Trusted Solaris environment
- SunOS 5.8 man pages that have been changed in the Trusted Solaris environment
- SunOS 5.8 man pages that remain unchanged.

The printed manual, the *Trusted Solaris 8 Reference Manual* contains:

- Man pages that have been added to the SunOS operating system by the Trusted Solaris environment
- Man pages that originated in SunOS 5.8, but have been modified in the Trusted Solaris environment to handle security requirements.

Users of printed manuals need both manuals in order to have a full set of man pages, since the *SunOS 5.8 Reference Manual* contains the common man pages that are not modified in the Trusted Solaris environment.

Man Page Sections

The following contains a brief description of each section in the man pages and the information it references:

- Section 1 describes, in alphabetical order, commands available with the operating system.
- Section 1M describes, in alphabetical order, commands that are used chiefly for system maintenance and administration purposes.
- Section 2 describes all of the system calls. Most of these calls have one or more error returns. An error condition is indicated by an otherwise impossible returned value.
- Section 3 describes functions found in various libraries, other than those functions that directly invoke UNIX system primitives, which are described in Section 2 of this volume.
- Section 4 outlines the formats of various files. The C structure declarations for the file formats are given where applicable.
- Section 5 contains miscellaneous documentation such as character set tables.
- Section 6 contains available games and demos.
- Section 7 describes various special files that refer to specific hardware peripherals, and device drivers. STREAMS software drivers, modules and the STREAMS-generic set of system calls are also described.
- Section 9 provides reference information needed to write device drivers in the kernel operating systems environment. It describes two device driver interface specifications: the Device Driver Interface (DDI) and the Driver/Kernel Interface (DKI).
- Section 9E describes the DDI/DKI, DDI-only, and DKI-only entry-point routines a developer may include in a device driver.
- Section 9F describes the kernel functions available for use by device drivers.
- Section 9S describes the data structures used by drivers to share information between the driver and the kernel.

Below is a generic format for man pages. The man pages of each manual section generally follow this order, but include only needed headings. For example, if there are no bugs to report, there is no BUGS section. See the `intro` pages for more information and detail about each section, and `man(1)` for more information about man pages in general.

NAME

This section gives the names of the commands or functions documented, followed by a brief description of what they do.

SYNOPSIS

This section shows the syntax of commands or functions. When a command or file does not exist in the standard path, its full pathname is shown. Options and

arguments are alphabetized, with single letter arguments first, and options with arguments next, unless a different argument order is required.

The following special characters are used in this section:

- [] The option or argument enclosed in these brackets is optional. If the brackets are omitted, the argument must be specified.
- . . . Ellipses. Several values may be provided for the previous argument, or the previous argument can be specified multiple times, for example, 'filename...'.
"filename...".
- | Separator. Only one of the arguments separated by this character can be specified at a time.
- { } Braces. The options and/or arguments enclosed within braces are interdependent, such that everything enclosed must be treated as a unit.

PROTOCOL

This section occurs only in subsection 3R to indicate the protocol description file.

DESCRIPTION

This section defines the functionality and behavior of the service. Thus it describes concisely what the command does. It does not discuss OPTIONS or cite EXAMPLES. Interactive commands, subcommands, requests, macros, functions and such, are described under USAGE.

IOCTL

This section appears on pages in Section 7 only. Only the device class which supplies appropriate parameters to the `ioctl` (2) system call is called `ioctl` and generates its own heading. `ioctl` calls for a specific device are listed alphabetically (on the man page for that specific device). `ioctl` calls are used for a particular class of devices all of which have an `io` ending, such as `mtio`(7I)

OPTIONS

This section lists the command options with a concise summary of what each option does. The options are listed literally and in the order they appear in the SYNOPSIS section. Possible arguments to options are discussed under the option, and where appropriate, default values are supplied.

OPERANDS

This section lists the command operands and describes how they affect the actions of the command.

OUTPUT

This section describes the output – standard output, standard error, or output files – generated by the command.

RETURN VALUES

If the man page documents functions that return values, this section lists these values and describes the conditions under which they are returned. If a function can return only constant values, such as 0 or -1, these values are listed in tagged

paragraphs. Otherwise, a single paragraph describes the return values of each function. Functions declared void do not return values, so they are not discussed in RETURN VALUES.

ERRORS

On failure, most functions place an error code in the global variable `errno` indicating why they failed. This section lists alphabetically all error codes a function can generate and describes the conditions that cause each error. When more than one condition can cause the same error, each condition is described in a separate paragraph under the error code.

USAGE

This section lists special rules, features, and commands that require in-depth explanations. The subsections listed here are used to explain built-in functionality:

- Commands
- Modifiers
- Variables
- Expressions
- Input Grammar

EXAMPLES

This section provides examples of usage or of how to use a command or function. Wherever possible a complete example including command-line entry and machine response is shown. Whenever an example is given, the prompt is shown as `example%`, or if the user must be root, `example#`. Examples are followed by explanations, variable substitution rules, or returned values. Most examples illustrate concepts from the SYNOPSIS, DESCRIPTION, OPTIONS, and USAGE sections.

ENVIRONMENT VARIABLES

This section lists any environment variables that the command or function affects, followed by a brief description of the effect.

EXIT STATUS

This section lists the values the command returns to the calling program or shell and the conditions that cause these values to be returned. Usually, zero is returned for successful completion, and values other than zero for various error conditions.

FILES

This section lists all file names referred to by the man page, files of interest, and files created or required by commands. Each is followed by a descriptive summary or explanation.

ATTRIBUTES

This section lists characteristics of commands, utilities, and device drivers by defining the attribute type and its corresponding value. See `attributes(5)` for more information.

SUMMARY OF TRUSTED SOLARIS CHANGES

This section describes changes to a Solaris item by Trusted Solaris software. It is present in man pages that have been modified from Solaris software.

SEE ALSO

This section lists references to other man pages, in-house documentation and outside publications. The references are divided into two sections, so that users of printed manuals can easily locate a man page in its appropriate printed manual.

DIAGNOSTICS

This section lists diagnostic messages with a brief explanation of the condition causing the error.

WARNINGS

This section lists warnings about special conditions which could seriously affect your working conditions. This is not a list of diagnostics.

NOTES

This section lists additional information that does not belong anywhere else on the page. It takes the form of an aside to the user, covering points of special interest. Critical information is never covered here.

BUGS

This section describes known bugs and, wherever possible, suggests workarounds.

Introduction

Intro(2)

NAME	Intro – introduction to system calls and error numbers
SYNOPSIS	<code>#include <errno.h></code>
DESCRIPTION	<p>This section describes all of the system calls in the Trusted Solaris environment.</p> <p>Trusted Solaris system calls are one of the following:</p> <ul style="list-style-type: none"> ■ Calls that are unique to and originate in the Trusted Solaris environment, such as <code>seconf(2)</code>. The <code>seconf()</code> system call allows processes to determine the value of a configurable security-related system variable, such as the variable that hides upgraded file names when set. ■ SunOS 5.8 system calls that have been modified to work within Trusted Solaris security policy, such as <code>link(2)</code>. Man pages for modified system calls have been rewritten to remove information that is not accurate for how the system call behaves within the Trusted Solaris environment. Modified man pages also have added descriptions for new features and arguments. ■ SunOS 5.8 system calls that remain unchanged from the Solaris 8 release, such as <code>exit(2)</code>. <p>Note – The printed <i>Trusted Solaris 8 4/01 Reference Manual</i> includes only those system calls that have been modified or originate in the Trusted Solaris environment. Printed versions of unchanged SunOS 5.8 man pages are found in the <i>SunOS 5.8 Reference Manual</i>. See Trusted Solaris Manual Page Display in Intro(1) for more discussion.</p> <p>When a man page for a system call states that the calling process must have or must assert a specified <i>privilege</i> or <i>privileges</i>, that means:</p> <ul style="list-style-type: none"> ■ The privilege(s) must be made available as <i>allowed</i> privileges on the executable, and ■ The privileges must be made available to the effective privilege set of the process in either of these two ways: <ul style="list-style-type: none"> ■ By inheritance from the parent process, or ■ As <i>forced</i> privileges assigned to the executable program. See <i>Process Privilege Sets and Inheritable Privileges</i> in the <i>DEFINITIONS</i> section, and see also the <i>Trusted Solaris Developer's Guide</i> for more complete descriptions of the topics mentioned here.
ERRORS	<p>Most of these calls return one or more error conditions. An error condition is indicated by an otherwise impossible return value. This is almost always <code>-1</code> or the null pointer; the individual descriptions specify the details. An error number is also made available in the external variable <code>errno</code>, which is not cleared on successful calls, so it should be tested only after an error has been indicated.</p> <p>In the case of multithreaded applications, the <code>_REENTRANT</code> flag must be defined on the command line at compilation time (<code>-D_REENTRANT</code>). When the <code>_REENTRANT</code> flag</p>

is defined, `errno` becomes a macro which enables each thread to have its own `errno`. This `errno` macro can be used on either side of the assignment, just as if it were a variable.

Applications should use bound threads rather than the `_lwp_*()` functions (see `thr_create(3THR)`). Using LWP's (lightweight processes) directly is not advised because libraries are only safe to use with threads, not LWP's.

Each system call description attempts to list all possible error numbers. The following is a complete list of the error numbers and their names as defined in `<errno.h>`.

1 EPERM	Appropriate privilege not asserted
	Typically this error indicates an attempt to modify a file in some way forbidden except to its owner or a process with the appropriate privilege. It is also returned for attempts by ordinary users to do things that always require a privilege. See <i>Privilege</i> in the DEFINITIONS section.
2 ENOENT	No such file or directory
	A file name is specified and the file should exist but doesn't, or one of the directories in a path name does not exist.
3 ESRCH	No such process, LWP, or thread
	No process can be found in the system that corresponds to the specified PID, <code>LWPID_t</code> , or <code>thread_t</code> .
4 EINTR	Interrupted system call
	An asynchronous signal (such as interrupt or quit), which the user has elected to catch, occurred during a system service function. If execution is resumed after processing the signal, it will appear as if the interrupted function call returned this error condition.
	In a multithreaded application, <code>EINTR</code> may be returned whenever another thread or LWP calls <code>fork(2)</code> .
5 EIO	I/O error
	Some physical I/O error has occurred. This error may in some cases occur on a call following the one to which it actually applies.
6 ENXIO	No such device or address

Intro(2)

	I/O on a special file refers to a subdevice which does not exist, or exists beyond the limit of the device. It may also occur when, for example, a tape drive is not on-line or no disk pack is loaded on a drive.
7 E2BIG	Arg list too long An argument list longer than ARG_MAX bytes is presented to a member of the exec family of functions (see exec(2)). The argument list limit is the sum of the size of the argument list plus the size of the environment's exported shell variables.
8 ENOEXEC	Exec format error A request is made to execute a file which, although it has the appropriate permissions, does not start with a valid format (see a.out(4)).
9 EBADF	Bad file number Either a file descriptor refers to no open file, or a read(2) (respectively, write(2)) request is made to a file that is open only for writing (respectively, reading).
10 ECHILD	No child processes A wait(2) function was executed by a process that had no existing or unwaited-for child processes.
11 EAGAIN	No more processes, or no more LWPs For example, the fork(2) function failed because the system's process table is full or the user is not allowed to create any more processes, or a call failed because of insufficient memory or swap space.
12 ENOMEM	Not enough space During execution of brk() or sbrk() (see brk(2)), or one of the exec family of functions, a program asks for more space than the system is able to supply. This is not a temporary condition; the maximum size is a system parameter. On some architectures, the error may also occur if the arrangement of text, data, and stack segments requires too many segmentation registers, or if there is not enough swap space during the fork(2) function. If this error occurs on a resource associated with Remote File Sharing (RFS), it indicates

	a memory depletion which may be temporary, dependent on system activity at the time the call was invoked.
13 EACCES	<p>Permission denied</p> <p>An attempt was made to access a file in a way forbidden by the Trusted Solaris security policy. This type of failure due to DAC or MAC restrictions may be bypassed at the discretion of the security administrator if the appropriate override privilege(s) are made available to be asserted by the calling process (which privilege to use depends on the type of access being denied). See <i>Discretionary Access Control</i>, <i>File Access</i>, <i>Mandatory Access Control</i>, <i>Privilege</i>, and <i>Security Policy</i> in the DEFINITIONS section.</p>
14 EFAULT	<p>Bad address</p> <p>The system encountered a hardware fault in attempting to use an argument of a routine. For example, <code>errno</code> potentially may be set to EFAULT any time a routine that takes a pointer argument is passed an invalid address, if the system can detect the condition. Because systems will differ in their ability to reliably detect a bad address, on some implementations passing a bad address to a routine will result in undefined behavior.</p>
15 ENOTBLK	<p>Block device required</p> <p>A non-block device or file was mentioned where a block device was required (for example, in a call to the <code>mount(2)</code> function).</p>
16 EBUSY	<p>Device busy</p> <p>An attempt was made to mount a device that was already mounted or an attempt was made to unmount a device on which there is an active file (open file, current directory, mounted-on file, active text segment). It will also occur if an attempt is made to enable accounting when it is already enabled. The device or resource is currently unavailable. EBUSY is also used by mutexes, semaphores, condition variables, and r/w locks, to indicate that a lock is held, and by the processor control function <code>P_ONLINE</code>.</p>
17 EEXIST	File exists

Intro(2)

	An existing file was mentioned in an inappropriate context (for example, call to the <code>link(2)</code> function).
18 EXDEV	Cross-device link
	A hard link to a file on another device was attempted.
19 ENODEV	No such device
	An attempt was made to apply an inappropriate operation to a device (for example, read a write-only device).
20 ENOTDIR	Not a directory
	A non-directory was specified where a directory is required (for example, in a path prefix or as an argument to the <code>chdir(2)</code> function).
21 EISDIR	Is a directory
	An attempt was made to write on a directory.
22 EINVAL	Invalid argument
	An invalid argument was specified (for example, unmounting a non-mounted device), mentioning an undefined signal in a call to the <code>signal(3C)</code> or <code>kill(2)</code> function.
23 ENFILE	File table overflow
	The system file table is full (that is, <code>SYS_OPEN</code> files are open, and temporarily no more files can be opened).
24 EMFILE	Too many open files
	No process may have more than <code>OPEN_MAX</code> file descriptors open at a time.
25 ENOTTY	Inappropriate <code>ioctl</code> for device
	A call was made to the <code>ioctl(2)</code> function specifying a file that is not a special character device.
26 ETXTBSY	Text file busy (obsolete)
	An attempt was made to execute a pure-procedure program that is currently open for writing. Also an attempt to open for writing or to remove a pure-procedure program that is being executed. (<i>This message is obsolete.</i>)
27 EFBIG	File too large

	<p>The size of the file exceeded the limit specified by resource <code>RLIMIT_FSIZE</code>; the file size exceeds the maximum supported by the file system; or the file size exceeds the offset maximum of the file descriptor. See the <code>File Descriptor</code> subsection of the <code>DEFINITIONS</code> section below.</p>
28 ENOSPC	<p>No space left on device</p> <p>While writing an ordinary file or creating a directory entry, there is no free space left on the device. In the <code>fcntl(2)</code> function, the setting or removing of record locks on a file cannot be accomplished because there are no more record entries left on the system.</p>
29 EPIPE	<p>Illegal seek</p> <p>A call to the <code>lseek(2)</code> function was issued to a pipe.</p>
30 EROFS	<p>Read-only file system</p> <p>An attempt to modify a file or directory was made on a device mounted read-only.</p>
31 EMLINK	<p>Too many links</p> <p>An attempt to make more than the maximum number of links, <code>LINK_MAX</code>, to a file.</p>
32 EPIPE	<p>Broken pipe</p> <p>A write on a pipe for which there is no process to read the data. This condition normally generates a signal; the error is returned if the signal is ignored.</p>
33 EDOM	<p>Math argument out of domain of func</p> <p>The argument of a function in the math package (3M) is out of the domain of the function.</p>
34 ERANGE	<p>Math result not representable</p> <p>The value of a function in the math package (3M) is not representable within machine precision.</p>
35 ENOMSG	<p>No message of desired type</p> <p>An attempt was made to receive a message of a type that does not exist on the specified message queue (see <code>msgrcv(2)</code>).</p>
36 EIDRM	<p>Identifier removed</p>

Intro(2)

	This error is returned to processes that resume execution due to the removal of an identifier from the file system's name space (see <code>msgctl(2)</code> , <code>semctl(2)</code> , and <code>shmctl(2)</code>).
37 ECHRNG	Channel number out of range
38 EL2NSYNC	Level 2 not synchronized
39 EL3HLT	Level 3 halted
40 EL3RST	Level 3 reset
41 ELNRNG	Link number out of range
42 EUNATCH	Protocol driver not attached
43 ENOCSI	No CSI structure available
44 EL2HLT	Level 2 halted
45 EDEADLK	Deadlock condition
	A deadlock situation was detected and avoided. This error pertains to file and record locking, and also applies to mutexes, semaphores, condition variables, and r/w locks.
46 ENOLCK	No record locks available
	There are no more locks available. The system lock table is full (see <code>fcntl(2)</code>).
47 ECANCELED	Operation canceled
	The associated asynchronous operation was canceled before completion.
48 ENOTSUP	Not supported
	This version of the system does not support this feature. Future versions of the system may provide support.
49 EDQUOT	Disc quota exceeded
	A <code>write(2)</code> to an ordinary file, the creation of a directory or symbolic link, or the creation of a directory entry failed because the user's quota of disk blocks was exhausted, or the allocation of an inode for a newly created file failed because the user's quota of inodes was exhausted.
58-59	Reserved

60 ENOSTR	<p>Device not a stream</p> <p>A <code>putmsg(2)</code> or <code>getmsg(2)</code> call was attempted on a file descriptor that is not a STREAMS device.</p>
61 ENODATA	No data available
62 ETIME	<p>Timer expired</p> <p>The timer set for a STREAMS <code>ioctl(2)</code> call has expired. The cause of this error is device-specific and could indicate either a hardware or software failure, or perhaps a timeout value that is too short for the specific operation. The status of the <code>ioctl()</code> operation is indeterminate. This is also returned in the case of <code>_lwp_cond_timedwait(2)</code> or <code>_lwp_cond_wait(2)</code>.</p>
63 ENOSR	<p>Out of stream resources</p> <p>During a STREAMS <code>open(2)</code> call, either no STREAMS queues or no STREAMS head data structures were available. This is a temporary condition; one may recover from it if other processes release resources.</p>
64 ENONET	<p>Machine is not on the network</p> <p>This error is Remote File Sharing (RFS) specific. It occurs when users try to advertise, unadvertise, mount, or unmount remote resources while the machine has not done the proper startup to connect to the network.</p>
65 ENOPKG	<p>Package not installed</p> <p>This error occurs when users attempt to use a call from a package which has not been installed.</p>
66 EREMOTE	<p>Object is remote</p> <p>This error is RFS-specific. It occurs when users try to advertise a resource which is not on the local machine, or try to mount/unmount a device (or pathname) that is on a remote machine.</p>
67 ENOLINK	<p>Link has been severed</p> <p>This error is RFS-specific. It occurs when the link (virtual circuit) connecting to a remote machine is gone.</p>
68 EADV	<p>Advertise error</p> <p>This error is RFS-specific. It occurs when users try to advertise a resource which has been advertised already,</p>

	or try to stop RFS while there are resources still advertised, or try to force unmount a resource when it is still advertised.
69 ESRMNT	Srmount error
	This error is RFS-specific. It occurs when an attempt is made to stop RFS while resources are still mounted by remote machines, or when a resource is readadvertised with a client list that does not include a remote machine that currently has the resource mounted.
70 ECOMM	Communication error on send
	This error is RFS-specific. It occurs when the current process is waiting for a message from a remote machine, and the virtual circuit fails.
71 EPROTO	Protocol error
	Some protocol error occurred. This error is device-specific, but is generally not related to a hardware failure.
76 EDOTDOT	Error 76
	This error is RFS-specific. A way for the server to tell the client that a process has transferred back from mount point.
77 EBADMSG	Not a data message
	During a <code>read(2)</code> , <code>getmsg(2)</code> , or <code>ioctl(2)</code> <code>I_RECVFD</code> call to a STREAMS device, something has come to the head of the queue that can not be processed. That something depends on the call:
	<code>read()</code> : control information or passed file descriptor.
	<code>getmsg()</code> : passed file descriptor.
	<code>ioctl()</code> : control or data information.
78 ENAMETOOLONG	File name too long
	The length of the path argument exceeds <code>PATH_MAX</code> , or the length of a path component exceeds <code>NAME_MAX</code> while <code>_POSIX_NO_TRUNC</code> is in effect; see <code>limits(4)</code> .
79 EOVERFLOW	Value too large for defined data type.
80 ENOTUNIQ	Name not unique on network

	Given log name not unique.
81 EBADFD	File descriptor in bad state
	Either a file descriptor refers to no open file or a read request was made to a file that is open only for writing.
82 EREMCHG	Remote address changed
83 ELIBACC	Cannot access a needed share library
	Trying to <code>exec</code> an <code>a.out</code> that requires a static shared library and the static shared library does not exist or the user does not have permission to use it.
84 ELIBBAD	Accessing a corrupted shared library
	Trying to <code>exec</code> an <code>a.out</code> that requires a static shared library (to be linked in) and <code>exec</code> could not load the static shared library. The static shared library is probably corrupted.
85 ELIBSCN	<code>.lib</code> section in <code>a.out</code> corrupted
	Trying to <code>exec</code> an <code>a.out</code> that requires a static shared library (to be linked in) and there was erroneous data in the <code>.lib</code> section of the <code>a.out</code> . The <code>.lib</code> section tells <code>exec</code> what static shared libraries are needed. The <code>a.out</code> is probably corrupted.
86 ELIBMAX	Attempting to link in more shared libraries than system limit
	Trying to <code>exec</code> an <code>a.out</code> that requires more static shared libraries than is allowed on the current configuration of the system. See <i>NFS Administration Guide</i>
87 ELIBEXEC	Cannot <code>exec</code> a shared library directly
	Attempting to <code>exec</code> a shared library directly.
88 EILSEQ	Error 88
	Illegal byte sequence. Handle multiple characters as a single character.
89 ENOSYS	Operation not applicable
90 ELOOP	Number of symbolic links encountered during path name traversal exceeds <code>MAXSYMLINKS</code>
91 ESTART	Restartable system call

Intro(2)

	Interrupted system call should be restarted.
92 ESTRPIPE	If pipe/FIFO, don't sleep in stream head
	Streams pipe error (not externally visible).
93 ENOTEMPTY	Directory not empty
94 EUSERS	Too many users
95 ENOTSOCK	Socket operation on non-socket
96 EDESTADDRREQ	Destination address required
	A required address was omitted from an operation on a transport endpoint. Destination address required.
97 EMGSIZE	Message too long
	A message sent on a transport provider was larger than the internal message buffer or some other network limit.
98 EPROTOTYPE	Protocol wrong type for socket
	A protocol was specified that does not support the semantics of the socket type requested.
99 ENOPROTOOPT	Protocol not available
	A bad option or level was specified when getting or setting options for a protocol.
120 EPROTONOSUPPORT	Protocol not supported
	The protocol has not been configured into the system or no implementation for it exists.
121 ESOCKTNOSUPPORT	Socket type not supported
	The support for the socket type has not been configured into the system or no implementation for it exists.
122 EOPNOTSUPP	Operation not supported on transport endpoint
	For example, trying to accept a connection on a datagram transport endpoint.
123 EPFNOSUPPORT	Protocol family not supported
	The protocol family has not been configured into the system or no implementation for it exists. Used for the Internet protocols.

124 EAFNOSUPPORT	Address family not supported by protocol family An address incompatible with the requested protocol was used.
125 EADDRINUSE	Address already in use User attempted to use an address already in use, and the protocol does not allow this.
126 EADDRNOTAVAIL	Cannot assign requested address Results from an attempt to create a transport endpoint with an address not on the current machine.
127 ENETDOWN	Network is down Operation encountered a dead network.
128 ENETUNREACH	Network is unreachable Operation was attempted to an unreachable network.
129 ENETRESET	Network dropped connection because of reset The host you were connected to crashed and rebooted.
130 ECONNABORTED	Software caused connection abort A connection abort was caused internal to your host machine.
131 ECONNRESET	Connection reset by peer A connection was forcibly closed by a peer. This normally results from a loss of the connection on the remote host due to a timeout or a reboot.
132 ENOBUFS	No buffer space available An operation on a transport endpoint or pipe was not performed because the system lacked sufficient buffer space or because a queue was full.
133 EISCONN	Transport endpoint is already connected A connect request was made on an already connected transport endpoint; or, a <code>sendto(3SOCKET)</code> or <code>sendmsg(3SOCKET)</code> request on a connected transport endpoint specified a destination when already connected.
134 ENOTCONN	Transport endpoint is not connected

Intro(2)

		A request to send or receive data was disallowed because the transport endpoint is not connected and (when sending a datagram) no address was supplied.
	143 ESHUTDOWN	Cannot send after transport endpoint shutdown
		A request to send data was disallowed because the transport endpoint has already been shut down.
	144 ETOOMANYREFS	Too many references: cannot splice
	145 ETIMEDOUT	Connection timed out
		A <code>connect(3SOCKET)</code> or <code>send(3SOCKET)</code> request failed because the connected party did not properly respond after a period of time; or a <code>write(2)</code> or <code>fsync(3C)</code> request failed because a file is on an NFS file system mounted with the <i>soft</i> option.
	146 ECONNREFUSED	Connection refused
		No connection could be made because the target machine actively refused it. This usually results from trying to connect to a service that is inactive on the remote host.
	147 EHOSTDOWN	Host is down
		A transport provider operation failed because the destination host was down.
	148 EHOSTUNREACH	No route to host
		A transport provider operation was attempted to an unreachable host.
	149 EALREADY	Operation already in progress
		An operation was attempted on a non-blocking object that already had an operation in progress.
	150 EINPROGRESS	Operation now in progress
		An operation that takes a long time to complete (such as a <code>connect ()</code>) was attempted on a non-blocking object.
	151 ESTALE	Stale NFS file handle
ACL	See <i>Access Control List</i>	

ACL Mask	Created for compatibility purposes, it masks out any existing ACL entries without destroying them when <code>chmod(1)</code> changes permissions on a file or directory. The masked names can then later be restored if <code>chmod</code> is run again to restore the original permissions.
Access Control List	A type of <i>discretionary access control</i> based on a list of entries that the owner can specify for a file or directory. The access control list (ACL) restricts or permits access to any number of individuals and groups, allowing finer-grained control than provided by the standard UNIX <i>permission bits</i> .
Accreditation Range	Actually not a range, but a set made up of labels. See <i>user accreditation range</i> and <i>system accreditation range</i> for more about the two types of accreditation ranges in the Trusted Solaris environment.
Background Process Group	Any process group that is not the foreground process group of a session that has established a connection with a controlling terminal.
CMW Label	A wrapper structure for a <i>sensitivity label</i> that contains other internal and obsolete information. This structure is part of the Trusted Solaris ABI.
Classification	The hierarchical portion of a <i>sensitivity label</i> or <i>clearance</i> , each of which has only one classification. Each classification has an external name (text string) and an internal number (integer), with the lowest number assigned to the lowest classification and the other numbers assigned to the rest of the classifications in a hierarchical relationship. In a sensitivity label assigned to a file or directory, a classification indicates a relative level of protection based on the sensitivity of the information contained in the file or directory. In a clearance assigned to a user and that user's processes, a classification indicates a level of trust.
Clearance	Each process has a clearance associated with it. A <i>clearance</i> consists of a <i>classification</i> and a set of <i>compartments</i> . It is similar to a <i>sensitivity label</i> . A process' clearance is an upper bound on the labels to which the process has access. A process can neither set its sensitivity label to a label that dominates its clearance, nor access an object (file or other process) whose sensitivity label dominates the process clearance.
Compartment	A word associated with one or more compartment bits that may be defined in the <code>label_encodings(4)</code> file to be part of a <i>sensitivity label</i> or <i>clearance</i> . Compartments represent areas of interest or work groups associated with the labels that contain compartments and are used in MAC decisions. Compartments have no intrinsic ordering; however, the <code>label_encodings</code> file can impose constraints that may be hierarchical on the allowable combinations of compartments with each other and with <i>classifications</i> .
Controlling Process	A session leader that established a connection to a controlling terminal.
Controlling Terminal	A terminal that is associated with a session. Each session may have, at most, one controlling terminal associated with it and a controlling terminal may be associated

Intro(2)

	with only one session. Certain input sequences from the controlling terminal cause signals to be sent to process groups in the session associated with the controlling terminal; see <code>termio(7I)</code> .
DAC	See <i>discretionary access control</i> .
Device Objects	Device objects include printers, workstations, tape drives, floppy drives, audio devices, and internal pseudo terminal devices. See <i>mandatory access control</i> for definitions of MAC policy. Devices are subject to the read-equal-write-equal policy.
Directory	Directories organize files into a hierarchical system where directories are the nodes in the hierarchy. A directory is a file that catalogs the list of files, including directories (sub-directories), that are directly beneath it in the hierarchy. Entries in a directory file are called links. A link associates a file identifier with a filename. By convention, a directory contains at least two links, <code>.</code> (dot) and <code>..</code> (dot-dot). The link called dot refers to the directory itself while dot-dot refers to its parent directory. The root directory, which is the top-most node of the hierarchy, has itself as its parent directory. The pathname of the root directory is <code>/</code> and the parent directory of the root directory is <code>/</code> .
Discretionary Access Control	The type of access granted or denied by the owner of a file or directory at the discretion of the owner. The Trusted Solaris environment provides two kinds of discretionary access (DAC) controls, <i>permission bits</i> and <i>access control lists</i> .
Disjoint	When two labels of any type (<i>sensitivity label</i> or <i>clearance</i>) are compared and neither of the two labels <i>dominates</i> the other, the labels are said to be disjoint. Information flow between disjoint labels is considered to be a downgrade.
Dominate	When any type of label (<i>sensitivity label</i> or <i>clearance</i>) has a security level equal to or greater than the security level of another label to which it is being compared, the first label is said to dominate the second. The classification of the dominant label must equal or be higher than the classification of the second label, and the dominant label must include all the words (compartments and markings, if present) in the other label. Sensitivity labels are compared for dominance when MAC decisions are being made. See <i>strictly dominate</i> and <i>disjoint</i> .
Downstream	In a stream, the direction from stream head to driver.
Driver	In a stream, the driver provides the interface between peripheral hardware and the stream. A driver can also be a pseudo-driver, such as a multiplexor or log driver (see <code>log(7D)</code>), which is not associated with a hardware device.
Effective User ID and Effective Group ID	An active process has an effective user ID and an effective group ID that are used to determine file access permissions (see below). The effective user ID and effective group ID are equal to the process's real user ID and real group ID, respectively, unless the process or one of its ancestors evolved from a file that had the set-user-ID bit or set-group-ID bit set (see <code>exec(2)</code>).
File Access	Even though, strictly speaking, files, directories, devices and other objects are treated as files in the UNIX system, only the access rules for <i>file system objects</i> are described

in this section. Because files, directories, and devices have slightly different mandatory access rules, these rules are separately described. See *process objects*, *System V IPC objects*, *STREAMS objects*, *network endpoint objects*, *device objects*, and *X window objects* for the rules that apply to these other types of objects.

A file, directory, or device may be accessed in three ways:

- The *name* of the file, directory, or device may be *viewed*,
- The *contents* or the *attributes* of the file, directory, or device may be *viewed*, or
- The *contents* or the *attributes* of the file, directory, or device may be *modified*.

In the Trusted Solaris environment, each of these types of access is granted or denied based on whether certain *discretionary access control* checks (described in *File Access Permissions*) and *mandatory access control* checks have been passed.

All types of access require that the *sensitivity label* of the process *dominates* the sensitivity label of all directories in the path prefix and that the owner of the process has discretionary access for each directory in the path prefix. View access to the name of the file, directory, or device requires only that this part of the check is passed (unless the system is configured to hide upgraded names).

For view access (read access) to the contents or attributes of a file or directory, the process' sensitivity label must dominate the sensitivity label of the file or directory. For view access to the contents of a device (for example, so you can read information on a tape in a tape drive), the process' sensitivity label must be equal to the sensitivity label of the device. The owner of the process also must have discretionary read access to the file, directory, or device.

For a process to write a file or to modify its attributes, the sensitivity label of the file must dominate the sensitivity label of the process and must be dominated by the process' clearance. (See *process clearance*.) For a process to write into a directory (to *create* a file or a symbolic link) the label of the process must equal the sensitivity label of the directory. For a process to write to a device (for example, store information on a tape in a tape drive), the sensitivity label of the process must equal the sensitivity label of the device. The security policy for device files can differ from the policy for regular files based on how the policy is defined in the `device_policy(4)` file, which can be changed by the security administrator. The owner of the process must have discretionary write access to the file, directory, or device.

For each type of failure of a MAC or DAC check, a specific override *privilege* may be asserted by the process, depending on the type of access being denied. See *process privilege sets*, and *inheritable privileges*.

These conditions and the listed override privileges apply to any type of access:

- If the sensitivity label of the process does not dominate the sensitivity label of a directory in the path prefix, then the process must assert the privilege to search up (search a directory whose sensitivity label dominates the process' sensitivity label), which is `PRIV_FILE_MAC_SEARCH`.

Intro(2)

- If the user on whose behalf the process is being executed does not assert discretionary search permission for a directory in the path prefix, then the process must have the privilege to override DAC search restrictions when accessing a directory, which is `PRIV_FILE_DAC_SEARCH`.

These conditions and the listed override privileges apply to view (read) access to a file or directory or to its attributes:

- If the sensitivity label of the process does not dominate the sensitivity label of the file or directory, then the process must assert the privilege to override MAC read restrictions, which is `PRIV_FILE_MAC_READ`.
- If the user on whose behalf the process is being executed does not have discretionary read permission for the file or directory, then the process must assert the privilege to override DAC read restrictions, which is `PRIV_FILE_DAC_READ`.

These conditions and the listed override privileges apply to modify (write) access to a file or directory or to its attributes:

- If the sensitivity label of the file does not dominate or if the sensitivity of the directory or device does not equal the sensitivity label of the process, and if the sensitivity label of the file, directory, or device is not dominated by the process' clearance, the process must assert the privilege that overrides MAC write restrictions, allowing the user to write up and to write above the process' clearance, which is `PRIV_FILE_MAC_WRITE`.
- If the user on whose behalf the process is being executed does not have discretionary write permission for the file or directory, then the process must assert the privilege to override DAC write restrictions, which is `PRIV_FILE_DAC_WRITE`.

File Access Permissions

Read, write, and execute/search permissions for a file are granted to a process if one or more of the following are true:

- If the effective UID of the process matches the UID of the file, access is granted if allowed by the file's "owner" permission set.
- If the effective UID of the process matches an ACL user entry, access is granted if allowed by the ACL entry and the ACL mask.
- If the effective GID of the process (or one of its supplemental groups) matches the GID of the file or the group ID of any ACL group entry, a permission set is computed as the inclusive OR of all matching group permission sets, specified as follows:
 - If the effective GID of the process (or one of its supplemental groups) matches the GID of the file and there is no owning group ACL entry for the file, the "group" permission set is considered a matching group permission set.
 - If the effective GID of the process (or one of its supplemental groups) matches the GID of the file and there is an owning group ACL entry for the file, the permissions set of that ACL entry is considered a matching group permission set.

	<ul style="list-style-type: none"> ■ If the effective GID of the process (or one of its supplemental groups) matches an ACL group entry, the permission set of that ACL entry is considered a matching group permission set. Access is granted if allowed by the computed permission set and the ACL mask, if any. ■ If none of the preceding cases applies, access is granted if allowed by the file's "other" permission set and the ACL mask, if any. <p>A process is granted access if it satisfies the appropriate test above or if it asserts the DAC override privilege corresponding to the desired operation. Otherwise, access is denied.</p>
File Descriptor	<p>A file descriptor is a small integer used to perform I/O on a file. The value of a file descriptor is from 0 to (NOFILES-1). A process may have no more than NOFILES file descriptors open simultaneously. A file descriptor is returned by calls such as <code>open(2)</code> or <code>pipe(2)</code>. The file descriptor is used as an argument by calls such as <code>read(2)</code>, <code>write(2)</code>, <code>ioctl(2)</code>, and <code>close(2)</code>.</p> <p>Each file descriptor has a corresponding offset maximum. For regular files that were opened without setting the <code>O_LARGEFILE</code> flag, the offset maximum is 2 Gbyte - 1 byte ($2^{31} - 1$ bytes). For regular files that were opened with the <code>O_LARGEFILE</code> flag set, the offset maximum is $2^{63} - 1$ bytes.</p>
File Name	<p>Names consisting of 1 to NAME_MAX characters may be used to name an ordinary file, special file or directory.</p> <p>These characters may be selected from the set of all character values excluding <code>\0</code> (null) and the ASCII code for <code>/</code> (slash).</p> <p>Note that it is generally unwise to use <code>*</code>, <code>?</code>, <code>[</code>, or <code>]</code> as part of file names because of the special meaning attached to these characters by the shell (see <code>sh(1)</code>, <code>csh(1)</code>, and <code>ksh(1)</code>). Although permitted, the use of unprintable characters in file names should be avoided.</p> <p>A file name is sometimes referred to as a pathname component. The interpretation of a pathname component is dependent on the values of <code>NAME_MAX</code> and <code>_POSIX_NO_TRUNC</code> associated with the path prefix of that component. If any pathname component is longer than <code>NAME_MAX</code> and <code>_POSIX_NO_TRUNC</code> is in effect for the path prefix of that component (see <code>fpathconf(2)</code> and <code>limits(4)</code>), it shall be considered an error condition in that implementation. Otherwise, the implementation shall use the first <code>NAME_MAX</code> bytes of the pathname component.</p>
File Privilege Sets	<p>These sets consist of the allowed and forced privileges specified for use by executable files (programs). The allowed set limits which privileges a process can use, whether the privileges are forced on the executable file or inherited (see <i>inheritable privileges</i>). Any privileges in the forced privilege set are available to any process that invokes the program, as long as they are also in the allowed set.</p>
File System Objects	<p>File-system objects include files (regular files, process files, and device-special files), directories, symbolic links, FIFOs (named pipes), pipes, and UNIX domain socket</p>

Intro(2)

	rendezvous. See <i>mandatory access control</i> for definitions of the MAC policies. See <i>File Access</i> for the MAC rules that apply to regular files, device files, symbolic links, and directories. The policies for the remaining file objects are as follows. UNIX domain socket rendezvous and FIFOs (named pipes) are subject to the write up read down policy. Pipes are subject to the read-equal-write-equal policy.
Foreground Process Group	Each session that has established a connection with a controlling terminal will distinguish one process group of the session as the foreground process group of the controlling terminal. This group has certain privileges when accessing its controlling terminal that are denied to background process groups.
Inheritable Privileges	The privileges that a process can pass to a program across an <code>execve(2)</code> without their being affected by the new program's forced or allowed privilege sets. (A child process created through a <code>fork(2)</code> receives all of a parent process' privilege sets with no change.) When a new program is executed by a process, the inheritable set of the process is set to be equal to the inheritable set of the old program: $I[\text{process}] = I[\text{program}]$. The inheritable set is not affected by the forced or allowed privileges on the currently executing program, which allows privileges to be passed from programs that cannot use them to programs that can.
{IOV_MAX}	Maximum number of entries in a <code>struct iovec</code> array.
Label	A security identifier assigned to an object based on the level of protection it needs and to a process based on the degree of trust afforded to the user on whose behalf the process is running.
Label Range	A set of sensitivity labels assigned to allocatable devices, commands and file systems, specified by designating a maximum label and a minimum label. For allocatable devices, the minimum and maximum labels limit the sensitivity labels at which devices may be allocated. See <code>allocate(1)</code> . For commands, the minimum and maximum labels limit the sensitivity labels at which the command may be executed. For file systems, the minimum and maximum labels limit the sensitivity labels at which information may be stored on each file system.
Label View Flags	The label view process attribute flags control the translation and display of the internal <code>admin low</code> and <code>admin high</code> labels. A value of <code>External</code> specifies that the <code>admin low</code> and <code>admin high</code> labels are mapped to the lowest and highest labels defined in the <code>label_encodings(4)</code> file. A value of <code>Internal</code> specifies that the <code>admin low</code> and <code>admin high</code> labels are translated to the respective strings specified in the <code>label_encodings</code> file. If no such names are specified, the strings <code>ADMIN_LOW</code> and <code>ADMIN_HIGH</code> are used. If no value is set, the default label view specified in the <code>label_encodings</code> file is used.
Label Translation Flags	These fifteen-bit flags support the <code>GFI FLAGS=</code> option in the <code>label_encodings(4)</code> file, which allows the use of these flags by applications written to use them. These flags are viewable and modifiable only by a trusted path process.
{LIMIT}	The braces notation, <code>{LIMIT}</code> , is used to denote a magnitude limitation imposed by the implementation. This indicates a value which may be defined by a header file

	(without the braces), or the actual value may be obtained at runtime by a call to the configuration inquiry <code>pathconf(2)</code> with the name argument <code>_PC_LIMIT</code> .
MAC	See <i>mandatory access control</i> .
MLD	See <i>multilevel directory</i> .
Mandatory Access Control	A type of control based on comparing the sensitivity label of an object to the sensitivity label of the process that is trying to access the object. The MAC policies that apply to various types of objects are <i>read equal</i> , <i>write equal</i> , <i>read down</i> , and <i>write up</i> . (See the individual definitions for each object type for the policy that applies.) When the <i>read equal</i> policy applies, an object may be accessed for reading only when the sensitivity label of the process is equal to the sensitivity label of the object. When the <i>write equal</i> policy applies, an object may be accessed for writing only when the sensitivity label of the process is equal to the sensitivity label of the object. When the <i>write up</i> policy applies, an object may be accessed for writing only when the sensitivity label of the process is dominated by the sensitivity label of the object, hence the process "writes up" to the object. The write up policy also includes write-equal. When the <i>read down</i> policy applies, an object may be accessed for reading only when the sensitivity label of the process dominates the sensitivity label of the object, hence the process "reads down" to the object. The read-down policy also includes read-equal.
Masks	The file mode creation mask of the process used during any create function calls to turn off permission bits in the <i>mode</i> argument supplied. Bit positions that are set in <code>umask</code> (<i>cmask</i>) are cleared in the mode of the created file.
Message	In a stream, one or more blocks of data or information, with associated STREAMS control structures. Messages can be of several defined types, which identify the message contents. Messages are the only means of transferring data and communicating within a stream.
Message Queue	In a stream, a linked list of messages awaiting processing by a module or driver.
Message Queue Identifier	<p>A message queue identifier (<code>msqid</code>) is a unique positive integer created by a <code>msgget(2)</code> call. Each <code>msqid</code> has a message queue and a data structure associated with it. The data structure is referred to as <code>msqid_ds</code> and contains the following members:</p> <pre> struct ipc_perm msg_perm; struct msg *msg_first; struct msg *msg_last; ulong_t msg_cbytes; ulong_t msg_qnum; ulong_t msg_qbytes; pid_t msg_lspid; pid_t msg_lrpid; time_t msg_stime; time_t msg_rtime; time_t msg_ctime; </pre> <p>The following are descriptions of the <code>msqid_ds</code> structure members:</p>

Intro(2)

The `msg_perm` member is an `ipc_perm` structure that specifies the message operation permission (see below). This structure includes the following members:

```
uid_t    cuid;    /* creator user id */
gid_t    cgid;    /* creator group id */
uid_t    uid;     /* user id */
gid_t    gid;     /* group id */
mode_t    mode;   /* r/w permission */
ulong_t  seq;     /* slot usage sequence # */
key_t    key;     /* key */
```

The `*msg_first` member is a pointer to the first message on the queue.

The `*msg_last` member is a pointer to the last message on the queue.

The `msg_cbytes` member is the current number of bytes on the queue.

The `msg_qnum` member is the number of messages currently on the queue.

The `msg_qbytes` member is the maximum number of bytes allowed on the queue.

The `msg_lspid` member is the process ID of the last process that performed a `msgsnd()` operation.

The `msg_lrpid` member is the process id of the last process that performed a `msgrcv()` operation.

The `msg_stime` member is the time of the last `msgsnd()` operation.

The `msg_rtime` member is the time of the last `msgrcv()` operation.

The `msg_ctime` member is the time of the last `msgctl()` operation that changed a member of the above structure.

Message Operation Permissions

In the `msgctl(2)`, `msgget(2)`, `msgrcv(2)`, and `msgsnd(2)` function descriptions, the permission required for an operation is given as `{token}`, where *token* is the type of permission needed, interpreted as follows:

```
00400  READ by user
00200  WRITE by user
00040  READ by group
00020  WRITE by group
00004  READ by others
00002  WRITE by others
```

Read and write permissions on a `msqid` are granted to a process if the read-equal-write-equal mandatory access control check is passed or if the process asserts the appropriate override privilege (either `PRIV_IPC_MAC_READ` or `PRIV_IPC_MAC_WRITE`), and if one of the following tests is true:

- The effective user ID of the process matches `msg_perm.cuid` or `msg_perm.uid` in the data structure associated with `msqid` and the appropriate bit of the “user” portion (0600) of `msg_perm.mode` is set.

	<ul style="list-style-type: none"> ■ Any group ID in the process credentials from the set (<code>cr_gid</code>, <code>cr_groups</code>) matches <code>msg_perm.cgid</code> or <code>msg_perm.gid</code> and the appropriate bit of the “group” portion (060) of <code>msg_perm.mode</code> is set. ■ The appropriate bit of the “other” portion (006) of <code>msg_perm.mode</code> is set. ■ The process has asserted the appropriate DAC privilege, either <code>PRIV_FILE_DAC_READ</code> or <code>PRIV_FILE_DAC_WRITE</code>. <p>Otherwise, the corresponding permissions are denied.</p>
Module	A module is an entity containing processing routines for input and output data. It always exists in the middle of a stream, between the stream’s head and a driver. A module is the STREAMS counterpart to the commands in a shell pipeline except that a module contains a pair of functions which allow independent bidirectional (downstream and upstream) data flow and processing.
Multilevel Directory	A directory in which information at differing sensitivity labels is maintained in separate subdirectories called <i>single-level directories</i> (SLDs), while appearing to most interfaces to be a single directory under a single name. In the Trusted Solaris environment, directories that are used by multiple standard applications to store files at varying labels, such as the <code>/tmp</code> directory, <code>/var/spool/mail</code> , and users’ <code>\$HOME</code> directories are set up to be MLDs. A process can access an MLD two ways: either by using pathname translation, or by using the adorned name. When a process refers to an MLD without the adorned name, the Trusted Solaris process transparently extends the reference to the SLD that corresponds to the process’ sensitivity label. If the process is creating a file and if the correct SLD does not already exist, the Trusted Solaris process creates the SLD and assigns it the process’ sensitivity label so that the correct single-level directory exists for the file. If the process wants to access the MLD directly, it should use the the MLD adornment on the final component of the path. The text string <code>.MLD.</code> is the default adornment. The adornment is a file system attribute that may be changed using <code>setfsattr(1M)</code> . Use of the adornment allows programs to refer directly to the MLD instead of to the SLD that has the same SL as the process.
Multiplexor	A multiplexor is a driver that allows streams associated with several user processes to be connected to a single driver, or several drivers to be connected to a single user process. STREAMS does not provide a general multiplexing driver, but does provide the facilities for constructing them and for connecting multiplexed configurations of streams.
Network Endpoint Objects	Network endpoint objects are sockets and the transport level interface (TLI). See <i>mandatory access control</i> for definitions of the MAC policies. Network endpoint objects are subject to the read-equal-write-equal policy.
Object	Anything in the Trusted Solaris environment that a process attempts to access. The six major object types are file system objects, process objects, System V IPC objects, STREAMS objects, network endpoint objects, device objects, and X window objects.
Offset Maximum	An offset maximum is an attribute of an open file description representing the largest value that can be used as a file offset.

Intro(2)

Orphaned Process Group	A process group in which the parent of every member in the group is either itself a member of the group, or is not a member of the process group's session.
Path Name	<p>A path name is a null-terminated character string starting with an optional slash (/), followed by zero or more directory names separated by slashes, optionally followed by a file name.</p> <p>If a path name begins with a slash, the path search begins at the root directory. Otherwise, the search begins from the current working directory.</p> <p>A slash by itself names the root directory.</p> <p>Unless specifically stated otherwise, the null path name is treated as if it named a non-existent file.</p>
Process ID	Each process in the system is uniquely identified during its lifetime by a positive integer called a process ID. A process ID may not be reused by the system until the process lifetime, process group lifetime, and session lifetime ends for any process ID, process group ID, and session ID equal to that process ID. Within a process, there are threads with thread id's, called <code>thread_t</code> and <code>LWPID_t</code> . These threads are not visible to the outside process.
Parent Process ID	A new process is created by a currently active process (see <code>fork(2)</code>). The parent process ID of a process is the process ID of its creator.
Privilege	Having appropriate privilege means having the capability to override some aspect of <i>security policy</i> . If a man page states that a system call needs to have or to assert "the appropriate privilege" to bypass DAC or MAC restrictions, see <i>File Access Permissions</i> for the override privilege that applies to the type of access being denied. A privilege is only granted by a site's security administrator after judging that the command itself or the person will use the privilege in a trustworthy manner. See <i>File Privilege Sets</i> and <i>Process Privilege Sets</i> .
Privilege Debugging Flag	This one-bit flag indicates that the process is in privilege debugging mode, an operational mode where any attempt by the process to use a privilege is logged. This flag can be viewed or cleared, but can be set only by a trusted path process. This flag is set by <code>runpd(1M)</code> when executing a command in privilege debugging mode, and then is inherited by the process. It works only if the <code>_PRIVS_DEBUG</code> kernel switch is also enabled (see <code>secconf(2)</code>).
Process Attribute Flags	Trusted Solaris flags that indicate security-related values that are copied from one process to another on <code>fork(2)</code> and cloned without changes on <code>exec(2)</code> . They are: the Trusted Path Flag, the Privilege Debugging Flag, the Network Token Mapping Process Flag, the Label View Flag (External View or Internal View), the Label Translation Flags, the Diskless Boot Flag, the Cut and Paste Selection Agent Flag, the Trusted System Printing flag, and the Automount flag. See <code>pattr(1)</code> , <code>getpattr(2)</code> , and <code>setpattr(2)</code> . Each flag has its own protection policy. Any process may view or clear any process attributes flags except for the Label Translation flags, which are viewable and clearable by only a process with the trusted path attribute. Any process may set

	the Label View flags, but only processes with the trusted path attribute may set any of the other process attribute flags.
Process Group	Each process in the system is a member of a process group that is identified by a process group ID. Any process that is not a process group leader may create a new process group and become its leader. Any process that is not a process group leader may join an existing process group that shares the same session as the process. A newly created process joins the process group of its parent.
Process Group Leader	A process group leader is a process whose process ID is the same as its process group ID.
Process Group ID	Each active process is a member of a process group and is identified by a positive integer called the process group ID. This ID is the process ID of the group leader. This grouping permits the signaling of related processes (see <code>kill(2)</code>).
Process Lifetime	A process lifetime begins when the process is forked and ends after it exits, when its termination has been acknowledged by its parent process. See <code>wait(2)</code> .
Process Group Lifetime	A process group lifetime begins when the process group is created by its process group leader, and ends when the lifetime of the last process in the group ends or when the last process in the group leaves the group.
Process Objects	Process and lightweight processes (independently scheduled threads of execution), which are subject to the write-up-read-down policy. See <i>object</i> for definitions of the MAC policies.
Process Privilege Sets	The privileges used by a process are stored in sets called the <i>inheritable</i> , <i>permitted</i> , <i>effective</i> , and <i>saved</i> sets. When a process executes a program through the <code>execve(2)</code> system call, the permitted (P) and effective (E) privilege sets are reset equal to the same value, which is the intersection of the process' previously existing inheritable (I) privileges and the program file's allowed (A) privileges intersected with the program file's forced (F) privileges: $P=E=(I[\text{process}] \cup F[\text{program}] \text{ restricted by } A[\text{program}])$. The saved privilege set is set initially to the intersection of the existing inheritable privilege set and the file's allowed privileges: $S=(I[\text{process}] \cap A[\text{program}])$, which allows the process to determine which privileges it had when the currently executing program was invoked. When a new program is invoked, the inheritable privilege set is initially set to be the same as the inheritable privileges of the process that invoked the current program: $I[\text{new}] = I[\text{old}]$. Setting the inheritable privileges without reference to the forced or allowed privileges on an executing program allows privileges to be passed without change from a program that cannot use them to one that can. For compatibility with the Solaris operating environment's super-user capability, if the effective UID is set by <code>setuid(2)</code> to be different from the original, the effective set is copied to the saved set and the effective set is cleared: $S=E$; $E=0$. When the process changes its effective user ID back to the original, the saved privilege set is copied to the effective set, thus restoring its privileged state: $E=S$. In addition to automatic changes in privilege sets as the result of <code>execve()</code> or <code>setuid()</code> , a process may manipulate its own privilege sets with the

Intro(2)

	<p>getppriv(2) and setppriv(2) system calls. For example, a process can use these calls to move permitted privileges into and out of its effective privilege set, for privilege bracketing. A process with the PRIV_SET_FPRIV privilege in its effective set can use setfpriv(2) to set privileges on a file. See the <i>Trusted Solaris Developer's Guide</i> for more details about how privileges may be manipulated within programs using system calls.</p>
Process Security Attributes	<p>Security attributes received by processes from the Solaris operating environment are: the process ID (PID), the real, effective, and saved user ID, the real, effective, or saved group ID, the supplementary group IDs, the user audit ID, the audit session ID, the audit preselection mask, the terminal ID, and the umask (see <i>Masks</i>). Security attributes received by processes from the Trusted Solaris system are: the process clearance, the CMW <i>label</i>, the process attribute flags, and the permitted, effective, inheritable, and saved process privilege sets.</p>
Processor Set ID	<p>The processors in a system may be divided into subsets, known as processor sets. A process bound to one of these sets will run only on processors in that set, and the processors in the set will normally run only processes that have been bound to the set. Each active processor set is identified by a positive integer. See pset_create(2).</p>
Read Queue	<p>In a stream, the message queue in a module or driver containing messages moving upstream.</p>
Real User ID and Real Group ID	<p>Each user allowed on the system is identified by a positive integer (0 to MAXUID) called a real user ID.</p> <p>Each user is also a member of a group. The group is identified by a positive integer called the real group ID.</p> <p>An active process has a real user ID and real group ID that are set to the real user ID and real group ID, respectively, of the user responsible for the creation of the process.</p>
Root Directory and Current Working Directory	<p>Each process has associated with it a concept of a root directory and a current working directory for the purpose of resolving path name searches. The root directory of a process need not be the root directory of the root file system.</p>
Saved Resource Limits	<p>Saved resource limits is an attribute of a process that provides some flexibility in the handling of unrepresentable resource limits, as described in the exec family of functions and setrlimit(2).</p>
Saved User ID and Saved Group ID	<p>The saved user ID and saved group ID are the values of the effective user ID and effective group ID prior to an exec of a file whose set user or set group file mode bit has been set (see exec(2)).</p>
SLD	<p>See <i>single-level directory</i>.</p>
Security Attribute	<p>An attribute used in enforcing the Trusted Solaris security policy. Various sets of security attributes, both from the Solaris operating environment and the Trusted</p>

Security Policy

Solaris operating environment, are assigned to processes, users, files, directories, file systems, hosts on the trusted network, allocatable devices, and other entities. See *Process Security Attributes*.

In the Trusted Solaris environment, the set of rules for DAC, MAC, and privilege interpretation that define how information may be accessed. At a customer site, the set of rules that define the sensitivity of the information being processed at that site and the measures that are used to protect the information from unauthorized access.

Semaphore Identifier

A semaphore identifier (`semid`) is a unique positive integer created by a `semget(2)` call. Each `semid` has a set of semaphores and a data structure associated with it. The data structure is referred to as `semid_ds` and contains the following members:

```
struct ipc_perm sem_perm;    /* operation permission struct */$
struct sem      *sem_base;   /* ptr to first semaphore in set */$
ushort_t        sem_nsems;    /* number of sems in set */$
time_t          sem_otime;    /* last operation time */$
time_t          sem_ctime;    /* last change time */$
/* Times measured in secs since */$
/* 00:00:00 GMT, Jan. 1, 1970 */$
```

The following are descriptions of the `semid_ds` structure members:

The `sem_perm` member is an `ipc_perm` structure that specifies the semaphore operation permission (see below). This structure includes the following members:

```
uid_t    uid;    /* user id */
gid_t    gid;    /* group id */
uid_t    cuid;   /* creator user id */
gid_t    cgid;   /* creator group id */
mode_t    mode;  /* r/a permission */
ulong_t  seq;    /* slot usage sequence number */
key_t    key;    /* key */
```

The `sem_nsems` member is equal to the number of semaphores in the set. Each semaphore in the set is referenced by a nonnegative integer referred to as a `sem_num`. `sem_num` values run sequentially from 0 to the value of `sem_nsems` minus 1.

The `sem_otime` member is the time of the last `semop(2)` operation.

The `sem_ctime` member is the time of the last `semctl(2)` operation that changed a member of the above structure.

A semaphore is a data structure called `sem` that contains the following members:

```
ushort_t  semval;    /* semaphore value */
pid_t     sempid;    /* pid of last operation */
ushort_t  semncnt;   /* # awaiting semval > cval */
ushort_t  semzcnt;   /* # awaiting semval = 0 */
```

The following are descriptions of the `sem` structure members:

The `semval` member is a non-negative integer that is the actual value of the semaphore.

Intro(2)

Semaphore Operation Permissions

The `sempid` member is equal to the process ID of the last process that performed a semaphore operation on this semaphore.

The `semncnt` member is a count of the number of processes that are currently suspended awaiting this semaphore's `semval` to become greater than its current value.

The `semzcnt` member is a count of the number of processes that are currently suspended awaiting this semaphore's `semval` to become 0.

In the `semop(2)` and `semctl(2)` function descriptions, the permission required for an operation is given as `{token}`, where `token` is the type of permission needed interpreted as follows:

```
00400  READ by user
00200  ALTER by user
00040  READ by group
00020  ALTER by group
00004  READ by others
00002  ALTER by others
```

Read and alter permissions on a `semid` are granted to a process if the read-equal-write-equal mandatory access control check is passed or if the process asserts the appropriate override privilege (either `PRIV_IPC_MAC_READ` or `PRIV_IPC_MAC_WRITE`), and if one of the following tests is true.

- The effective user ID of the process matches `sem_perm.cuid` or `sem_perm.uid` in the data structure associated with `semid` and the appropriate bit of the "user" portion (0600) of `sem_perm.mode` is set.
- The effective group ID of the process matches `sem_perm.cgid` or `sem_perm.gid` and the appropriate bit of the "group" portion (060) of `sem_perm.mode` is set.
- The appropriate bit of the "other" portion (06) of `sem_perm.mode` is set.
- The process has asserted the appropriate DAC privilege, either `PRIV_FILE_DAC_READ` or `PRIV_FILE_DAC_WRITE`.

Otherwise, the corresponding permissions are denied.

Sensitivity Label

A *sensitivity label* defines the level of protection afforded to a labeled object or the level of access granted a labeled subject. Sensitivity labels are used in all mandatory access control (MAC) decisions by the Trusted Solaris environment. A sensitivity label consists of a hierarchical classification and a set of non-hierarchical compartments. This classification-and-compartments pair is known as the level of the sensitivity label.

Session

A session is a group of processes identified by a common ID called a session ID, capable of establishing a connection with a controlling terminal. Any process that is not a process group leader may create a new session and process group, becoming the session leader of the session and process group leader of the process group. A newly created process joins the session of its creator.

Session ID	Each session in the system is uniquely identified during its lifetime by a positive integer called a session ID, the process ID of its session leader.
Session Leader	A session leader is a process whose session ID is the same as its process and process group ID.
Session Lifetime	A session lifetime begins when the session is created by its session leader, and ends when the lifetime of the last process that is a member of the session ends, or when the last process that is a member in the session leaves the session.
Shared Memory Identifier	<p>A shared memory identifier (shmid) is a unique positive integer created by a <code>semget(2)</code> call. Each shmid has a segment of memory (referred to as a shared memory segment) and a data structure associated with it. (Note that these shared memory segments must be explicitly removed by the user after the last reference to them is removed.) The data structure is referred to as <code>shmid_ds</code> and contains the following members:</p> <pre> struct ipc_perm shm_perm; /* operation permission struct */ int shm_segsz; /* size of segment */ struct region *shm_reg; /* ptr to region structure */ char pad[4]; /* for swap compatibility */ pid_t shm_lpid; /* pid of last operation */ pid_t shm_cpid; /* creator pid */ ushort_t shm_nattch; /* number of current attaches */ ushort_t shm_cnattch; /* used only for shminfo */ time_t shm_atime; /* last attach time */ time_t shm_dtime; /* last detach time */ time_t shm_ctime; /* last change time */ /* Times measured in secs since */ /* 00:00:00 GMT, Jan. 1, 1970 */ </pre> <p>The following are descriptions of the <code>shmid_ds</code> structure members:</p> <p>The <code>shm_perm</code> member is an <code>ipc_perm</code> structure that specifies the shared memory operation permission (see below). This structure includes the following members:</p> <pre> uid_t cuid; /* creator user id */ gid_t cgid; /* creator group id */ uid_t uid; /* user id */ gid_t gid; /* group id */ mode_t mode; /* r/w permission */ ulong_t seq; /* slot usage sequence # */ key_t key; /* key */ </pre> <p>The <code>shm_segsz</code> member specifies the size of the shared memory segment in bytes.</p> <p>The <code>shm_cpid</code> member is the process ID of the process that created the shared memory identifier.</p> <p>The <code>shm_lpid</code> member is the process ID of the last process that performed a <code>shmat()</code> or <code>shmdt()</code> operation (see <code>shmop(2)</code>).</p>

Intro(2)

Shared Memory Operation Permissions

The `shm_nattch` member is the number of processes that currently have this segment attached.

The `shm_atime` member is the time of the last `shmat()` operation (see `shmop(2)`).

The `shm_dtime` member is the time of the last `shmdt()` operation (see `shmop(2)`).

The `shm_ctime` member is the time of the last `shmctl(2)` operation that changed one of the members of the above structure.

In the `shmctl(2)`, `shmat()`, and `shmdt()` (see `shmop(2)`) function descriptions, the permission required for an operation is given as `{token}`, where *token* is the type of permission needed interpreted as follows:

```
00400  READ by user
00200  WRITE by user
00040  READ by group
00020  WRITE by group
00004  READ by others
00002  WRITE by others
```

Read and write permissions for a `shmid` are granted to a process if the read-equal-write-equal mandatory access control check is passed or if the process asserts the appropriate override privilege (either `PRIV_IPC_MAC_READ` or `PRIV_IPC_MAC_WRITE`), and if one of the following tests is true:

- The effective user ID of the process is super-user.
- The effective user ID of the process matches `shm_perm.cuid` or `shm_perm.uid` in the data structure associated with `shmid` and the appropriate bit of the “user” portion (0600) of `shm_perm.mode` is set.
- The effective group ID of the process matches `shm_perm.cgid` or `shm_perm.gid` and the appropriate bit of the “group” portion (060) of `shm_perm.mode` is set.
- The appropriate bit of the “other” portion (06) of `shm_perm.mode` is set.
- The process has asserted the appropriate DAC privilege, either `PRIV_FILE_DAC_READ` or `PRIV_FILE_DAC_WRITE`.

Otherwise, the corresponding permissions are denied.

Single-Level Directory

A directory within an MLD containing only files at a single sensitivity label. The SLD directory name is derived from the SL of the process that created it. For example, the name of an SLD in `/tmp` would be in the form `/tmp/.SLD.<sensitivity_label_of_creating_process>/. All subsequent references to the file in the /tmp directory would be made transparently as /tmp/file. Because pathname translation is transparent, the process would not need to explicitly reference the SLD directory, unless it chose to do so using the MLD adornment and the name of the SLD.`

Special Processes	The process with ID 0 and the process with ID 1 are special processes referred to as <code>proc0</code> and <code>proc1</code> ; see <code>kill(2)</code> . <code>proc0</code> is the process scheduler. <code>proc1</code> is the initialization process (<i>init</i>); <code>proc1</code> is the ancestor of every other process in the system and is used to control the process structure.
STREAMS	A set of kernel mechanisms that support the development of network services and data communication drivers. It defines interface standards for character input/output within the kernel and between the kernel and user level processes. The STREAMS mechanism is composed of utility routines, kernel facilities and a set of data structures.
Stream	A stream is a full-duplex data path within the kernel between a user process and driver routines. The primary components are a stream head, a driver, and zero or more modules between the stream head and driver. A stream is analogous to a shell pipeline, except that data flow and processing are bidirectional.
Stream Head	In a stream, the stream head is the end of the stream that provides the interface between the stream and a user process. The principal functions of the stream head are processing STREAMS-related system calls and passing data and information between a user process and the stream.
Strictly Dominate	When any type of label (<i>sensitivity label</i> or <i>clearance</i>) has a security level greater than the security level of another label to which it is being compared, the first label strictly dominates the second label. Strict dominance is dominance without equality, which occurs either when the classification of the first label is higher than that of the second label and the first label contains all the second label's compartments, or when the classifications of both labels are the same while the first label contains all the compartments in the second label plus one or more additional compartments.
System Accreditation Range	The set of all valid (well-formed) labels created according to the rules defined by each site's security administrator in the <code>label_encodings(4)</code> file, plus the two administrative labels that are used in every Trusted Solaris environment, <code>ADMIN_LOW</code> and <code>ADMIN_HIGH</code> .
Superuser	A process is recognized as a superuser process and is granted special privileges, such as immunity from file permissions, if its effective user ID is 0. In the Trusted Solaris environment, superuser is replaced by administrative roles that share responsibility for the environment.
Trusted Path Flag	Also called the Trusted Path Attribute, this one-bit flag indicates that the process is executing in the trusted path.
Upstream	In a stream, the direction from driver to stream head.
Write Queue	In a stream, the message queue in a module or driver containing messages moving downstream.
X Window Objects	X window objects are the windows in the common desktop environment (which is based on the X Window system). See <i>mandatory access control</i> for definitions of the

Intro(2)

MAC policies. Window objects are generally subject to the read-equal-write-equal policy. See the X library man pages (in `/usr/openwin/man/man3`) for exceptions.

System Calls

access(2)

NAME	access – determine accessibility of a file								
SYNOPSIS	<pre>#include <unistd.h> int access(const char *path, int amode);</pre>								
DESCRIPTION	<p>The <code>access()</code> function checks the file named by the pathname pointed to by the <i>path</i> argument for accessibility according to the bit pattern contained in <i>amode</i>, using the real user ID in place of the effective user ID and the real group ID in place of the effective group ID. This allows a setuid process to verify that the user running it would have had permission to access this file.</p> <p>The value of <i>amode</i> is either the bitwise inclusive OR of the access permissions to be checked (<code>R_OK</code>, <code>W_OK</code>, <code>X_OK</code>) or the existence test, <code>F_OK</code>.</p> <p>These constants are defined in <code><unistd.h></code> as follows:</p> <table><tr><td><code>R_OK</code></td><td>Test for read permission.</td></tr><tr><td><code>W_OK</code></td><td>Test for write permission.</td></tr><tr><td><code>X_OK</code></td><td>Test for execute or search permission.</td></tr><tr><td><code>F_OK</code></td><td>Check existence of file</td></tr></table> <p>See <code>intro(2)</code> for additional information about "File Access Permission".</p> <p>If any access permissions are to be checked, each will be checked individually, as described in <code>intro(2)</code>. If the process has appropriate privileges, an implementation may indicate success for <code>X_OK</code> even if none of the execute file permission bits are set.</p>	<code>R_OK</code>	Test for read permission.	<code>W_OK</code>	Test for write permission.	<code>X_OK</code>	Test for execute or search permission.	<code>F_OK</code>	Check existence of file
<code>R_OK</code>	Test for read permission.								
<code>W_OK</code>	Test for write permission.								
<code>X_OK</code>	Test for execute or search permission.								
<code>F_OK</code>	Check existence of file								
RETURN VALUES	If the requested access is permitted, <code>access()</code> succeeds and returns 0. Otherwise, <code>-1</code> is returned and <code>errno</code> is set to indicate the error.								
ERRORS	<p>The <code>access()</code> function will fail if:</p> <table><tr><td><code>EACCES</code></td><td>Permission bits of the file mode do not permit the requested access. The calling process does not have mandatory read, write, execute, or search access to the final component in <i>path</i>. To override this restriction, the calling process may assert one or more of these privileges depending on the value in <i>amode</i>. <code>PRIV_FILE_MAC_WRITE</code>, <code>PRIV_FILE_DAC_WRITE</code>, <code>PRIV_FILE_MAC_READ</code>, <code>PRIV_FILE_DAC_READ</code>, <code>PRIV_FILE_MAC_SEARCH</code> (in the case of a directory), <code>PRIV_FILE_DAC_SEARCH</code>, and <code>PRIV_FILE_DAC_EXECUTE</code>.</td></tr><tr><td><code>EFAULT</code></td><td><i>path</i> points to an illegal address.</td></tr><tr><td><code>EINTR</code></td><td>A signal was caught during the <code>access()</code> function.</td></tr><tr><td><code>ELOOP</code></td><td>Too many symbolic links were encountered in resolving <i>path</i>.</td></tr></table>	<code>EACCES</code>	Permission bits of the file mode do not permit the requested access. The calling process does not have mandatory read, write, execute, or search access to the final component in <i>path</i> . To override this restriction, the calling process may assert one or more of these privileges depending on the value in <i>amode</i> . <code>PRIV_FILE_MAC_WRITE</code> , <code>PRIV_FILE_DAC_WRITE</code> , <code>PRIV_FILE_MAC_READ</code> , <code>PRIV_FILE_DAC_READ</code> , <code>PRIV_FILE_MAC_SEARCH</code> (in the case of a directory), <code>PRIV_FILE_DAC_SEARCH</code> , and <code>PRIV_FILE_DAC_EXECUTE</code> .	<code>EFAULT</code>	<i>path</i> points to an illegal address.	<code>EINTR</code>	A signal was caught during the <code>access()</code> function.	<code>ELOOP</code>	Too many symbolic links were encountered in resolving <i>path</i> .
<code>EACCES</code>	Permission bits of the file mode do not permit the requested access. The calling process does not have mandatory read, write, execute, or search access to the final component in <i>path</i> . To override this restriction, the calling process may assert one or more of these privileges depending on the value in <i>amode</i> . <code>PRIV_FILE_MAC_WRITE</code> , <code>PRIV_FILE_DAC_WRITE</code> , <code>PRIV_FILE_MAC_READ</code> , <code>PRIV_FILE_DAC_READ</code> , <code>PRIV_FILE_MAC_SEARCH</code> (in the case of a directory), <code>PRIV_FILE_DAC_SEARCH</code> , and <code>PRIV_FILE_DAC_EXECUTE</code> .								
<code>EFAULT</code>	<i>path</i> points to an illegal address.								
<code>EINTR</code>	A signal was caught during the <code>access()</code> function.								
<code>ELOOP</code>	Too many symbolic links were encountered in resolving <i>path</i> .								

ENAMETOOLONG	The length of the <i>path</i> argument exceeds PATH_MAX, or a pathname component is longer than NAME_MAX while _POSIX_NO_TRUNC is in effect.
ENOENT	A component of <i>path</i> does not name an existing file or <i>path</i> is an empty string.
ENOLINK	<i>path</i> points to a remote machine and the link to that machine is no longer active.
ENOTDIR	A component of the path prefix is not a directory.
EROFS	Write access is requested for a file on a read-only file system.
The access () function may fail if:	
EINVAL	The value of the <i>amode</i> argument is invalid.
ENAMETOOLONG	Pathname resolution of a symbolic link produced an intermediate result whose length exceeds PATH_MAX.
ETXTBSY	Write access is requested for a pure procedure (shared text) file that is being executed.

USAGE Additional values of *amode* other than the set defined in the description may be valid, for example, if a system has extended access controls.

ATTRIBUTES See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Async-Signal-Safe

SUMMARY OF TRUSTED SOLARIS CHANGES

Appropriate privilege is required to override access checks.

Certain uses of this interface may present a covert channel. If a covert channel is exploited, the execution of the process may be delayed. To avoid this delay, the process may assert the PRIV_PROC_NODELAY privilege.

Trusted Solaris 8
4/01 Reference
Manual
SunOS 5.8
Reference Manual

intro(2), chmod(2), stat(2)

attributes(5)

acct(2)

NAME	acct – Enable or disable process accounting																		
SYNOPSIS	<pre>#include <unistd.h> int acct(const char *path);</pre>																		
DESCRIPTION	<p>The <code>acct()</code> function enables or disables the system process accounting routine. If the routine is enabled, an accounting record will be written in an accounting file for each process that terminates. The termination of a process can be caused by either an <code>exit(2)</code> call or a <code>signal(3C)</code>. The effective privilege set of the process calling <code>acct()</code> must include <code>PRIV_SYS_CONFIG</code>.</p> <p>The <i>path</i> argument points to the pathname of the accounting file, whose file format is described on the <code>acct(3HEAD)</code> manual page.</p> <p>The accounting routine is enabled if <i>path</i> is non-zero and no errors occur during the function. It is disabled if <i>path</i> is <code>(char *) NULL</code> and no errors occur during the function.</p>																		
RETURN VALUES	<p><code>acct()</code> returns:</p> <table><tr><td>0</td><td>On success.</td></tr><tr><td>-1</td><td>On failure, and sets <code>errno</code> to indicate the error.</td></tr></table>	0	On success.	-1	On failure, and sets <code>errno</code> to indicate the error.														
0	On success.																		
-1	On failure, and sets <code>errno</code> to indicate the error.																		
ERRORS	<p>The <code>acct()</code> function will fail if:</p> <table><tr><td>EACCES</td><td>The file named by <i>path</i> is not an ordinary file.</td></tr><tr><td>EBUSY</td><td>An attempt is being made to enable accounting using the same file that is currently being used.</td></tr><tr><td>EFAULT</td><td>The <i>path</i> argument points to an illegal address.</td></tr><tr><td>ELOOP</td><td>Too many symbolic links were encountered in translating <i>path</i>.</td></tr><tr><td>ENAMETOOLONG</td><td>The length of the <i>path</i> argument exceeds <code>PATH_MAX</code>, or the length of a <i>path</i> argument exceeds <code>NAME_MAX</code> while <code>_POSIX_NO_TRUNC</code> is in effect.</td></tr><tr><td>ENOENT</td><td>One or more components of the accounting file pathname do not exist.</td></tr><tr><td>ENOTDIR</td><td>A component of the path prefix is not a directory.</td></tr><tr><td>EPERM</td><td>The effective privilege set of the calling process does not have <code>PRIV_SYS_CONFIG</code>.</td></tr><tr><td>EROFS</td><td>The named file resides on a read-only file system.</td></tr></table>	EACCES	The file named by <i>path</i> is not an ordinary file.	EBUSY	An attempt is being made to enable accounting using the same file that is currently being used.	EFAULT	The <i>path</i> argument points to an illegal address.	ELOOP	Too many symbolic links were encountered in translating <i>path</i> .	ENAMETOOLONG	The length of the <i>path</i> argument exceeds <code>PATH_MAX</code> , or the length of a <i>path</i> argument exceeds <code>NAME_MAX</code> while <code>_POSIX_NO_TRUNC</code> is in effect.	ENOENT	One or more components of the accounting file pathname do not exist.	ENOTDIR	A component of the path prefix is not a directory.	EPERM	The effective privilege set of the calling process does not have <code>PRIV_SYS_CONFIG</code> .	EROFS	The named file resides on a read-only file system.
EACCES	The file named by <i>path</i> is not an ordinary file.																		
EBUSY	An attempt is being made to enable accounting using the same file that is currently being used.																		
EFAULT	The <i>path</i> argument points to an illegal address.																		
ELOOP	Too many symbolic links were encountered in translating <i>path</i> .																		
ENAMETOOLONG	The length of the <i>path</i> argument exceeds <code>PATH_MAX</code> , or the length of a <i>path</i> argument exceeds <code>NAME_MAX</code> while <code>_POSIX_NO_TRUNC</code> is in effect.																		
ENOENT	One or more components of the accounting file pathname do not exist.																		
ENOTDIR	A component of the path prefix is not a directory.																		
EPERM	The effective privilege set of the calling process does not have <code>PRIV_SYS_CONFIG</code> .																		
EROFS	The named file resides on a read-only file system.																		
SUMMARY OF TRUSTED SOLARIS CHANGES SUNOS 5.8 Reference Manual	<p>The effective privilege set of the process calling <code>acct()</code> must include <code>PRIV_SYS_CONFIG</code>.</p> <p><code>exit(2)</code>, <code>signal(3C)</code>, <code>acct(3HEAD)</code></p>																		

NAME	acl, facl – Get or set a file's Access Control List (ACL)						
SYNOPSIS	<pre>#include <sys/acl.h> int acl(char *pathp, int cmd, int nentries, aclent_t *aclbufp); int facl(int fildes, int cmd, int nentries, aclent_t *aclbufp);</pre>						
DESCRIPTION	<p>The <code>acl()</code> and <code>facl()</code> functions get or set the ACL of a file whose name is given by <i>pathp</i> or referenced by the open file descriptor <i>fildes</i>. The <i>nentries</i> argument specifies how many ACL entries fit into buffer <i>aclbufp</i>. The <code>acl()</code> function is used to manipulate ACL on file system objects.</p> <p>The following values for <i>cmd</i> are supported:</p> <table> <tr> <td>SETACL</td><td><i>nentries</i> ACL entries, specified in buffer <i>aclbufp</i>, are stored in the file's ACL. This command can be executed only by a process that has an effective user ID equal to the owner of the file. To override this restriction, the calling process may assert the <code>PRIV_FILE_SETDAC</code> privilege.</td></tr> <tr> <td>GETACL</td><td>Buffer <i>aclbufp</i> is filled with the file's ACL entries. Read access to the file is not required, but all directories in the path name must be searchable.</td></tr> <tr> <td>GETACLCNT</td><td>The number of entries in the file's ACL is returned. Read access to the file is not required, but all directories in the path name must be searchable.</td></tr> </table>	SETACL	<i>nentries</i> ACL entries, specified in buffer <i>aclbufp</i> , are stored in the file's ACL. This command can be executed only by a process that has an effective user ID equal to the owner of the file. To override this restriction, the calling process may assert the <code>PRIV_FILE_SETDAC</code> privilege.	GETACL	Buffer <i>aclbufp</i> is filled with the file's ACL entries. Read access to the file is not required, but all directories in the path name must be searchable.	GETACLCNT	The number of entries in the file's ACL is returned. Read access to the file is not required, but all directories in the path name must be searchable.
SETACL	<i>nentries</i> ACL entries, specified in buffer <i>aclbufp</i> , are stored in the file's ACL. This command can be executed only by a process that has an effective user ID equal to the owner of the file. To override this restriction, the calling process may assert the <code>PRIV_FILE_SETDAC</code> privilege.						
GETACL	Buffer <i>aclbufp</i> is filled with the file's ACL entries. Read access to the file is not required, but all directories in the path name must be searchable.						
GETACLCNT	The number of entries in the file's ACL is returned. Read access to the file is not required, but all directories in the path name must be searchable.						
RETURN VALUES	<p>Upon successful completion, <code>acl()</code> and <code>facl()</code> return 0 if <i>cmd</i> is SETACL. If <i>cmd</i> is GETACL or GETACLCNT, the number of ACL entries is returned. Otherwise, -1 is returned and <code>errno</code> is set to indicate the error.</p> <p>The audit record has multiple events that represent the requested function. For SETACL, the audit record includes the old and new ACLs.</p>						
ERRORS	<p>The <code>acl()</code> function will fail if:</p> <table> <tr> <td>EACCESS</td><td> <p>The caller does not have access to a component of the pathname. To override this restriction, the calling process may assert one or both of these privileges: <code>PRIV_FILE_DAC_SEARCH</code> and <code>PRIV_FILE_MAC_SEARCH</code>.</p> <p>The <i>cmd</i> argument is SETACL and <i>nentries</i> is less than three.</p> <p>The <i>cmd</i> argument is SETACL and the ACL specified in <i>aclbufp</i> is not valid.</p> </td></tr> <tr> <td>EFAULT</td><td>The <i>pathp</i> or <i>aclbufp</i> argument points to an illegal address.</td></tr> <tr> <td>EINVAL</td><td> <p>The <i>cmd</i> argument is not GETACL, SETACL, or GETACLCNT; the <i>cmd</i> argument is SETACL and <i>nentries</i> is less than 3; or the <i>cmd</i> argument is SETACL and the ACL specified in <i>aclbufp</i> is not valid.</p> </td></tr> </table>	EACCESS	<p>The caller does not have access to a component of the pathname. To override this restriction, the calling process may assert one or both of these privileges: <code>PRIV_FILE_DAC_SEARCH</code> and <code>PRIV_FILE_MAC_SEARCH</code>.</p> <p>The <i>cmd</i> argument is SETACL and <i>nentries</i> is less than three.</p> <p>The <i>cmd</i> argument is SETACL and the ACL specified in <i>aclbufp</i> is not valid.</p>	EFAULT	The <i>pathp</i> or <i>aclbufp</i> argument points to an illegal address.	EINVAL	<p>The <i>cmd</i> argument is not GETACL, SETACL, or GETACLCNT; the <i>cmd</i> argument is SETACL and <i>nentries</i> is less than 3; or the <i>cmd</i> argument is SETACL and the ACL specified in <i>aclbufp</i> is not valid.</p>
EACCESS	<p>The caller does not have access to a component of the pathname. To override this restriction, the calling process may assert one or both of these privileges: <code>PRIV_FILE_DAC_SEARCH</code> and <code>PRIV_FILE_MAC_SEARCH</code>.</p> <p>The <i>cmd</i> argument is SETACL and <i>nentries</i> is less than three.</p> <p>The <i>cmd</i> argument is SETACL and the ACL specified in <i>aclbufp</i> is not valid.</p>						
EFAULT	The <i>pathp</i> or <i>aclbufp</i> argument points to an illegal address.						
EINVAL	<p>The <i>cmd</i> argument is not GETACL, SETACL, or GETACLCNT; the <i>cmd</i> argument is SETACL and <i>nentries</i> is less than 3; or the <i>cmd</i> argument is SETACL and the ACL specified in <i>aclbufp</i> is not valid.</p>						

acl(2)

EIO	A disk I/O error has occurred while storing or retrieving the ACL.
ENOENT	A component of the path does not exist.
ENOSPC	The <i>cmd</i> argument is GETACL and <i>nentries</i> is less than the number of entries in the file's ACL, or the <i>cmd</i> argument is SETACL and there is insufficient space in the file system to store the ACL.
ENOTDIR	A component of the path specified by <i>pathp</i> is not a directory, or the <i>cmd</i> argument is SETACL and an attempt is made to set a default ACL on a file type other than a directory.
ENOSYS	The <i>cmd</i> argument is SETACL and the file specified by <i>pathp</i> resides on a file system that does not support ACLs, or the <code>acl ()</code> function is not supported by this implementation.
EPERM	The <i>cmd</i> argument is SETACL and the effective user ID of the caller does not match the owner of the file. To override this restriction, the calling process may assert the <code>PRIV_FILE_SETDAC</code> privilege.
EROFS	The <i>cmd</i> argument is SETACL and the file specified by <i>pathp</i> resides on a file system that is mounted read-only.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

**SUMMARY OF
TRUSTED
SOLARIS
CHANGES**

Appropriate privilege is required to override access or ownership checks.

The audit record has multiple events that represent the requested function. For SETACL the audit record includes the old and new ACLs.

**SunOS 5.8
Reference Manual**

`getfacl(1)`, `setfacl(1)`, `aclcheck(3SEC)`, `aclsort(3SEC)`, `attributes(5)`

NAME	adjtime – Correct the time to allow synchronization of the system clock								
SYNOPSIS	<pre>#include <sys/time.h> int adjtime(struct timeval *delta, struct timeval *olddelta);</pre>								
DESCRIPTION	<p>The <code>adjtime()</code> function adjusts the system's notion of the current time as returned by <code>gettimeofday(3C)</code>, advancing or retarding it by the amount of time specified in the <code>struct timeval</code> pointed to by <i>delta</i>.</p> <p>The adjustment is effected by speeding up (if that amount of time is positive) or slowing down (if that amount of time is negative) the system's clock by some small percentage, generally a fraction of one percent. The time is always a monotonically increasing function. A time correction from an earlier call to <code>adjtime()</code> may not be finished when <code>adjtime()</code> is called again.</p> <p>If <i>delta</i> is 0, then <i>olddelta</i> returns the status of the effects of the previous <code>adjtime()</code> call with no effect on the time correction as a result of this call. If <i>olddelta</i> is not a null pointer, then the structure it points to will contain, upon successful return, the number of seconds and/or microseconds still to be corrected from the earlier call. If <i>olddelta</i> is a null pointer, the corresponding information will not be returned.</p> <p>This call may be used in time servers that synchronize the clocks of computers in a local area network. Such time servers would slow down the clocks of some machines and speed up the clocks of others to bring them to the average network time.</p> <p>The calling process must have the <code>PRIV_SYS_CONFIG</code> privilege in order to adjust the time of day.</p> <p>The adjustment value will be silently rounded to the resolution of the system clock.</p>								
RETURN VALUES	Upon successful completion, <code>adjtime()</code> returns 0. Otherwise, it returns -1 and sets <code>errno</code> to indicate the error.								
ERRORS	<p>The <code>adjtime()</code> function will fail if:</p> <table> <tr> <td>EFAULT</td><td>The <i>delta</i> or <i>olddelta</i> argument points outside the process's allocated address space, or <i>olddelta</i> points to a region of the process's allocated address space that is not writable.</td></tr> <tr> <td>EINVAL</td><td>The <code>tv_usec</code> member of <i>delta</i> is not within valid range (-1000000 to 1000000).</td></tr> <tr> <td>EPERM</td><td>The effective user of the calling process is not super-user.</td></tr> </table> <p>Additionally, the <code>adjtime()</code> function will fail for 32-bit interfaces if:</p> <table> <tr> <td>EOVERFLOW</td><td>The size of the <code>tv_sec</code> member of the <code>timeval</code> structure pointed to by <i>olddelta</i> is too small to contain the correct number of seconds.</td></tr> </table>	EFAULT	The <i>delta</i> or <i>olddelta</i> argument points outside the process's allocated address space, or <i>olddelta</i> points to a region of the process's allocated address space that is not writable.	EINVAL	The <code>tv_usec</code> member of <i>delta</i> is not within valid range (-1000000 to 1000000).	EPERM	The effective user of the calling process is not super-user.	EOVERFLOW	The size of the <code>tv_sec</code> member of the <code>timeval</code> structure pointed to by <i>olddelta</i> is too small to contain the correct number of seconds.
EFAULT	The <i>delta</i> or <i>olddelta</i> argument points outside the process's allocated address space, or <i>olddelta</i> points to a region of the process's allocated address space that is not writable.								
EINVAL	The <code>tv_usec</code> member of <i>delta</i> is not within valid range (-1000000 to 1000000).								
EPERM	The effective user of the calling process is not super-user.								
EOVERFLOW	The size of the <code>tv_sec</code> member of the <code>timeval</code> structure pointed to by <i>olddelta</i> is too small to contain the correct number of seconds.								
SUMMARY OF TRUSTED SOLARIS CHANGES	The calling process must have the <code>PRIV_SYS_CONFIG</code> privilege in order to use this system call.								

adjtime(2)

**SunOS 5.8
Reference Manual**

date(1), gettimeofday(3C)

NAME	audit – write a record to the audit log						
SYNOPSIS	<pre>cc [flags...] file ... -lbsm -lsocket -lnsl -lintl [library...] #include <sys/param.h> #include <bsm/audit.h> int audit(caddr_t record, int length);</pre>						
DESCRIPTION	<p>The <code>audit()</code> function is used to write a record to the system audit log. The data pointed to by <i>record</i> is written to the log after a minimal consistency check, with the <i>length</i> parameter specifying the size of the record in bytes. The data should be a well-formed audit record as described by <code>audit.log(4)</code>.</p> <p>The kernel validates the record header token type and length, and sets the time stamp value before writing the record to the audit log. The kernel does not do any preselection for user-level generated events. If the audit policy is set to include sequence or trailer tokens, the kernel will append them to the record.</p> <p>If the event number is between 2048 and 32767, the calling process must have the <code>PRIV_PROC_AUDIT_TCB</code> privilege in its set of effective privileges. If the event number is between 32768 and 65535, the caller must have the <code>PRIV_PROC_AUDIT_APPL</code> privilege in its set of effective privileges.</p>						
RETURN VALUES	<p><code>audit()</code> returns:</p> <p>0 On success.</p> <p>–1 On failure, and sets <code>errno</code> to indicate the error.</p>						
ERRORS	<p>The <code>audit()</code> function will fail if:</p> <table> <tr> <td>EFAULT</td><td>The <i>record</i> argument points outside the process's allocated address space.</td></tr> <tr> <td>EINVAL</td><td>The record header token ID is invalid or the length is either less than the header token size or greater than <code>MAXAUDITDATA</code>.</td></tr> <tr> <td>EPERM</td><td>The process's effective privilege set does not contain the proper privilege for this operation.</td></tr> </table>	EFAULT	The <i>record</i> argument points outside the process's allocated address space.	EINVAL	The record header token ID is invalid or the length is either less than the header token size or greater than <code>MAXAUDITDATA</code> .	EPERM	The process's effective privilege set does not contain the proper privilege for this operation.
EFAULT	The <i>record</i> argument points outside the process's allocated address space.						
EINVAL	The record header token ID is invalid or the length is either less than the header token size or greater than <code>MAXAUDITDATA</code> .						
EPERM	The process's effective privilege set does not contain the proper privilege for this operation.						
SUMMARY OF TRUSTED SOLARIS CHANGES	<p>This functionality is active only if auditing is enabled. By default, auditing is enabled in the Trusted Solaris environment. See <i>Trusted Solaris Audit Administration Manual</i> for more information.</p> <p>See the DESCRIPTION section for information about which privileges are needed to use this call when the event number being audited is in the application set or the kernel set.</p> <p><code>auditwrite(3TSOL)</code> is the preferred interface for creating audit records in the Trusted Solaris environment.</p>						
ATTRIBUTES	Available only on Trusted Solaris systems with auditing enabled.						

audit(2)

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsr

**Trusted Solaris 8
4/01 Reference
Manual**
**SunOS 5.8
Reference Manual**

auditd(1M), auditon(2), auditsvc(2), getaudit(2), auditwrite(3TSOL),
audit.log(4)
attributes(5)

NAME	auditon – manipulate auditing																										
SYNOPSIS	<pre>cc [flags...] file ... -lbsm -lsocket -lnsl -lintl [library...] #include <sys/param.h> #include <bsm/audit.h> int auditon(int cmd, caddr_t data, int length);</pre>																										
DESCRIPTION	<p>The <code>auditon()</code> function performs various audit subsystem control operations. The <i>cmd</i> argument designates the particular audit control command. The <i>data</i> argument is a pointer to command-specific data. The <i>length</i> argument is the length in bytes of the command-specific data.</p> <p>The following commands are supported:</p> <table> <tr> <td>A_GETCOND</td><td>Return the system audit on/off/disabled condition in the integer long pointed to by <i>data</i>. The following values may be returned:</td></tr> <tr> <td>AUC_AUDITING</td><td>Auditing has been turned on.</td></tr> <tr> <td>AUC_NOAUDIT</td><td>Auditing has been turned off.</td></tr> <tr> <td>AUC_DISABLED</td><td>Auditing package installed, not turned on.</td></tr> <tr> <td>A_SETCOND</td><td>Set the system's audit on/off condition to the value in the integer long to which <i>data</i> points. The following audit states may be set:</td></tr> <tr> <td>AUC_AUDITING</td><td>Turns on audit record generation.</td></tr> <tr> <td>AUC_NOAUDIT</td><td>Turns off audit record generation.</td></tr> <tr> <td>A_GETCLASS</td><td>Return the event to class mapping for the designated audit event. The <i>data</i> argument points to the <code>au_evclass_map</code> structure containing the event number. The preselection class mask is returned in the same structure.</td></tr> <tr> <td>A_SETCLASS</td><td>Set the event class preselection mask for the designated audit event. The <i>data</i> argument points to the <code>au_evclass_map</code> structure containing the event number and class mask.</td></tr> <tr> <td>A_GETKMASK</td><td>Return the kernel preselection mask in the <code>au_mask</code> structure pointed to by <i>data</i>. This is the mask used to preselect non-attributable audit events.</td></tr> <tr> <td>A_SETKMASK</td><td>Set the kernel preselection mask. The <i>data</i> argument points to the <code>au_mask</code> structure containing the class mask. This is the mask used to preselect non-attributable audit events.</td></tr> <tr> <td>A_GETPINFO</td><td>Return the audit ID, preselection mask, terminal ID and audit session ID of the specified process in the <code>auditpinfo</code> structure pointed to by <i>data</i>.</td></tr> <tr> <td>A_SETPMASK</td><td>Set the preselection mask of the specified process. The <i>data</i> argument points to the <code>auditpinfo</code> structure containing the</td></tr> </table>	A_GETCOND	Return the system audit on/off/disabled condition in the integer long pointed to by <i>data</i> . The following values may be returned:	AUC_AUDITING	Auditing has been turned on.	AUC_NOAUDIT	Auditing has been turned off.	AUC_DISABLED	Auditing package installed, not turned on.	A_SETCOND	Set the system's audit on/off condition to the value in the integer long to which <i>data</i> points. The following audit states may be set:	AUC_AUDITING	Turns on audit record generation.	AUC_NOAUDIT	Turns off audit record generation.	A_GETCLASS	Return the event to class mapping for the designated audit event. The <i>data</i> argument points to the <code>au_evclass_map</code> structure containing the event number. The preselection class mask is returned in the same structure.	A_SETCLASS	Set the event class preselection mask for the designated audit event. The <i>data</i> argument points to the <code>au_evclass_map</code> structure containing the event number and class mask.	A_GETKMASK	Return the kernel preselection mask in the <code>au_mask</code> structure pointed to by <i>data</i> . This is the mask used to preselect non-attributable audit events.	A_SETKMASK	Set the kernel preselection mask. The <i>data</i> argument points to the <code>au_mask</code> structure containing the class mask. This is the mask used to preselect non-attributable audit events.	A_GETPINFO	Return the audit ID, preselection mask, terminal ID and audit session ID of the specified process in the <code>auditpinfo</code> structure pointed to by <i>data</i> .	A_SETPMASK	Set the preselection mask of the specified process. The <i>data</i> argument points to the <code>auditpinfo</code> structure containing the
A_GETCOND	Return the system audit on/off/disabled condition in the integer long pointed to by <i>data</i> . The following values may be returned:																										
AUC_AUDITING	Auditing has been turned on.																										
AUC_NOAUDIT	Auditing has been turned off.																										
AUC_DISABLED	Auditing package installed, not turned on.																										
A_SETCOND	Set the system's audit on/off condition to the value in the integer long to which <i>data</i> points. The following audit states may be set:																										
AUC_AUDITING	Turns on audit record generation.																										
AUC_NOAUDIT	Turns off audit record generation.																										
A_GETCLASS	Return the event to class mapping for the designated audit event. The <i>data</i> argument points to the <code>au_evclass_map</code> structure containing the event number. The preselection class mask is returned in the same structure.																										
A_SETCLASS	Set the event class preselection mask for the designated audit event. The <i>data</i> argument points to the <code>au_evclass_map</code> structure containing the event number and class mask.																										
A_GETKMASK	Return the kernel preselection mask in the <code>au_mask</code> structure pointed to by <i>data</i> . This is the mask used to preselect non-attributable audit events.																										
A_SETKMASK	Set the kernel preselection mask. The <i>data</i> argument points to the <code>au_mask</code> structure containing the class mask. This is the mask used to preselect non-attributable audit events.																										
A_GETPINFO	Return the audit ID, preselection mask, terminal ID and audit session ID of the specified process in the <code>auditpinfo</code> structure pointed to by <i>data</i> .																										
A_SETPMASK	Set the preselection mask of the specified process. The <i>data</i> argument points to the <code>auditpinfo</code> structure containing the																										

auditon(2)

	process ID and the preselection mask. The other fields of the structure are ignored and should be set to <code>NULL</code> .								
<code>A_SETUMASK</code>	Set the preselection mask for all processes with the specified audit ID. The <i>data</i> argument points to the <code>auditinfo</code> structure containing the audit ID and the preselection mask. The other fields of the structure are ignored and should be set to <code>NULL</code> .								
<code>A_SETSMASK</code>	Set the preselection mask for all processes with the specified audit session ID. The <i>data</i> argument points to the <code>auditinfo</code> structure containing the audit session ID and the preselection mask. The other fields of the structure are ignored and should be set to <code>NULL</code> .								
<code>A_GETQCTRL</code>	Return the kernel audit queue control parameters. These control the high and low water marks of the number of audit records allowed in the audit queue. The high water mark is the maximum allowed number of undelivered audit records. The low water mark determines when threads blocked on the queue are wakened. Another parameter controls the size of the data buffer used by <code>auditsvc(2)</code> to write data to the audit trail. There is also a parameter that specifies a maximum delay before data is attempted to be written to the audit trail. The audit queue parameters are returned in the <code>au_qctrl</code> structure pointed to by <i>data</i> .								
<code>A_SETQCTRL</code>	Set the kernel audit queue control parameters as described above in the <code>A_GETQCTRL</code> command. The <i>data</i> argument points to the <code>au_qctrl</code> structure containing the audit queue control parameters. The default and maximum values 'A/B' for the audit queue control parameters are: <table> <tr> <td>high water</td><td>100/10000 (audit records)</td></tr> <tr> <td>low water</td><td>10/1024 (audit records)</td></tr> <tr> <td>output buffer size</td><td>1024/1048576 (bytes)</td></tr> <tr> <td>delay</td><td>20/20000 (hundredths second)</td></tr> </table>	high water	100/10000 (audit records)	low water	10/1024 (audit records)	output buffer size	1024/1048576 (bytes)	delay	20/20000 (hundredths second)
high water	100/10000 (audit records)								
low water	10/1024 (audit records)								
output buffer size	1024/1048576 (bytes)								
delay	20/20000 (hundredths second)								
<code>A_GETCWD</code>	Return the current working directory as kept by the audit subsystem. This is a path anchored on the real root, rather than on the active root. The <i>data</i> argument points to a buffer into which the path is copied. The <i>length</i> argument is the length of the buffer.								
<code>A_GETCAR</code>	Return the current active root as kept by the audit subsystem. This path may be used to anchor an absolute path for a path token generated by an application. The <i>data</i> argument points to a buffer into which the path is copied. The <i>length</i> argument is the length of the buffer.								

Policy Flags	A_GETSTAT	Return the system audit statistics in the <code>audit_stat</code> structure pointed to by <i>data</i> .
	A_SETSTAT	Reset system audit statistics values. The kernel statistics value is reset if the corresponding field in the statistics structure pointed to by the <i>data</i> argument is <code>CLEAR_VAL</code> . Otherwise, the value is not changed.
	A_SETFSIZE	Set the maximum size of an audit trail file. When the audit file reaches the designated size, it is closed and a new file started. If the maximum size is unset, the audit trail file generated by <code>auditsvc()</code> will grow to the size of the file system. The <i>data</i> argument points to the <code>au_fstat_t</code> structure containing the maximum audit file size in bytes. The size can not be set less than 0x80000 bytes.
	A_GETFSIZE	Return the maximum audit file size and current file size in the <code>au_fstat_t</code> structure pointed to by the <i>data</i> argument.
	A_GETPOLICY	Return the audit policy flags in the integer long pointed to by <i>data</i> .
	A_SETPOLICY	Set the audit policy flags to the values in the integer long pointed to by <i>data</i> .
	A process must have <code>PRIV_SYS_AUDIT</code> , <code>PRIV_PROC_AUDIT_TCB</code> , or <code>PRIV_PROC_AUDIT_APPL</code> in its set of effective privileges in order to successfully execute these commands: <code>A_GETCOND</code> , <code>A_GETCLASS</code> , <code>A_GETPINFO</code> , <code>A_GETCWD</code> , <code>A_GETCAR</code> , and <code>A_GETPOLICY</code> .	
	A process must have <code>PRIV_SYS_AUDIT</code> in its set of effective privileges in order to successfully execute these commands: <code>A_SETCOND</code> , <code>A_SETCLASS</code> , <code>A_GETKMASK</code> , <code>A_SETKMASK</code> , <code>A_SETPMASK</code> , <code>A_SETUMASK</code> , <code>A_SETSMASK</code> , <code>A_GETQCTRL</code> , <code>A_SETQCTRL</code> , <code>A_GETSTAT</code> , <code>A_SETSTAT</code> , and <code>A_SETPOLICY</code> .	
	AUDIT_ACL	Include in the audit data an ACL attribute for each object accessed. Note that regardless of policy, if there is no ACL associated with an object, an attribute will not be generated. This information is not included by default.
	AUDIT_AHLT	Halt the machine if an asynchronous audit event occurs that cannot be delivered because the audit queue has reached the high-water mark or because there are insufficient resources to construct an audit record.
Policy Flags	AUDIT_CNT	Do not suspend processes when audit storage is full or inaccessible. The default action is to suspend processes until storage becomes available.
	AUDIT_ARGV	Include in the audit record the argument list for the <code>exec(2)</code> system call. The default action is not to include this information.

auditon(2)

	AUDIT_ARGE	Include in the audit record the environment variables for the <code>execv(2)</code> system call. The default action is not to include this information.
	AUDIT_SEQ	Add a sequence token to each audit record. The default action is not to include this token.
	AUDIT_TRAIL	Append a trailer token to each audit record. The default action is not to include this token.
	AUDIT_GROUP	Include the supplementary groups list in audit records. The default action is not to include it.
	AUDIT_SLABEL	Include slabels in audit records. The default action is to include slabels in audit records.
	AUDIT_PASSWD	Include as part of the audit record any bad authentication data encountered during a login operation. The default action is not to include the password in the audit record.
	AUDIT_PATH	Include secondary paths in audit records. Examples of secondary paths are dynamically loaded, shared library modules and the command shell path for executable scripts.
	AUDIT_WINDATA_DOWN	Include in an audit record any downgraded data moved between windows. By default, this data is not included.
	AUDIT_WINDATA_UP	Include in an audit record any upgraded data moved between windows. By default, this data is not included.
RETURN VALUES	auditon() returns:	
	0	On success.
	-1	On failure, and sets <code>errno</code> to indicate the error.
ERRORS	The auditon() function will fail if:	
	E2BIG	The <i>length</i> field for the command was too small to hold the returned value.
	EFAULT	The copy of data to/from the kernel failed.
	EINVAL	One of the system call arguments was illegal..
	EPERM	The process did not have the appropriate privilege in its effective set.
USAGE	The auditon() function may be invoked only by privileged processes.	
SUMMARY OF TRUSTED SOLARIS CHANGES	These policy flags have been added in the Trusted Solaris operating environment: AUDIT_ACL, AUDIT_AHLT, AUDIT_SLABEL, AUDIT_PASSWD,	

	auditon(2)
	AUDIT_WINDATA_DOWN, and AUDIT_WINDATA_UP. The DESCRIPTION section explains which privileges are required to use which audit-control commands.
Trusted Solaris 8 4/01 Reference Manual	auditd(1M), audit(2), auditsvc(2), audit.log(4) <i>Trusted Solaris Audit Administration</i>
SunOS 5.8 Reference Manual	attributes(5)

auditsvc(2)

NAME	auditsvc – Write audit log to specified file descriptor												
SYNOPSIS	<pre>cc [flag...] file ... -lbsm -lsocket -lnsl -lintl [library...] #include <sys/param.h> #include <bsm/audit.h> int auditsvc(int fd, int limit);</pre>												
DESCRIPTION	<p>The <code>auditsvc()</code> function specifies the audit log file to the kernel. The kernel writes audit records to this file until an exceptional condition occurs and then the call returns. The <i>fd</i> argument is a file descriptor that identifies the audit file. Applications should open this file for writing before calling <code>auditsvc()</code>.</p> <p>The <i>limit</i> argument specifies the number of free blocks that must be available in the audit file system, and causes <code>auditsvc()</code> to return when the free disk space on the audit filesystem drops below this limit. Thus, the invoking program can take action to avoid running out of disk space.</p> <p>The <code>auditsvc()</code> function does not return until one of the following conditions occurs:</p> <ul style="list-style-type: none"> ■ The process receives a signal that is not blocked or ignored. ■ An error is encountered writing to the audit log file. ■ The minimum free space (as specified by <i>limit</i>), has been reached. <p>A process must have <code>PRIV_SYS_AUDIT</code> in its set of effective privileges in order to execute this call successfully.</p>												
RETURN VALUES	The <code>auditsvc()</code> function returns only on an error.												
ERRORS	<p>The <code>auditsvc()</code> function will fail if:</p> <table> <tr> <td>EAGAIN</td><td>The descriptor referred to a <i>stream</i>, was marked for System V-style non-blocking I/O, and no data could be written immediately.</td></tr> <tr> <td>EBADF</td><td>The <i>fd</i> argument is not a valid descriptor open for writing.</td></tr> <tr> <td>EBUSY</td><td>A second process attempted to perform this call.</td></tr> <tr> <td>EFBIG</td><td>An attempt was made to write a file that exceeds the process's file size limit or the maximum file size.</td></tr> <tr> <td>EINTR</td><td>The call is forced to terminate prematurely due to the arrival of a signal whose <code>SV_INTERRUPT</code> bit in <code>sv_flags</code> is set (see <code>sigvec(3UCB)</code>). The <code>signal(3C)</code> function sets this bit for any signal it catches.</td></tr> <tr> <td>EINVAL</td><td>Auditing is disabled. See <code>auditon(2)</code>.</td></tr> </table>	EAGAIN	The descriptor referred to a <i>stream</i> , was marked for System V-style non-blocking I/O, and no data could be written immediately.	EBADF	The <i>fd</i> argument is not a valid descriptor open for writing.	EBUSY	A second process attempted to perform this call.	EFBIG	An attempt was made to write a file that exceeds the process's file size limit or the maximum file size.	EINTR	The call is forced to terminate prematurely due to the arrival of a signal whose <code>SV_INTERRUPT</code> bit in <code>sv_flags</code> is set (see <code>sigvec(3UCB)</code>). The <code>signal(3C)</code> function sets this bit for any signal it catches.	EINVAL	Auditing is disabled. See <code>auditon(2)</code> .
EAGAIN	The descriptor referred to a <i>stream</i> , was marked for System V-style non-blocking I/O, and no data could be written immediately.												
EBADF	The <i>fd</i> argument is not a valid descriptor open for writing.												
EBUSY	A second process attempted to perform this call.												
EFBIG	An attempt was made to write a file that exceeds the process's file size limit or the maximum file size.												
EINTR	The call is forced to terminate prematurely due to the arrival of a signal whose <code>SV_INTERRUPT</code> bit in <code>sv_flags</code> is set (see <code>sigvec(3UCB)</code>). The <code>signal(3C)</code> function sets this bit for any signal it catches.												
EINVAL	Auditing is disabled. See <code>auditon(2)</code> .												

		auditsvc(2)
		<i>fd</i> does not refer to a file of an appropriate type. Regular files are always appropriate.
	EIO	An I/O error occurred while reading from or writing to the file system.
	ENOSPC	The user's quota of disk blocks on the file system containing the file has been exhausted; audit filesystem space is below the specified limit; or there is no free space remaining on the file system containing the file.
	ENXIO	A hangup occurred on the <i>stream</i> being written to.
	EPERM	The process did not have the proper privilege in its effective set.
	EWOLDBLOCK	The file was marked for 4.2 BSD-style non-blocking I/O, and no data could be written immediately.
SUMMARY OF TRUSTED SOLARIS CHANGES	This functionality is active only if auditing is enabled. By default, auditing is enabled in the Trusted Solaris environment. See <i>Trusted Solaris Audit Administration Manual</i> for more information.	
	A process must have PRIV_SYS_AUDIT in its set of effective privileges in order to execute this call successfully.	
Trusted Solaris 8 4/01 Reference Manual	auditd(1M), audit(2), auditon(2), audit.log(4)	
Trusted Solaris 8 4/01 Reference Manual	sigvec(3UCB)	

chdir(2)

NAME	chdir, fchdir – change working directory																		
SYNOPSIS	<pre>#include <unistd.h> int chdir(const char *path) ; int fchdir(int fildes) ;</pre>																		
DESCRIPTION	<p>The <code>chdir()</code> and <code>fchdir()</code> functions cause a directory pointed to by <i>path</i> or <i>fildes</i> to become the current working directory. The starting point for path searches for path names not beginning with / (slash). The <i>path</i> argument points to the path name of a directory. The <i>fildes</i> argument is an open file descriptor of a directory.</p> <p>For a directory to become the current directory, a process must have execute (search) access to the directory.</p>																		
RETURN VALUES	<p><code>chdir()</code> returns:</p> <p>0 On success.</p> <p>-1 On failure, and sets <code>errno</code> to indicate the error.</p>																		
ERRORS	<p>The <code>chdir()</code> function will fail if:</p> <table> <tr> <td>EACCES</td><td>Search permission is denied for some component of <i>path</i>. To override this restriction, the calling process may assert one or both of these privileges: <code>PRIV_FILE_DAC_SEARCH</code> and <code>PRIV_FILE_MAC_SEARCH</code>.</td></tr> <tr> <td>EFAULT</td><td>The <i>path</i> argument points to an illegal address.</td></tr> <tr> <td>EINTR</td><td>A signal was caught during the execution of the <code>chdir()</code> function.</td></tr> <tr> <td>EIO</td><td>An I/O error occurred while reading from or writing to the file system.</td></tr> <tr> <td>ELOOP</td><td>Too many symbolic links were encountered in translating <i>path</i>.</td></tr> <tr> <td>ENAMETOOLONG</td><td>The length of the <i>path</i> argument exceeds <code>PATH_MAX</code>, or the length of a <i>path</i> component exceeds <code>NAME_MAX</code> while <code>_POSIX_NO_TRUNC</code> is in effect.</td></tr> <tr> <td>ENOENT</td><td>Either a component of the path prefix or the directory named by <i>path</i> does not exist or is a null pathname.</td></tr> <tr> <td>ENOLINK</td><td>The <i>path</i> argument points to a remote machine and the link to that machine is no longer active.</td></tr> <tr> <td>ENOTDIR</td><td>A component of the path name is not a directory.</td></tr> </table> <p>The <code>fchdir()</code> function will fail if:</p>	EACCES	Search permission is denied for some component of <i>path</i> . To override this restriction, the calling process may assert one or both of these privileges: <code>PRIV_FILE_DAC_SEARCH</code> and <code>PRIV_FILE_MAC_SEARCH</code> .	EFAULT	The <i>path</i> argument points to an illegal address.	EINTR	A signal was caught during the execution of the <code>chdir()</code> function.	EIO	An I/O error occurred while reading from or writing to the file system.	ELOOP	Too many symbolic links were encountered in translating <i>path</i> .	ENAMETOOLONG	The length of the <i>path</i> argument exceeds <code>PATH_MAX</code> , or the length of a <i>path</i> component exceeds <code>NAME_MAX</code> while <code>_POSIX_NO_TRUNC</code> is in effect.	ENOENT	Either a component of the path prefix or the directory named by <i>path</i> does not exist or is a null pathname.	ENOLINK	The <i>path</i> argument points to a remote machine and the link to that machine is no longer active.	ENOTDIR	A component of the path name is not a directory.
EACCES	Search permission is denied for some component of <i>path</i> . To override this restriction, the calling process may assert one or both of these privileges: <code>PRIV_FILE_DAC_SEARCH</code> and <code>PRIV_FILE_MAC_SEARCH</code> .																		
EFAULT	The <i>path</i> argument points to an illegal address.																		
EINTR	A signal was caught during the execution of the <code>chdir()</code> function.																		
EIO	An I/O error occurred while reading from or writing to the file system.																		
ELOOP	Too many symbolic links were encountered in translating <i>path</i> .																		
ENAMETOOLONG	The length of the <i>path</i> argument exceeds <code>PATH_MAX</code> , or the length of a <i>path</i> component exceeds <code>NAME_MAX</code> while <code>_POSIX_NO_TRUNC</code> is in effect.																		
ENOENT	Either a component of the path prefix or the directory named by <i>path</i> does not exist or is a null pathname.																		
ENOLINK	The <i>path</i> argument points to a remote machine and the link to that machine is no longer active.																		
ENOTDIR	A component of the path name is not a directory.																		

chdir(2)

EACCES	Search permission is denied for <i>fil</i> des. To override this restriction, the calling process may assert one or both of these privileges: PRIV_FILE_DAC_SEARCH and PRIV_FILE_MAC_SEARCH.
EBADF	The <i>fil</i> des argument is not an open file descriptor.
EINTR	A signal was caught during the execution of the <code>fchdir()</code> function.
EIO	An I/O error occurred while reading from or writing to the file system.
ENOLINK	The <i>fil</i> des argument points to a remote machine and the link to that machine is no longer active.
ENOTDIR	The open file descriptor <i>fil</i> des does not refer to a directory.

ATTRIBUTES See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	chdir() is Async-Signal-Safe

Appropriate privilege is required to override access checks.

chroot(2)

attributes(5)

chmod(2)

NAME	chmod, fchmod – change access permission mode of file																																													
SYNOPSIS	<pre>#include <sys/types.h> #include <sys/stat.h> int chmod(const char *path, mode_t mode); int fchmod(int fildes, mode_t mode);</pre>																																													
DESCRIPTION	<p>The chmod() and fchmod() functions set the access permission portion of the mode of the file whose name is given by path or referenced by the open file descriptor fildes to the bit pattern contained in mode. Access permission bits are interpreted as follows:</p> <table><tr><td>S_ISUID</td><td>04000</td><td>Set user ID on execution.</td></tr><tr><td>S_ISGID</td><td>020#0</td><td>Set group ID on execution if # is 7, 5, 3, or 1. Enable mandatory file/record locking if # is 6, 4, 2, or 0.</td></tr><tr><td>S_ISVTX</td><td>01000</td><td>Save text image after execution.</td></tr><tr><td>S_IRWXU</td><td>00700</td><td>Read, write, execute by owner.</td></tr><tr><td>S_IRUSR</td><td>00400</td><td>Read by owner.</td></tr><tr><td>S_IWUSR</td><td>00200</td><td>Write by owner.</td></tr><tr><td>S_IXUSR</td><td>00100</td><td>Execute (search if a directory) by owner.</td></tr><tr><td>S_IRWXG</td><td>00070</td><td>Read, write, execute by group.</td></tr><tr><td>S_IRGRP</td><td>00040</td><td>Read by group.</td></tr><tr><td>S_IWGRP</td><td>00020</td><td>Write by group.</td></tr><tr><td>S_IXGRP</td><td>00010</td><td>Execute by group.</td></tr><tr><td>S_IRWXO</td><td>00007</td><td>Read, write, execute (search) by others.</td></tr><tr><td>S_IROTH</td><td>00004</td><td>Read by others.</td></tr><tr><td>S_IWOTH</td><td>00002</td><td>Write by others.</td></tr><tr><td>S_IXOTH</td><td>00001</td><td>Execute by others.</td></tr></table> <p>Modes are constructed by the bitwise OR operation of the access permission bits.</p> <p>The effective user ID of the process must match the owner of the file or the process must have the PRIV_FILE_SETDAC privilege to change the mode of a file.</p> <p>If the process is not a privileged process and the file is not a directory, mode bit 01000 (save text image on execution) is cleared. The calling process may assert the PRIV_SYS_CONFIG privilege to override this restriction.</p>	S_ISUID	04000	Set user ID on execution.	S_ISGID	020#0	Set group ID on execution if # is 7, 5, 3, or 1. Enable mandatory file/record locking if # is 6, 4, 2, or 0.	S_ISVTX	01000	Save text image after execution.	S_IRWXU	00700	Read, write, execute by owner.	S_IRUSR	00400	Read by owner.	S_IWUSR	00200	Write by owner.	S_IXUSR	00100	Execute (search if a directory) by owner.	S_IRWXG	00070	Read, write, execute by group.	S_IRGRP	00040	Read by group.	S_IWGRP	00020	Write by group.	S_IXGRP	00010	Execute by group.	S_IRWXO	00007	Read, write, execute (search) by others.	S_IROTH	00004	Read by others.	S_IWOTH	00002	Write by others.	S_IXOTH	00001	Execute by others.
S_ISUID	04000	Set user ID on execution.																																												
S_ISGID	020#0	Set group ID on execution if # is 7, 5, 3, or 1. Enable mandatory file/record locking if # is 6, 4, 2, or 0.																																												
S_ISVTX	01000	Save text image after execution.																																												
S_IRWXU	00700	Read, write, execute by owner.																																												
S_IRUSR	00400	Read by owner.																																												
S_IWUSR	00200	Write by owner.																																												
S_IXUSR	00100	Execute (search if a directory) by owner.																																												
S_IRWXG	00070	Read, write, execute by group.																																												
S_IRGRP	00040	Read by group.																																												
S_IWGRP	00020	Write by group.																																												
S_IXGRP	00010	Execute by group.																																												
S_IRWXO	00007	Read, write, execute (search) by others.																																												
S_IROTH	00004	Read by others.																																												
S_IWOTH	00002	Write by others.																																												
S_IXOTH	00001	Execute by others.																																												

chmod(2)

If neither the process is privileged, nor the file's group is a member of the process's supplementary group list, and the effective group ID of the process does not match the group ID of the file, mode bit 02000 (set group ID on execution) is cleared.

If a directory is writable and has `S_ISVTX` (the sticky bit) set, files within that directory can be removed or renamed only if one or more of the following is true (see `unlink(2)` and `rename(2)`):

- the user owns the file
- the user owns the directory
- the file is writable by the user
- the user is a privileged user

If a directory has the set group ID bit set, a given file created within that directory will have the same group ID as the directory, if that group ID is part of the group ID set of the process that created the file. Otherwise, the newly created file's group ID will be set to the effective group ID of the creating process.

If the mode bit 02000 (set group ID on execution) is set and the mode bit 00010 (execute or search by group) is not set, mandatory file/record locking will exist on a regular file. This may affect future calls to `open(2)`, `creat(2)`, `read(2)`, and `write(2)` on this file.

Upon successful completion, `chmod()` and `fchmod()` mark for update the `st_ctime` field of the file.

RETURN VALUES

`chmod()` returns:

- 0 On success.
- 1 On failure, and sets `errno` to indicate the error.

ERRORS

The `chmod()` function will fail if:

- | | |
|---------------------|--|
| <code>EACCES</code> | <p>Search permission is denied on a component of the path prefix of <i>path</i>. To override this restriction, the calling process may assert the <code>PRIV_FILE_DAC_SEARCH</code> privilege and/or the <code>PRIV_FILE_MAC_SEARCH</code> privilege.</p> <p>The calling process does not own the final object specified in <i>path</i> or does not own <i>files</i>. To override this restriction, the calling process may assert the <code>PRIV_FILE_SETDAC</code> privilege.</p> <p>Write permission is denied on <i>path</i> or <i>files</i>. To override this restriction, the calling process may assert the <code>PRIV_FILE_DAC_WRITE</code> and/or the <code>PRIV_FILE_MAC_WRITE</code> privilege.</p> |
| <code>EFAULT</code> | <p>The <i>path</i> argument points to an illegal address.</p> |

chmod(2)

EINTR	A signal was caught during execution of the function.
EIO	An I/O error occurred while reading from or writing to the file system.
ELOOP	Too many symbolic links were encountered in translating <i>path</i> .
ENAMETOOLONG	The length of the <i>path</i> argument exceeds <code>PATH_MAX</code> , or the length of a <i>path</i> component exceeds <code>NAME_MAX</code> while <code>_POSIX_NO_TRUNC</code> is in effect.
ENOENT	Either a component of the path prefix or the file referred to by <i>path</i> does not exist or is a null pathname.
ENOLINK	The <i>fildev</i> argument points to a remote machine and the link to that machine is no longer active.
ENOTDIR	A component of the prefix of <i>path</i> is not a directory.
EPERM	The effective user ID does not match the owner of the file. To override this restriction, the calling process may assert the <code>PRIV_FILE_SETDAC</code> privilege.
EROFS	The file referred to by <i>path</i> resides on a read-only file system.

The `fchmod()` function will fail if:

EBADF	The <i>fildev</i> argument is not an open file descriptor
EIO	An I/O error occurred while reading from or writing to the file system.
EINTR	A signal was caught during execution of the <code>fchmod()</code> function.
ENOLINK	The <i>path</i> argument points to a remote machine and the link to that machine is no longer active.
EROFS	The file referred to by <i>fildev</i> resides on a read-only file system.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	<code>chmod()</code> is Async-Signal-Safe

SUMMARY OF TRUSTED SOLARIS CHANGES

Appropriate privilege is required to override access checks.

- To override a search permission error, the calling process requires the `PRIV_FILE_MAC_SEARCH` privilege.

- To override a write permission error, the calling process requires the `PRIV_FILE_DAC_WRITE` and/or the `PRIV_FILE_MAC_WRITE` privilege.
- If the calling process does not own the object, the calling process requires the `PRIV_FILE_SETDAC` privilege.

To set the sticky bit on a file, the calling process may assert the `PRIV_SYS_CONFIG` privilege.

To set the set-user-ID on a user who is not the effective user ID of the calling process, the calling process may assert the `PRIV_FILE_SETID` privilege.

To set the set-group-ID bit on a group not in effective or supplementary group IDs of the calling process, the calling process may assert the `PRIV_FILE_SETID` privilege.

**Trusted Solaris 8
4/01 Reference
Manual**
**SunOS 5.8
Reference Manual**

`chmod(1)`, `chown(2)`, `creat(2)`, `fcntl(2)`, `mknod(2)`, `open(2)`, `read(2)`, `rename(2)`,
`stat(2)`, `write(2)`

`mkfifo(3C)`, `stat(3HEAD)`, `attributes(5)`

System Interface Guide

NOTES

If you use `chmod()` to change the file group owner permissions on a file with ACL entries, both the file group owner permissions and the ACL mask are changed to the new permissions. Be aware that the new ACL mask permissions may change the effective permissions for additional users and groups who have ACL entries on the file.

chown(2)

NAME	chown, lchown, fchown – change owner and group of a file
SYNOPSIS	<pre>#include <unistd.h> #include <sys/types.h> int chown(const char *path, uid_t owner, gid_t group); int lchown(const char *path, uid_t owner, gid_t group); int fchown(int fildes, uid_t owner, gid_t group);</pre>
DESCRIPTION	<p>The <code>chown()</code> function sets the owner ID and group ID of the file specified by <i>path</i> or referenced by the open file descriptor <i>fildes</i> to <i>owner</i> and <i>group</i> respectively. If <i>owner</i> or <i>group</i> is specified as <code>-1</code>, <code>chown()</code> does not change the corresponding ID of the file.</p> <p>The <code>lchown()</code> function sets the owner ID and group ID of the named file in the same manner as <code>chown()</code>, unless the named file is a symbolic link. In this case, <code>lchown()</code> changes the ownership of the symbolic link file itself, while <code>chown()</code> changes the ownership of the file or directory to which the symbolic link refers.</p> <p>If <code>chown()</code>, <code>lchown()</code>, or <code>fchown()</code> is invoked, the set-user-ID and set-group-ID bits of the file mode, <code>chmod(2)</code>, respectively, are cleared. See <code>chmod(2)</code>. To bypass this restriction, the process may assert the <code>PRIV_FILE_SETID</code> privilege.</p> <p>The operating system has a configuration option, <code>_POSIX_CHOWN_RESTRICTED</code>, to restrict ownership changes for the <code>chown()</code>, <code>lchown()</code>, and <code>fchown()</code> functions. When <code>_POSIX_CHOWN_RESTRICTED</code> is not in effect, the effective user ID of the process must match the owner of the file. To override this restriction, the calling process must assert the <code>PRIV_FILE_CHOWN</code> privilege. When <code>_POSIX_CHOWN_RESTRICTED</code> is not in effect, the effective user ID of the process must match the owner of the file or the process must be the super-user to change the ownership of a file. When <code>_POSIX_CHOWN_RESTRICTED</code> is in effect, the <code>chown()</code>, <code>lchown()</code>, and <code>fchown()</code> functions require that the calling process assert the <code>PRIV_FILE_CHOWN</code> privilege to change the user ID of a file. To change the group ID of a file, the process must be the owner of the file and the new group ID must be the group of the process ID or must be in the supplementary group list of the process. To override this restriction, the calling process may assert the <code>PRIV_FILE_CHOWN</code> privilege.</p> <p>set rstchown = 1</p> <p>To disable this option, include the following line in <code>/etc/system</code>:</p> <p>set rstchown = 0</p> <p>See <code>system(4)</code> and <code>fpathconf(2)</code>.</p> <p>Upon successful completion, <code>chown()</code>, <code>fchown()</code> and <code>lchown()</code> mark for update the <code>st_ctime</code> field of the file.</p>

RETURN VALUES	chown() returns:	
	0	On success.
	-1	On failure, and sets errno to indicate the error.
ERRORS	The chown() and lchown() functions will fail if:	
	EACCES	Search permission is denied on a component of the path prefix of <i>path</i> . To override this restriction, the calling process may assert one or both of these privileges: PRIV_FILE_DAC_SEARCH and PRIV_FILE_MAC_SEARCH.
		Write permission is denied on <i>path</i> or <i>filides</i> . To override this restriction, the calling process may assert the PRIV_FILE_MAC_WRITE privilege.
	EFAULT	The <i>path</i> argument points to an illegal address.
	EINTR	A signal was caught during the execution of the chown() or lchown() function.
	EINVAL	The <i>group</i> or <i>owner</i> argument is out of range.
	EIO	An I/O error occurred while reading from or writing to the file system.
	ELOOP	Too many symbolic links were encountered in translating <i>path</i> .
	ENAMETOOLONG	The length of the <i>path</i> argument exceeds PATH_MAX, or the length of a <i>path</i> component exceeds NAME_MAX while _POSIX_NO_TRUNC is in effect.
	ENOLINK	The <i>path</i> argument points to a remote machine and the link to that machine is no longer active.
	ENOENT	Either a component of the path prefix or the file referred to by <i>path</i> does not exist or is a null pathname.
	ENOTDIR	A component of the path prefix of <i>path</i> is not a directory.
	EPERM	The effective user ID does not match the owner of the file. If _POSIX_CHOWN_RESTRICTED is set, the calling process must assert the PRIV_FILE_CHOWN privilege. If _POSIX_CHOWN_RESTRICTED is not set, the calling process may assert the PRIV_FILE_CHOWN privilege.
	EROFS	The named file resides on a read-only file system.
	The fchown() function will fail if:	
	EBADF	The <i>filides</i> argument is not an open file descriptor.

chown(2)

EIO	An I/O error occurred while reading from or writing to the file system.
EINTR	A signal was caught during execution of the function.
ENOLINK	The <i>fildev</i> argument points to a remote machine and the link to that machine is no longer active.
EINVAL	The <i>group</i> or <i>owner</i> argument is out of range.
EPERM	The effective user ID does not match the owner of the file, or the process is not the super-user and <code>_POSIX_CHOWN_RESTRICTED</code> indicates that such privilege is required.
EROFS	The named file referred to by <i>fildev</i> resides on a read-only file system.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	<code>chown()</code> is Async-Signal-Safe

SUMMARY OF TRUSTED SOLARIS CHANGES

Appropriate privilege is required to override access checks.

When the ownership of *path* and *fildev* is changed, the set-user-ID and set-group-ID bits are cleared. The calling process may assert the `PRIV_FILE_SETID` privilege to bypass this restriction.

To change the user ID of the file when the calling process does not own the file and `_POSIX_CHOWN_RESTRICTED` is not in effect, the calling process may assert the `PRIV_FILE_CHOWN` privilege.

To change the group ID of the file when the calling process does not own the file, and the new group ID is not in the group ID of the process or in the supplementary group list of the process, and `_POSIX_CHOWN_RESTRICTED` is not in effect, the calling process may assert the `PRIV_FILE_CHOWN` privilege.

Trusted Solaris 8
4/01 Reference
Manual
Solaris 8
Reference Manual

`chgrp(1)`, `chown(1)`, `chmod(2)`

`attributes(5)`

NAME	chroot, fchroot – change root directory										
SYNOPSIS	<pre>#include <unistd.h> int chroot(const char *path); int fchroot(int fildes);</pre>										
DESCRIPTION	<p>The <code>chroot()</code> and <code>fchroot()</code> functions cause a directory to become the root directory, the starting point for path searches for path names beginning with / (slash). The user's working directory is unaffected by the <code>chroot()</code> and <code>fchroot()</code> functions.</p> <p>The <i>path</i> argument points to a path name naming a directory. The <i>fildes</i> argument to <code>fchroot()</code> is the open file descriptor of the directory which is to become the root.</p> <p>The calling process must assert the <code>PRIV_PROC_CHROOT</code> privilege to use this system call. While it is always possible to change to the system root using the <code>fchroot()</code> function, it is not guaranteed to succeed in any other case, even should <i>fildes</i> be valid in all respects.</p> <p>The <code>“.”</code> entry in the root directory is interpreted to mean the root directory itself. Therefore, <code>“.”</code> cannot be used to access files outside the subtree rooted at the root directory. Instead, <code>fchroot()</code> can be used to reset the root to a directory that was opened before the root directory was changed.</p>										
RETURN VALUES	<p><code>chroot()</code> returns:</p> <p>0 On success.</p> <p>-1 On failure, and sets <code>errno</code> to indicate the error.</p>										
ERRORS	<p>The <code>chroot()</code> function will fail if:</p> <table> <tr> <td>EACCES</td><td>Search permission is denied for a component of the path prefix of <i>dirname</i>, or search permission is denied for the directory referred to by <i>dirname</i>. To override these restrictions, the calling process may assert one or both of these privileges: <code>PRIV_FILE_DAC_SEARCH</code> and <code>PRIV_FILE_MAC_SEARCH</code>.</td></tr> <tr> <td>EBADF</td><td>The descriptor is not valid.</td></tr> <tr> <td>EFAULT</td><td>The <i>path</i> argument points to an illegal address.</td></tr> <tr> <td>EINVAL</td><td>The <code>fchroot()</code> function attempted to change to a directory that is not the system root and external circumstances do not allow this.</td></tr> <tr> <td>EINTR</td><td>A signal was caught during the execution of the <code>chroot()</code> function.</td></tr> </table>	EACCES	Search permission is denied for a component of the path prefix of <i>dirname</i> , or search permission is denied for the directory referred to by <i>dirname</i> . To override these restrictions, the calling process may assert one or both of these privileges: <code>PRIV_FILE_DAC_SEARCH</code> and <code>PRIV_FILE_MAC_SEARCH</code> .	EBADF	The descriptor is not valid.	EFAULT	The <i>path</i> argument points to an illegal address.	EINVAL	The <code>fchroot()</code> function attempted to change to a directory that is not the system root and external circumstances do not allow this.	EINTR	A signal was caught during the execution of the <code>chroot()</code> function.
EACCES	Search permission is denied for a component of the path prefix of <i>dirname</i> , or search permission is denied for the directory referred to by <i>dirname</i> . To override these restrictions, the calling process may assert one or both of these privileges: <code>PRIV_FILE_DAC_SEARCH</code> and <code>PRIV_FILE_MAC_SEARCH</code> .										
EBADF	The descriptor is not valid.										
EFAULT	The <i>path</i> argument points to an illegal address.										
EINVAL	The <code>fchroot()</code> function attempted to change to a directory that is not the system root and external circumstances do not allow this.										
EINTR	A signal was caught during the execution of the <code>chroot()</code> function.										

chroot(2)

	EIO	An I/O error occurred while reading from or writing to the file system.
	ELOOP	Too many symbolic links were encountered in translating <i>path</i> .
	ENAMETOOLONG	The length of the <i>path</i> argument exceeds <code>PATH_MAX</code> , or the length of a <i>path</i> component exceeds <code>NAME_MAX</code> while <code>_POSIX_NO_TRUNC</code> is in effect.
	ENOENT	The named directory does not exist or is a null pathname.
	ENOLINK	The <i>path</i> argument points to a remote machine and the link to that machine is no longer active.
	ENOTDIR	Any component of the path name is not a directory.
	EPERM	The calling process must assert the <code>PRIV_PROC_CHROOT</code> privilege to change the root directory.
SUMMARY OF TRUSTED SOLARIS CHANGES		Appropriate privilege is required to override access checks.
		The calling process must assert the <code>PRIV_PROC_CHROOT</code> privilege to change the root directory.
Trusted Solaris 8 4/01 Reference WARNINGS	chroot(1M)	
		The only use of <code>fchroot()</code> that is appropriate is to change back to the system root.

NAME	chstate – Change the view of a host state between labeled and unlabeled
SYNOPSIS	<pre>cc [flags...] file... -ltsol #include <tsol/tndb.h> int chstate(tsol_chstateop_t <i>state</i>, struct netbuf *<i>addr</i>);</pre>
DESCRIPTION	<p>A host regards another host as labeled or unlabeled, based on the remote host's database caches that are loaded in the kernel. In some cases (for example, when a diskless client boots), the server host must initially regard the client as an unlabeled host even though the client is a labeled host; at a later time, the server host can regard the client as a labeled host. <code>chstate()</code> allows a process to toggle the view of a host between labeled and unlabeled.</p> <p>The argument <i>state</i> is of the following type:</p> <pre>typedef enum { STATE_UNLABELED = 1, STATE_LABELED = 2 } tsol_chstateop_t;</pre> <p>The argument <i>addr</i> is a pointer to the <code>netbuf</code> structure:</p> <pre>struct netbuf { unsigned int maxlen; unsigned int len; char *buf; };</pre> <p>where <i>buf</i> contains the address of the host whose view is being changed. Currently, only the IP address format is supported; and it should be specified as type <code>sockaddr_in</code>.</p> <p><code>chstate()</code> requires the <code>PRIV_SYS_NET_CONFIG</code> privilege.</p>
RETURN VALUES	<p><code>chstate()</code> returns:</p> <p>0 On success.</p> <p>-1 On failure, and sets <code>errno</code> to indicate the error.</p>
ERRORS	<p><code>chstate()</code> may fail for one of these reasons:</p> <p>EFAULT The <i>addr</i> argument points to a bad address.</p> <p>EINVAL Either the <i>state</i> argument is not one of the listed type constants, or the remote host template for the host specified by <i>addr</i> is not available (after using fallback mechanism).</p> <p>EPERM The calling process does not have the <code>PRIV_SYS_NET_CONFIG</code> privilege.</p>

creat(2)

NAME	creat – create a new file or rewrite an existing one		
SYNOPSIS	<pre>#include <sys/types.h> #include <sys/stat.h> #include <fcntl.h> int creat(const char *path, mode_t mode);</pre>		
DESCRIPTION	<p>The <code>creat()</code> function creates a new ordinary file or prepares to rewrite an existing file named by the path name pointed to by <i>path</i>.</p> <p>If the file exists, the length is truncated to 0 and the mode and owner are unchanged.</p> <p>If the file does not exist the file's owner ID is set to the effective user ID of the process. The group ID of the file is set to the effective group ID of the process, or if the <code>S_ISGID</code> bit is set in the parent directory then the group ID of the file is inherited from the parent directory. The access permission bits of the file mode are set to the value of <i>mode</i> modified as follows:</p> <ul style="list-style-type: none"> ■ If the group ID of the new file does not match the effective group ID or one of the supplementary group IDs, the <code>S_ISGID</code> bit is cleared. The calling process may assert the <code>PRIV_FILE_SETID</code> privilege to override clearing of the <code>S_ISGID</code> bit. ■ All bits set in the process's file mode creation mask (see <code>umask(2)</code>) are correspondingly cleared in the file's permission mask. ■ The "save text image after execution bit" of the mode is cleared. (See <code>chmod(2)</code> for the values of <i>mode</i>.) The calling process may assert the <code>PRIV_SYS_CONFIG</code> privilege to override the clearing of the <code>S_ISVTX</code> bit. <p>If the file exists, its sensitivity label is unchanged. If the file does not exist, it is created with its sensitivity label set to the sensitivity label of the calling process.</p> <p>Upon successful completion, a write-only file descriptor is returned and the file is open for writing, even if the mode does not permit writing. The file pointer is set to the beginning of the file. The file descriptor is set to remain open across <i>exec</i> functions (see <code>fcntl(2)</code>). A new file may be created with a mode that forbids writing.</p> <p>The call <code>creat(path, mode)</code> is equivalent to:</p> <pre>open(path, O_WRONLY O_CREAT O_TRUNC, mode)</pre>		
RETURN VALUES	Upon successful completion, a non-negative integer representing the lowest numbered unused file descriptor is returned. Otherwise, -1 is returned, no files are created or modified, and <code>errno</code> is set to indicate the error.		
ERRORS	<p>The <code>creat()</code> function will fail:</p> <table> <tr> <td>EACCES</td><td>Search permission is denied on a component of the path prefix. To override this restriction, the calling process may assert one or both of these privileges: <code>PRIV_FILE_DAC_SEARCH</code> and <code>PRIV_FILE_MAC_SEARCH</code>.</td></tr> </table>	EACCES	Search permission is denied on a component of the path prefix. To override this restriction, the calling process may assert one or both of these privileges: <code>PRIV_FILE_DAC_SEARCH</code> and <code>PRIV_FILE_MAC_SEARCH</code> .
EACCES	Search permission is denied on a component of the path prefix. To override this restriction, the calling process may assert one or both of these privileges: <code>PRIV_FILE_DAC_SEARCH</code> and <code>PRIV_FILE_MAC_SEARCH</code> .		

creat(2)

The file does not exist and the directory in which the file is to be created does not permit writing. To override this restriction, the calling process may assert one or both of these privileges: `PRIV_FILE_DAC_WRITE` and `PRIV_FILE_MAC_WRITE`.

The file exists and write permission to *path* is denied. To override this restriction, the calling process may assert one or both of these privileges: `PRIV_FILE_MAC_WRITE` and `PRIV_FILE_DAC_WRITE`.

EAGAIN	The file exists, mandatory file/record locking is set, and there are outstanding record locks on the file. [See <code>chmod(2)</code> .]
EDQUOT	The directory where the new file entry is being placed cannot be extended because the user's quota of disk blocks on that file system has been exhausted, or the user's quota of inodes on the file system where the file is being created has been exhausted.
EFAULT	The <i>path</i> argument points to an illegal address.
EINTR	A signal was caught during the execution of the <code>creat()</code> function.
EISDIR	The named file is an existing directory.
ELOOP	Too many symbolic links were encountered in translating <i>path</i> .
EMFILE	The process has too many open files. [See <code>getrlimit(2)</code> .]
ENFILE	The system file table is full.
ENOENT	A component of the path prefix does not exist, or the path name is null.
ENOLINK	The <i>path</i> argument points to a remote machine and the link to that machine is no longer active.
ENOSPC	The file system is out of inodes.
ENOTDIR	A component of the path prefix is not a directory.
EOVERFLOW	The file is a large file at the time of <code>creat()</code> .
EROFS	The named file resides or would reside on a read-only file system.

USAGE The `creat()` function has a transitional interface for 64-bit file offsets. See `1f64(5)`.

ATTRIBUTES See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Async-Signal-Safe

creat(2)

**SUMMARY OF
TRUSTED
SOLARIS
CHANGES**

Appropriate privilege is required to override access checks.

To override clearing of the S_ISVTX bit, the calling process may assert the PRIV_SYS_CONFIG privilege. To override the clearing of the S_ISGID bit, the calling process may assert the PRIV_FILE_SETID privilege.

If *path* exists, its sensitivity label is unchanged. If *path* does not exist, it is created with its sensitivity label set to the sensitivity label of the calling process.

**Trusted Solaris 8
4/01 Reference
Manual**

chmod(2), fcntl(2), getrlimit(2), lseek(2), open(2), read(2), write(2)

close(2), dup(2), umask(2), attributes(5), lf64(5), stat(5)

NAME	devpolicy – Get/set device driver policy table				
SYNOPSIS	<pre>cc [flag...] file #include <sys/tsol/devpolicy.h> int devpolicy(devpolicy_op_t op, devpolicy_t *tbl, int *len);</pre>				
DESCRIPTION	<p>devpolicy() sets and gets the device policy table.</p> <p>Allowed values for <i>op</i> are specified in <sys/tsol/devpolicy.h> and may be one of the following:</p> <table> <tr> <td>TSOL_GET_DEVPOLICY</td><td>Get the device policy table. The <i>tbl</i> argument points to a buffer containing the devpolicy_t array, and <i>len</i> contains the length of the array. devpolicy() returns in <i>len</i> the number of elements that the kernel has filled in the array.</td></tr> <tr> <td>TSOL_SET_DEVPOLICY</td><td>Set the device policy table. The <i>tbl</i> argument points to the devpolicy_t structure to be downloaded to the kernel, and <i>len</i> contains the length of the array. For this call to succeed, the calling process must have PRIV_SYS_DEVICES in its set of effective privileges.</td></tr> </table>	TSOL_GET_DEVPOLICY	Get the device policy table. The <i>tbl</i> argument points to a buffer containing the devpolicy_t array, and <i>len</i> contains the length of the array. devpolicy() returns in <i>len</i> the number of elements that the kernel has filled in the array.	TSOL_SET_DEVPOLICY	Set the device policy table. The <i>tbl</i> argument points to the devpolicy_t structure to be downloaded to the kernel, and <i>len</i> contains the length of the array. For this call to succeed, the calling process must have PRIV_SYS_DEVICES in its set of effective privileges.
TSOL_GET_DEVPOLICY	Get the device policy table. The <i>tbl</i> argument points to a buffer containing the devpolicy_t array, and <i>len</i> contains the length of the array. devpolicy() returns in <i>len</i> the number of elements that the kernel has filled in the array.				
TSOL_SET_DEVPOLICY	Set the device policy table. The <i>tbl</i> argument points to the devpolicy_t structure to be downloaded to the kernel, and <i>len</i> contains the length of the array. For this call to succeed, the calling process must have PRIV_SYS_DEVICES in its set of effective privileges.				
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> <tr> <td>Availability</td><td>SUNWtsu</td></tr> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
Availability	SUNWtsu				
Trusted Solaris 8 4/01 Reference Manual	devpolicy(1M)				
Solaris 9 4/01 Reference Manual	attributes(5)				

exec(2)

NAME	exec, execl, execv, execl, execve, execlp, execvp – execute a file
SYNOPSIS	<pre>#include <unistd.h> int execl(const char *path, const char *arg0, ..., const char *argn, char * /*NULL*/); int execv(const char *path, char *const argv[]); int execl(const char *path, const char *arg0, ..., const char *argn, char * /*NULL*/, char *const envp[]); int execve(const char *path, char *const argv[], char *const envp[]); int execlp(const char *file, const char *arg0, ..., const char *argn, char * /*NULL*/); int execvp(const char *file, char *const argv[]);</pre>
DESCRIPTION	<p>Each of the functions in the <code>exec</code> family replace the current process image with a new process image. The new image is constructed from a regular, executable file called the <i>new process image file</i>. This file is either an executable object file or a file of data for an interpreter. There is no return from a successful call to one of these functions because the calling process image is overlaid by the new process image.</p> <p>An interpreter file begins with a line of the form</p> <pre>#! pathname [arg]</pre> <p>where <i>pathname</i> is the path of the interpreter, and <i>arg</i> is an optional argument. When an interpreter file is executed, the system invokes the specified interpreter. The <i>pathname</i> specified in the interpreter file is passed as <i>arg0</i> to the interpreter. If <i>arg</i> was specified in the interpreter file, it is passed as <i>arg1</i> to the interpreter. The remaining arguments to the interpreter are <i>arg0</i> through <i>argn</i> of the originally <code>exec'd</code> file. The interpreter named by <i>pathname</i> must not be an interpreter file.</p> <p>When a C-language program is executed as a result of this call, it is entered as a C-language function call as follows:</p> <pre>int main (int argc, char *argv[], char *envp[]);</pre> <p>where <i>argc</i> is the argument count, <i>argv</i> is an array of character pointers to the arguments themselves, and <i>envp</i> is an array of character pointers to the environment strings. The <i>argv</i> and <i>environ</i> arrays are each terminated by a null pointer. The null pointer terminating the <i>argv</i> array is not counted in <i>argc</i>. As indicated, <i>argc</i> is at least one, and the first member of the array points to a string containing the name of the file.</p> <p>The arguments specified by a program with one of the <code>exec</code> functions are passed on to the new process image in the <code>main()</code> arguments.</p>

The *path* argument points to a path name that identifies the new process image file.

The *file* argument is used to construct a pathname that identifies the new process image file. If the *file* argument contains a slash character, it is used as the pathname for this file. Otherwise, the path prefix for this file is obtained by a search of the directories passed in the PATH environment variable (see `environ(5)`). The environment is supplied typically by the shell. If the process image file is not a valid executable object file, `exec1p()` and `execvp()` use the contents of that file as standard input to the shell. In this case, the shell becomes the new process image. In a standard-conforming application (see `standards(5)`), the `exec` family of functions use `/usr/bin/ksh` (see `ksh(1)`); otherwise, they use `/usr/bin/sh` (see `sh(1)`).

The arguments represented by *arg0...* are pointers to null-terminated character strings. These strings constitute the argument list available to the new process image. The list is terminated by a null pointer. The *arg0* argument should point to a filename that is associated with the process being started by one of the `exec` functions.

The *argv* argument is an array of character pointers to null-terminated strings. The last member of this array must be a null pointer. These strings constitute the argument list available to the new process image. The value in *argv*[0] should point to a filename that is associated with the process being started by one of the `exec` functions.

The *envp* argument is an array of character pointers to null-terminated strings. These strings constitute the environment for the new process image. The *envp* array is terminated by a null pointer. For `exec1()`, `execv()`, `execvp()`, and `exec1p()`, the C-language run-time start-off routine places a pointer to the environment of the calling process in the global object `extern char **environ`, and it is used to pass the environment of the calling process to the new process image.

The number of bytes available for the new process's combined argument and environment lists is ARG_MAX. It is implementation-dependent whether null terminators, pointers, and/or any alignment bytes are included in this total.

The calling process must have read and execute access to the new process file or have the following in its set of effective privileges:

```
PRIV_FILE_DAC_SEARCH
PRIV_FILE_DAC_EXECUTE
PRIV_FILE_MAC_SEARCH
PRIV_FILE_MAC_READ
```

File descriptors open in the calling process image remain open in the new process image, except for those whose close-on-exec flag `FD_CLOEXEC` is set; (see `fcntl(2)`). For those file descriptors that remain open, all attributes of the open file description, including file locks, remain unchanged.

Directory streams open in the calling process image are closed in the new process image.

exec(2)

The state of conversion descriptors and message catalogue descriptors in the new process image is undefined. For the new process, the equivalent of:

`setlocale(LC_ALL, "C")` is executed at startup.

Signals set to the default action (`SIG_DFL`) in the calling process image are set to the default action in the new process image (see `signal(3C)`). Signals set to be ignored (`SIG_IGN`) by the calling process image are set to be ignored by the new process image. Signals set to be caught by the calling process image are set to the default action in the new process image (see `signal(3HEAD)`). After a successful call to any of the `exec` functions, alternate signal stacks are not preserved and the `SA_ONSTACK` flag is cleared for all signals.

After a successful call to any of the `exec` functions, any functions previously registered by `atexit(3C)` are no longer registered.

The saved resource limits in the new process image are set to be a copy of the process's corresponding hard and soft resource limits.

If the `ST_NOSUID` bit is set for the file system containing the new process image file, then the effective user ID, effective group ID, saved set-user-ID, and saved set-group-ID are unchanged in the new process image. If the set-user-ID mode bit of the new process image file is set (see `chmod(2)`), the effective user ID of the new process image is set to the owner ID of the new process image file. Similarly, if the set-group-ID mode bit of the new process image file is set, the effective group ID of the new process image is set to the group ID of the new process image file. The real user ID and real group ID of the new process image remain the same as those of the calling process image. The effective user ID and effective group ID of the new process image are saved (as the saved set-user-ID and the saved set-group-ID for use by `setuid(2)`).

If the process has the `PRIV_PROC_OWNER` privilege, the set-user-ID and set-group-ID bits will be honored when the process is being controlled by `ptrace()`.

Any shared memory segments attached to the calling process image will not be attached to the new process image (see `shmop(2)`). Any mappings established through `mmap()` are not preserved across an `exec`. Memory mappings created in the process are unmapped before the address space is rebuilt for the new process image. (see `mmap(2)`).

Memory locks established by the calling process via calls to `mlockall(3C)` or `mlock(3C)` are removed. If locked pages in the address space of the calling process are also mapped into the address spaces the locks established by the other processes will be unaffected by the call by this process to the `exec` function. If the `exec` function fails, the effect on memory locks is unspecified.

If `_XOPEN_REALTIME` is defined and has a value other than `-1`, any named semaphores open in the calling process are closed as if by appropriate calls to `sem_close(3RT)`

Profiling is disabled for the new process; see `profil(2)`.

Timers created by the calling process with `timer_create(3RT)` are deleted before replacing the current process image with the new process image.

For the `SCHED_FIFO` and `SCHED_RR` scheduling policies, the policy and priority settings are not changed by a call to an `exec` function.

All open message queue descriptors in the calling process are closed, as described in `mq_close(3RT)`.

Any outstanding asynchronous I/O operations may be cancelled. Those asynchronous I/O operations that are not canceled will complete as if the `exec()` function had not yet occurred, but any associated signal notifications are suppressed. It is unspecified whether the `exec()` function itself blocks awaiting such I/O completion. In no event, however, will the new process image created by the `exec()` function be affected by the presence of outstanding asynchronous I/O operations at the time the `exec()` function is called.

The new process also inherits the following attributes from the calling process:

- nice value (see `nice(2)`)
- scheduler class and priority (see `priocntl(2)`)
- process ID
- parent process ID
- process group ID
- supplementary group IDs
- `semadj` values (see `semop(2)`)
- session membership (see `exit(2)` and `signal(3C)`)
- real user ID
- real group ID
- trace flag (see `ptrace(2)` request 0)
- time left until an alarm clock signal (see `alarm(2)`)
- current working directory
- root directory
- file mode creation mask (see `umask(2)`)
- file size limit (see `ulimit(2)`)
- resource limits (see `getrlimit(2)`)
- `tms_utime`, `tms_stime`, `tms_cutime`, and `tms_cstime` (see `times(2)`)
- file-locks (see `fcntl(2)` and `lockf(3C)`)
- controlling terminal
- process signal mask (see `sigprocmask(2)`)
- pending signals (see `sigpending(2)`)
- clearance (see `getclearance(2)`)
- sensitivity label (see `getcmwlabel(2)`)
- inheritable privilege set (see `getppriv(2)`)
- process attribute flags (see `getpattr(2)`)

exec(2)

A call to any `exec` function from a process with more than one thread results in all threads being terminated and the new executable image being loaded and executed. No destructor functions will be called.

The four privilege sets of the new process are updated as described in the following equations where $E1$, $P1$, $S1$, $I1$ are the four privilege sets of the calling process; $E2$, $P2$, $S2$, $I2$ are the four privilege sets of the new process; and F and A are the forced set and the allowed set of the program file:

$$\begin{aligned} E2 &= P2 = (I1 \text{ union } F) \text{ intersect } A_i \\ S2 &= I1 \text{ intersect } A \\ I2 &= I1 \end{aligned}$$

When a script file is run, the resulting forced privileges are the combination of the forced privileges of the script and the forced privileges of the interpreter program; and the resulting allowed privileges are the allowed privileges of the interpreter program. The privilege update equations for a script executable could be expressed like this:

$$\begin{aligned} E2 &= P2 = (I1 \text{ union } F_s \text{ union } F_i) \text{ intersect } A_i \\ S2 &= I1 \text{ intersect } A_i \\ I2 &= I1 \text{ where} \end{aligned}$$

F_s is the forced privilege set of the script, F_i is the forced privilege set of the interpreter program, and A_i is the allowed privilege set of the interpreter program.

Upon successful completion, each of the functions in the `exec` family marks for update the `st_atime` field of the file. If an `exec` function failed but was able to locate the *process image file*, whether the `st_atime` field is marked for update is unspecified. Should the function succeed, the process image file is considered to have been opened with `open(2)`. The corresponding `close(2)` is considered to occur at a time after this open, but before process termination or successful completion of a subsequent call to one of the `exec` functions. The `argv[]` and `envp[]` arrays of pointers and the strings to which those arrays point will not be modified by a call to one of the `exec` functions, except as a consequence of replacing the process image. The `argv[]` and `envp[]` arrays of pointers and the strings to which those arrays point will not be modified by a call to one of the `exec` functions, except as a consequence of replacing the process image.

The saved resource limits in the new process image are set to be a copy of the process's corresponding hard and soft limits.

RETURN VALUES

If a function in the `exec` family returns to the calling process image, an error has occurred; the return value is `-1` and `errno` is set to indicate the error.

ERRORS

The `exec` functions will fail if:

<code>E2BIG</code>	The number of bytes in the new process's argument list is greater than the system-imposed limit of <code>ARG_MAX</code> bytes. The argument
--------------------	---

	list limit is sum of the size of the argument list plus the size of the environment's exported shell variables.
EACCES	Search permission is denied for a directory listed in the new process file's path prefix; the new process file is not an ordinary file; or the new process file mode denies execute permission. Moreover, the calling process does not have <code>PRIV_FILE_DAC_SEARCH</code> and/or <code>PRIV_FILE_MAC_SEARCH</code> to override the restriction.
EAGAIN	Total amount of system memory available when reading using raw I/O is temporarily insufficient.
EFAULT	An argument points to an illegal address.
EINTR	A signal was caught during the execution of one of the functions in the <code>exec</code> family.
ELOOP	Too many symbolic links were encountered in translating <i>path</i> or <i>file</i> .
ENAMETOOLONG	The length of the <i>file</i> or <i>path</i> argument exceeds <code>PATH_MAX</code> , or the length of a <i>file</i> or <i>path</i> component exceeds <code>{NAME_MAX}</code> while <code>{_POSIX_NO_TRUNC}</code> is in effect.
ENOENT	One or more components of the new process path name of the file do not exist or is a null pathname.
ENOLINK	The <i>path</i> argument points to a remote machine and the link to that machine is no longer active.
ENOTDIR	A component of the new process path of the file prefix is not a directory.
The <code>exec</code> functions, except for <code>exec1p()</code> and <code>execvp()</code> , will fail if:	
ENOEXEC	The new process image file has the appropriate access permission but is not in the proper format.
The <code>exec</code> functions may fail if:	
ENAMETOOLONG	Pathname resolution of a symbolic link produced an intermediate result whose length exceeds <code>PATH_MAX</code> .
ENOMEM	The new process image requires more memory than <code>RLIMIT_VMEM</code> , the limit imposed by <code>setrlimit()</code> (see <code>brk(2)</code>).
ETXTBSY	The new process image file is a pure procedure (shared text) file that is currently open for writing by some process.
USAGE	As the state of conversion descriptors and message catalogue descriptors in the new process image is undefined, portable applications should not rely on their use and should close them prior to calling one of the <code>exec</code> functions.

exec(2)

Applications that require other than the default POSIX locale should call `setlocale(3C)` with the appropriate parameters to establish the locale of the new process.

The *environ* array should not be accessed directly by the application.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	<code>execle()</code> and <code>execve()</code> are Async-Signal-Safe

SUMMARY OF TRUSTED SOLARIS CHANGES

MAC search and execute permissions on the executable object are required. Process privilege sets are updated upon execution of the program. Other Trusted Solaris process attributes, such as clearance, sensitivity label, and process attribute flags, are unchanged.

Trusted Solaris 8 4/01 Reference Manual

`chmod(2)`, `fcntl(2)`, `fork(2)`, `getrlimit(2)`, `nice(2)`, `priocntl(2)`, `semop(2)`, `shmop(2)`, `setlocale(3C)`,

SunOS 5.8 Reference Manual

`ksh(1)`, `ps(1)`, `sh(1)`, `alarm(2)`, `brk(2)`, `exit(2)`, `mmap(2)`, `profil(2)`, `ptrace(2)`, `sigpending(2)`, `sigprocmask(2)`, `times(2)`, `umask(2)`, `lockf(3C)`, `signal(3C)`, `system(3C)`, `timer_create(3RT)`, `a.out(4)`, `attributes(5)`, `environ(5)`, `standards(5)`

NAME	exec, execl, execv, execl, execve, execlp, execvp – execute a file
SYNOPSIS	<pre>#include <unistd.h> int execl(const char *path, const char *arg0, ..., const char *argn, char * /*NULL*/); int execv(const char *path, char *const argv[]); int execl(const char *path, const char *arg0, ..., const char *argn, char * /*NULL*/, char *const envp[]); int execve(const char *path, char *const argv[], char *const envp[]); int execlp(const char *file, const char *arg0, ..., const char *argn, char * /*NULL*/); int execvp(const char *file, char *const argv[]);</pre>
DESCRIPTION	<p>Each of the functions in the <code>exec</code> family replace the current process image with a new process image. The new image is constructed from a regular, executable file called the <i>new process image file</i>. This file is either an executable object file or a file of data for an interpreter. There is no return from a successful call to one of these functions because the calling process image is overlaid by the new process image.</p> <p>An interpreter file begins with a line of the form</p> <pre>#! pathname [arg]</pre> <p>where <i>pathname</i> is the path of the interpreter, and <i>arg</i> is an optional argument. When an interpreter file is executed, the system invokes the specified interpreter. The <i>pathname</i> specified in the interpreter file is passed as <i>arg0</i> to the interpreter. If <i>arg</i> was specified in the interpreter file, it is passed as <i>arg1</i> to the interpreter. The remaining arguments to the interpreter are <i>arg0</i> through <i>argn</i> of the originally <code>exec'd</code> file. The interpreter named by <i>pathname</i> must not be an interpreter file.</p> <p>When a C-language program is executed as a result of this call, it is entered as a C-language function call as follows:</p> <pre>int main (int argc, char *argv[], char *envp[]);</pre> <p>where <i>argc</i> is the argument count, <i>argv</i> is an array of character pointers to the arguments themselves, and <i>envp</i> is an array of character pointers to the environment strings. The <i>argv</i> and <i>environ</i> arrays are each terminated by a null pointer. The null pointer terminating the <i>argv</i> array is not counted in <i>argc</i>. As indicated, <i>argc</i> is at least one, and the first member of the array points to a string containing the name of the file.</p> <p>The arguments specified by a program with one of the <code>exec</code> functions are passed on to the new process image in the <code>main()</code> arguments.</p>

execl(2)

The *path* argument points to a path name that identifies the new process image file.

The *file* argument is used to construct a pathname that identifies the new process image file. If the *file* argument contains a slash character, it is used as the pathname for this file. Otherwise, the path prefix for this file is obtained by a search of the directories passed in the PATH environment variable (see `environ(5)`). The environment is supplied typically by the shell. If the process image file is not a valid executable object file, `execlp()` and `execvp()` use the contents of that file as standard input to the shell. In this case, the shell becomes the new process image. In a standard-conforming application (see `standards(5)`), the `exec` family of functions use `/usr/bin/ksh` (see `ksh(1)`); otherwise, they use `/usr/bin/sh` (see `sh(1)`).

The arguments represented by *arg0*... are pointers to null-terminated character strings. These strings constitute the argument list available to the new process image. The list is terminated by a null pointer. The *arg0* argument should point to a filename that is associated with the process being started by one of the `exec` functions.

The *argv* argument is an array of character pointers to null-terminated strings. The last member of this array must be a null pointer. These strings constitute the argument list available to the new process image. The value in *argv*[0] should point to a filename that is associated with the process being started by one of the `exec` functions.

The *envp* argument is an array of character pointers to null-terminated strings. These strings constitute the environment for the new process image. The *envp* array is terminated by a null pointer. For `execl()`, `execv()`, `execvp()`, and `execlp()`, the C-language run-time start-off routine places a pointer to the environment of the calling process in the global object `extern char **environ`, and it is used to pass the environment of the calling process to the new process image.

The number of bytes available for the new process's combined argument and environment lists is ARG_MAX. It is implementation-dependent whether null terminators, pointers, and/or any alignment bytes are included in this total.

The calling process must have read and execute access to the new process file or have the following in its set of effective privileges:

```
PRIV_FILE_DAC_SEARCH
PRIV_FILE_DAC_EXECUTE
PRIV_FILE_MAC_SEARCH
PRIV_FILE_MAC_READ
```

File descriptors open in the calling process image remain open in the new process image, except for those whose close-on-exec flag `FD_CLOEXEC` is set; (see `fcntl(2)`). For those file descriptors that remain open, all attributes of the open file description, including file locks, remain unchanged.

Directory streams open in the calling process image are closed in the new process image.

The state of conversion descriptors and message catalogue descriptors in the new process image is undefined. For the new process, the equivalent of:

`setlocale(LC_ALL, "C")` is executed at startup.

Signals set to the default action (`SIG_DFL`) in the calling process image are set to the default action in the new process image (see `signal(3C)`). Signals set to be ignored (`SIG_IGN`) by the calling process image are set to be ignored by the new process image. Signals set to be caught by the calling process image are set to the default action in the new process image (see `signal(3HEAD)`). After a successful call to any of the `exec` functions, alternate signal stacks are not preserved and the `SA_ONSTACK` flag is cleared for all signals.

After a successful call to any of the `exec` functions, any functions previously registered by `atexit(3C)` are no longer registered.

The saved resource limits in the new process image are set to be a copy of the process's corresponding hard and soft resource limits.

If the `ST_NOSUID` bit is set for the file system containing the new process image file, then the effective user ID, effective group ID, saved set-user-ID, and saved set-group-ID are unchanged in the new process image. If the set-user-ID mode bit of the new process image file is set (see `chmod(2)`), the effective user ID of the new process image is set to the owner ID of the new process image file. Similarly, if the set-group-ID mode bit of the new process image file is set, the effective group ID of the new process image is set to the group ID of the new process image file. The real user ID and real group ID of the new process image remain the same as those of the calling process image. The effective user ID and effective group ID of the new process image are saved (as the saved set-user-ID and the saved set-group-ID for use by `setuid(2)`).

If the process has the `PRIV_PROC_OWNER` privilege, the set-user-ID and set-group-ID bits will be honored when the process is being controlled by `ptrace()`.

Any shared memory segments attached to the calling process image will not be attached to the new process image (see `shmop(2)`). Any mappings established through `mmap()` are not preserved across an `exec`. Memory mappings created in the process are unmapped before the address space is rebuilt for the new process image. (see `mmap(2)`).

Memory locks established by the calling process via calls to `mlockall(3C)` or `mlock(3C)` are removed. If locked pages in the address space of the calling process are also mapped into the address spaces the locks established by the other processes will be unaffected by the call by this process to the `exec` function. If the `exec` function fails, the effect on memory locks is unspecified.

If `_XOPEN_REALTIME` is defined and has a value other than `-1`, any named semaphores open in the calling process are closed as if by appropriate calls to `sem_close(3RT)`

execl(2)

Profiling is disabled for the new process; see `profil(2)`.

Timers created by the calling process with `timer_create(3RT)` are deleted before replacing the current process image with the new process image.

For the `SCHED_FIFO` and `SCHED_RR` scheduling policies, the policy and priority settings are not changed by a call to an `exec` function.

All open message queue descriptors in the calling process are closed, as described in `mq_close(3RT)`.

Any outstanding asynchronous I/O operations may be cancelled. Those asynchronous I/O operations that are not canceled will complete as if the `exec()` function had not yet occurred, but any associated signal notifications are suppressed. It is unspecified whether the `exec()` function itself blocks awaiting such I/O completion. In no event, however, will the new process image created by the `exec()` function be affected by the presence of outstanding asynchronous I/O operations at the time the `exec()` function is called.

The new process also inherits the following attributes from the calling process:

- nice value (see `nice(2)`)
- scheduler class and priority (see `priocntl(2)`)
- process ID
- parent process ID
- process group ID
- supplementary group IDs
- `semadj` values (see `semop(2)`)
- session membership (see `exit(2)` and `signal(3C)`)
- real user ID
- real group ID
- trace flag (see `ptrace(2)` request 0)
- time left until an alarm clock signal (see `alarm(2)`)
- current working directory
- root directory
- file mode creation mask (see `umask(2)`)
- file size limit (see `ulimit(2)`)
- resource limits (see `getrlimit(2)`)
- `tms_utime`, `tms_stime`, `tms_cutime`, and `tms_cstime` (see `times(2)`)
- file-locks (see `fcntl(2)` and `lockf(3C)`)
- controlling terminal
- process signal mask (see `sigprocmask(2)`)
- pending signals (see `sigpending(2)`)
- clearance (see `getclearance(2)`)
- sensitivity label (see `getcmwlabel(2)`)
- inheritable privilege set (see `getppriv(2)`)
- process attribute flags (see `getpattr(2)`)

execl(2)

A call to any `exec` function from a process with more than one thread results in all threads being terminated and the new executable image being loaded and executed. No destructor functions will be called.

The four privilege sets of the new process are updated as described in the following equations where $E1, P1, S1, I1$ are the four privilege sets of the calling process; $E2, P2, S2, I2$ are the four privilege sets of the new process; and F and A are the forced set and the allowed set of the program file:

$$\begin{aligned} E2 &= P2 = (I1 \text{ union } F) \text{ intersect } A_i \\ S2 &= I1 \text{ intersect } A \\ I2 &= I1 \end{aligned}$$

When a script file is run, the resulting forced privileges are the combination of the forced privileges of the script and the forced privileges of the interpreter program; and the resulting allowed privileges are the allowed privileges of the interpreter program. The privilege update equations for a script executable could be expressed like this:

$$\begin{aligned} E2 &= P2 = (I1 \text{ union } F_s \text{ union } F_i) \text{ intersect } A_i \\ S2 &= I1 \text{ intersect } A_i \\ I2 &= I1 \text{ where} \end{aligned}$$

F_s is the forced privilege set of the script, F_i is the forced privilege set of the interpreter program, and A_i is the allowed privilege set of the interpreter program.

Upon successful completion, each of the functions in the `exec` family marks for update the `st_atime` field of the file. If an `exec` function failed but was able to locate the *process image file*, whether the `st_atime` field is marked for update is unspecified. Should the function succeed, the process image file is considered to have been opened with `open(2)`. The corresponding `close(2)` is considered to occur at a time after this open, but before process termination or successful completion of a subsequent call to one of the `exec` functions. The `argv[]` and `envp[]` arrays of pointers and the strings to which those arrays point will not be modified by a call to one of the `exec` functions, except as a consequence of replacing the process image. The `argv[]` and `envp[]` arrays of pointers and the strings to which those arrays point will not be modified by a call to one of the `exec` functions, except as a consequence of replacing the process image.

The saved resource limits in the new process image are set to be a copy of the process's corresponding hard and soft limits.

RETURN VALUES

If a function in the `exec` family returns to the calling process image, an error has occurred; the return value is `-1` and `errno` is set to indicate the error.

ERRORS

The `exec` functions will fail if:

E2BIG	The number of bytes in the new process's argument list is greater than the system-imposed limit of <code>ARG_MAX</code> bytes. The argument
-------	---

execl(2)

	list limit is sum of the size of the argument list plus the size of the environment's exported shell variables.
EACCES	Search permission is denied for a directory listed in the new process file's path prefix; the new process file is not an ordinary file; or the new process file mode denies execute permission. Moreover, the calling process does not have <code>PRIV_FILE_DAC_SEARCH</code> and/or <code>PRIV_FILE_MAC_SEARCH</code> to override the restriction.
EAGAIN	Total amount of system memory available when reading using raw I/O is temporarily insufficient.
EFAULT	An argument points to an illegal address.
EINTR	A signal was caught during the execution of one of the functions in the <code>exec</code> family.
ELOOP	Too many symbolic links were encountered in translating <i>path</i> or <i>file</i> .
ENAMETOOLONG	The length of the <i>file</i> or <i>path</i> argument exceeds <code>PATH_MAX</code> , or the length of a <i>file</i> or <i>path</i> component exceeds <code>{NAME_MAX}</code> while <code>{_POSIX_NO_TRUNC}</code> is in effect.
ENOENT	One or more components of the new process path name of the file do not exist or is a null pathname.
ENOLINK	The <i>path</i> argument points to a remote machine and the link to that machine is no longer active.
ENOTDIR	A component of the new process path of the file prefix is not a directory.
The <code>exec</code> functions, except for <code>execlp()</code> and <code>execvp()</code> , will fail if:	
ENOEXEC	The new process image file has the appropriate access permission but is not in the proper format.
The <code>exec</code> functions may fail if:	
ENAMETOOLONG	Pathname resolution of a symbolic link produced an intermediate result whose length exceeds <code>PATH_MAX</code> .
ENOMEM	The new process image requires more memory than <code>RLIMIT_VMEM</code> , the limit imposed by <code>setrlimit()</code> (see <code>brk(2)</code>).
ETXTBSY	The new process image file is a pure procedure (shared text) file that is currently open for writing by some process.
USAGE	As the state of conversion descriptors and message catalogue descriptors in the new process image is undefined, portable applications should not rely on their use and should close them prior to calling one of the <code>exec</code> functions.

execl(2)

Applications that require other than the default POSIX locale should call `setlocale(3C)` with the appropriate parameters to establish the locale of the new process.

The *environ* array should not be accessed directly by the application.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	<code>execle()</code> and <code>execve()</code> are Async-Signal-Safe

SUMMARY OF TRUSTED SOLARIS CHANGES

MAC search and execute permissions on the executable object are required. Process privilege sets are updated upon execution of the program. Other Trusted Solaris process attributes, such as clearance, sensitivity label, and process attribute flags, are unchanged.

Trusted Solaris 8 4/01 Reference Manual

`chmod(2)`, `fcntl(2)`, `fork(2)`, `getrlimit(2)`, `nice(2)`, `prctl(2)`, `semop(2)`, `shmop(2)`, `setlocale(3C)`,

SunOS 5.8 Reference Manual

`ksh(1)`, `ps(1)`, `sh(1)`, `alarm(2)`, `brk(2)`, `exit(2)`, `mmap(2)`, `profil(2)`, `ptrace(2)`, `sigpending(2)`, `sigprocmask(2)`, `times(2)`, `umask(2)`, `lockf(3C)`, `signal(3C)`, `system(3C)`, `timer_create(3RT)`, `a.out(4)`, `attributes(5)`, `environ(5)`, `standards(5)`

execle(2)

NAME	exec, execl, execv, execl, execve, execlp, execvp – execute a file
SYNOPSIS	<pre>#include <unistd.h> int execl(const char *path, const char *arg0, ..., const char *argn, char * /*NULL*/); int execv(const char *path, char *const argv[]); int execl(const char *path, const char *arg0, ..., const char *argn, char * /*NULL*/, char *const envp[]); int execve(const char *path, char *const argv[], char *const envp[]); int execlp(const char *file, const char *arg0, ..., const char *argn, char * /*NULL*/); int execvp(const char *file, char *const argv[]);</pre>
DESCRIPTION	<p>Each of the functions in the <code>exec</code> family replace the current process image with a new process image. The new image is constructed from a regular, executable file called the <i>new process image file</i>. This file is either an executable object file or a file of data for an interpreter. There is no return from a successful call to one of these functions because the calling process image is overlaid by the new process image.</p> <p>An interpreter file begins with a line of the form</p> <pre>#! pathname [arg]</pre> <p>where <i>pathname</i> is the path of the interpreter, and <i>arg</i> is an optional argument. When an interpreter file is executed, the system invokes the specified interpreter. The <i>pathname</i> specified in the interpreter file is passed as <i>arg0</i> to the interpreter. If <i>arg</i> was specified in the interpreter file, it is passed as <i>arg1</i> to the interpreter. The remaining arguments to the interpreter are <i>arg0</i> through <i>argn</i> of the originally <code>exec'd</code> file. The interpreter named by <i>pathname</i> must not be an interpreter file.</p> <p>When a C-language program is executed as a result of this call, it is entered as a C-language function call as follows:</p> <pre>int main (int argc, char *argv[], char *envp[]);</pre> <p>where <i>argc</i> is the argument count, <i>argv</i> is an array of character pointers to the arguments themselves, and <i>envp</i> is an array of character pointers to the environment strings. The <i>argv</i> and <i>environ</i> arrays are each terminated by a null pointer. The null pointer terminating the <i>argv</i> array is not counted in <i>argc</i>. As indicated, <i>argc</i> is at least one, and the first member of the array points to a string containing the name of the file.</p> <p>The arguments specified by a program with one of the <code>exec</code> functions are passed on to the new process image in the <code>main()</code> arguments.</p>

The *path* argument points to a path name that identifies the new process image file.

The *file* argument is used to construct a pathname that identifies the new process image file. If the *file* argument contains a slash character, it is used as the pathname for this file. Otherwise, the path prefix for this file is obtained by a search of the directories passed in the PATH environment variable (see *environ(5)*). The environment is supplied typically by the shell. If the process image file is not a valid executable object file, *exec1p()* and *execvp()* use the contents of that file as standard input to the shell. In this case, the shell becomes the new process image. In a standard-conforming application (see *standards(5)*), the *exec* family of functions use */usr/bin/ksh* (see *ksh(1)*); otherwise, they use */usr/bin/sh* (see *sh(1)*).

The arguments represented by *arg0...* are pointers to null-terminated character strings. These strings constitute the argument list available to the new process image. The list is terminated by a null pointer. The *arg0* argument should point to a filename that is associated with the process being started by one of the *exec* functions.

The *argv* argument is an array of character pointers to null-terminated strings. The last member of this array must be a null pointer. These strings constitute the argument list available to the new process image. The value in *argv[0]* should point to a filename that is associated with the process being started by one of the *exec* functions.

The *envp* argument is an array of character pointers to null-terminated strings. These strings constitute the environment for the new process image. The *envp* array is terminated by a null pointer. For *exec1()*, *execv()*, *execvp()*, and *exec1p()*, the C-language run-time start-off routine places a pointer to the environment of the calling process in the global object `extern char **environ`, and it is used to pass the environment of the calling process to the new process image.

The number of bytes available for the new process's combined argument and environment lists is ARG_MAX. It is implementation-dependent whether null terminators, pointers, and/or any alignment bytes are included in this total.

The calling process must have read and execute access to the new process file or have the following in its set of effective privileges:

```
PRIV_FILE_DAC_SEARCH
PRIV_FILE_DAC_EXECUTE
PRIV_FILE_MAC_SEARCH
PRIV_FILE_MAC_READ
```

File descriptors open in the calling process image remain open in the new process image, except for those whose close-on-exec flag *FD_CLOEXEC* is set; (see *fcntl(2)*). For those file descriptors that remain open, all attributes of the open file description, including file locks, remain unchanged.

Directory streams open in the calling process image are closed in the new process image.

execle(2)

The state of conversion descriptors and message catalogue descriptors in the new process image is undefined. For the new process, the equivalent of:

`setlocale(LC_ALL, "C")` is executed at startup.

Signals set to the default action (`SIG_DFL`) in the calling process image are set to the default action in the new process image (see `signal(3C)`). Signals set to be ignored (`SIG_IGN`) by the calling process image are set to be ignored by the new process image. Signals set to be caught by the calling process image are set to the default action in the new process image (see `signal(3HEAD)`). After a successful call to any of the `exec` functions, alternate signal stacks are not preserved and the `SA_ONSTACK` flag is cleared for all signals.

After a successful call to any of the `exec` functions, any functions previously registered by `atexit(3C)` are no longer registered.

The saved resource limits in the new process image are set to be a copy of the process's corresponding hard and soft resource limits.

If the `ST_NOSUID` bit is set for the file system containing the new process image file, then the effective user ID, effective group ID, saved set-user-ID, and saved set-group-ID are unchanged in the new process image. If the set-user-ID mode bit of the new process image file is set (see `chmod(2)`), the effective user ID of the new process image is set to the owner ID of the new process image file. Similarly, if the set-group-ID mode bit of the new process image file is set, the effective group ID of the new process image is set to the group ID of the new process image file. The real user ID and real group ID of the new process image remain the same as those of the calling process image. The effective user ID and effective group ID of the new process image are saved (as the saved set-user-ID and the saved set-group-ID for use by `setuid(2)`).

If the process has the `PRIV_PROC_OWNER` privilege, the set-user-ID and set-group-ID bits will be honored when the process is being controlled by `ptrace()`.

Any shared memory segments attached to the calling process image will not be attached to the new process image (see `shmop(2)`). Any mappings established through `mmap()` are not preserved across an `exec`. Memory mappings created in the process are unmapped before the address space is rebuilt for the new process image. (see `mmap(2)`).

Memory locks established by the calling process via calls to `mlockall(3C)` or `mlock(3C)` are removed. If locked pages in the address space of the calling process are also mapped into the address spaces the locks established by the other processes will be unaffected by the call by this process to the `exec` function. If the `exec` function fails, the effect on memory locks is unspecified.

If `_XOPEN_REALTIME` is defined and has a value other than `-1`, any named semaphores open in the calling process are closed as if by appropriate calls to `sem_close(3RT)`

Profiling is disabled for the new process; see `profil(2)`.

Timers created by the calling process with `timer_create(3RT)` are deleted before replacing the current process image with the new process image.

For the `SCHED_FIFO` and `SCHED_RR` scheduling policies, the policy and priority settings are not changed by a call to an `exec` function.

All open message queue descriptors in the calling process are closed, as described in `mq_close(3RT)`.

Any outstanding asynchronous I/O operations may be cancelled. Those asynchronous I/O operations that are not canceled will complete as if the `exec()` function had not yet occurred, but any associated signal notifications are suppressed. It is unspecified whether the `exec()` function itself blocks awaiting such I/O completion. In no event, however, will the new process image created by the `exec()` function be affected by the presence of outstanding asynchronous I/O operations at the time the `exec()` function is called.

The new process also inherits the following attributes from the calling process:

- nice value (see `nice(2)`)
- scheduler class and priority (see `prctl(2)`)
- process ID
- parent process ID
- process group ID
- supplementary group IDs
- `semadj` values (see `semop(2)`)
- session membership (see `exit(2)` and `signal(3C)`)
- real user ID
- real group ID
- trace flag (see `ptrace(2)` request 0)
- time left until an alarm clock signal (see `alarm(2)`)
- current working directory
- root directory
- file mode creation mask (see `umask(2)`)
- file size limit (see `ulimit(2)`)
- resource limits (see `getrlimit(2)`)
- `tms_utime`, `tms_stime`, `tms_cutime`, and `tms_cstime` (see `times(2)`)
- file-locks (see `fcntl(2)` and `lockf(3C)`)
- controlling terminal
- process signal mask (see `sigprocmask(2)`)
- pending signals (see `sigpending(2)`)
- clearance (see `getclearance(2)`)
- sensitivity label (see `getcmwlabel(2)`)
- inheritable privilege set (see `getppriv(2)`)
- process attribute flags (see `getpattr(2)`)

execle(2)

A call to any `exec` function from a process with more than one thread results in all threads being terminated and the new executable image being loaded and executed. No destructor functions will be called.

The four privilege sets of the new process are updated as described in the following equations where $E1$, $P1$, $S1$, $I1$ are the four privilege sets of the calling process; $E2$, $P2$, $S2$, $I2$ are the four privilege sets of the new process; and F and A are the forced set and the allowed set of the program file:

$$\begin{aligned} E2 &= P2 = (I1 \text{ union } F) \text{ intersect } A_i \\ S2 &= I1 \text{ intersect } A \\ I2 &= I1 \end{aligned}$$

When a script file is run, the resulting forced privileges are the combination of the forced privileges of the script and the forced privileges of the interpreter program; and the resulting allowed privileges are the allowed privileges of the interpreter program. The privilege update equations for a script executable could be expressed like this:

$$\begin{aligned} E2 &= P2 = (I1 \text{ union } F_s \text{ union } F_i) \text{ intersect } A_i \\ S2 &= I1 \text{ intersect } A_i \\ I2 &= I1 \text{ where} \end{aligned}$$

F_s is the forced privilege set of the script, F_i is the forced privilege set of the interpreter program, and A_i is the allowed privilege set of the interpreter program.

Upon successful completion, each of the functions in the `exec` family marks for update the `st_atime` field of the file. If an `exec` function failed but was able to locate the *process image file*, whether the `st_atime` field is marked for update is unspecified. Should the function succeed, the process image file is considered to have been opened with `open(2)`. The corresponding `close(2)` is considered to occur at a time after this open, but before process termination or successful completion of a subsequent call to one of the `exec` functions. The `argv[]` and `envp[]` arrays of pointers and the strings to which those arrays point will not be modified by a call to one of the `exec` functions, except as a consequence of replacing the process image. The `argv[]` and `envp[]` arrays of pointers and the strings to which those arrays point will not be modified by a call to one of the `exec` functions, except as a consequence of replacing the process image.

The saved resource limits in the new process image are set to be a copy of the process's corresponding hard and soft limits.

RETURN VALUES

If a function in the `exec` family returns to the calling process image, an error has occurred; the return value is `-1` and `errno` is set to indicate the error.

ERRORS

The `exec` functions will fail if:

<code>E2BIG</code>	The number of bytes in the new process's argument list is greater than the system-imposed limit of <code>ARG_MAX</code> bytes. The argument
--------------------	---

	list limit is sum of the size of the argument list plus the size of the environment's exported shell variables.
EACCES	Search permission is denied for a directory listed in the new process file's path prefix; the new process file is not an ordinary file; or the new process file mode denies execute permission. Moreover, the calling process does not have <code>PRIV_FILE_DAC_SEARCH</code> and/or <code>PRIV_FILE_MAC_SEARCH</code> to override the restriction.
EAGAIN	Total amount of system memory available when reading using raw I/O is temporarily insufficient.
EFAULT	An argument points to an illegal address.
EINTR	A signal was caught during the execution of one of the functions in the <code>exec</code> family.
ELOOP	Too many symbolic links were encountered in translating <i>path</i> or <i>file</i> .
ENAMETOOLONG	The length of the <i>file</i> or <i>path</i> argument exceeds <code>PATH_MAX</code> , or the length of a <i>file</i> or <i>path</i> component exceeds <code>{NAME_MAX}</code> while <code>{_POSIX_NO_TRUNC}</code> is in effect.
ENOENT	One or more components of the new process path name of the file do not exist or is a null pathname.
ENOLINK	The <i>path</i> argument points to a remote machine and the link to that machine is no longer active.
ENOTDIR	A component of the new process path of the file prefix is not a directory.
The <code>exec</code> functions, except for <code>exec1p()</code> and <code>execvp()</code> , will fail if:	
ENOEXEC	The new process image file has the appropriate access permission but is not in the proper format.
The <code>exec</code> functions may fail if:	
ENAMETOOLONG	Pathname resolution of a symbolic link produced an intermediate result whose length exceeds <code>PATH_MAX</code> .
ENOMEM	The new process image requires more memory than <code>RLIMIT_VMEM</code> , the limit imposed by <code>setrlimit()</code> (see <code>brk(2)</code>).
ETXTBSY	The new process image file is a pure procedure (shared text) file that is currently open for writing by some process.
USAGE	As the state of conversion descriptors and message catalogue descriptors in the new process image is undefined, portable applications should not rely on their use and should close them prior to calling one of the <code>exec</code> functions.

execle(2)

Applications that require other than the default POSIX locale should call `setlocale(3C)` with the appropriate parameters to establish the locale of the new process.

The *environ* array should not be accessed directly by the application.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	<code>execle()</code> and <code>execve()</code> are Async-Signal-Safe

SUMMARY OF TRUSTED SOLARIS CHANGES

MAC search and execute permissions on the executable object are required. Process privilege sets are updated upon execution of the program. Other Trusted Solaris process attributes, such as clearance, sensitivity label, and process attribute flags, are unchanged.

Trusted Solaris 8 4/01 Reference Manual

`chmod(2)`, `fcntl(2)`, `fork(2)`, `getrlimit(2)`, `nice(2)`, `priocntl(2)`, `semop(2)`, `shmop(2)`, `setlocale(3C)`,

SunOS 5.8 Reference Manual

`ksh(1)`, `ps(1)`, `sh(1)`, `alarm(2)`, `brk(2)`, `exit(2)`, `mmap(2)`, `profil(2)`, `ptrace(2)`, `sigpending(2)`, `sigprocmask(2)`, `times(2)`, `umask(2)`, `lockf(3C)`, `signal(3C)`, `system(3C)`, `timer_create(3RT)`, `a.out(4)`, `attributes(5)`, `environ(5)`, `standards(5)`

NAME	exec, execl, execv, execl, execve, execlp, execvp – execute a file
SYNOPSIS	<pre>#include <unistd.h> int execl(const char *path, const char *arg0, ..., const char *argn, char * /*NULL*/); int execv(const char *path, char *const argv[]); int execl(const char *path, const char *arg0, ..., const char *argn, char * /*NULL*/, char *const envp[]); int execve(const char *path, char *const argv[], char *const envp[]); int execlp(const char *file, const char *arg0, ..., const char *argn, char * /*NULL*/); int execvp(const char *file, char *const argv[]);</pre>
DESCRIPTION	<p>Each of the functions in the <code>exec</code> family replace the current process image with a new process image. The new image is constructed from a regular, executable file called the <i>new process image file</i>. This file is either an executable object file or a file of data for an interpreter. There is no return from a successful call to one of these functions because the calling process image is overlaid by the new process image.</p> <p>An interpreter file begins with a line of the form</p> <pre>#! pathname [arg]</pre> <p>where <i>pathname</i> is the path of the interpreter, and <i>arg</i> is an optional argument. When an interpreter file is executed, the system invokes the specified interpreter. The <i>pathname</i> specified in the interpreter file is passed as <i>arg0</i> to the interpreter. If <i>arg</i> was specified in the interpreter file, it is passed as <i>arg1</i> to the interpreter. The remaining arguments to the interpreter are <i>arg0</i> through <i>argn</i> of the originally <code>exec'd</code> file. The interpreter named by <i>pathname</i> must not be an interpreter file.</p> <p>When a C-language program is executed as a result of this call, it is entered as a C-language function call as follows:</p> <pre>int main (int argc, char *argv[], char *envp[]);</pre> <p>where <i>argc</i> is the argument count, <i>argv</i> is an array of character pointers to the arguments themselves, and <i>envp</i> is an array of character pointers to the environment strings. The <i>argv</i> and <i>environ</i> arrays are each terminated by a null pointer. The null pointer terminating the <i>argv</i> array is not counted in <i>argc</i>. As indicated, <i>argc</i> is at least one, and the first member of the array points to a string containing the name of the file.</p> <p>The arguments specified by a program with one of the <code>exec</code> functions are passed on to the new process image in the <code>main()</code> arguments.</p>

execlp(2)

The *path* argument points to a path name that identifies the new process image file.

The *file* argument is used to construct a pathname that identifies the new process image file. If the *file* argument contains a slash character, it is used as the pathname for this file. Otherwise, the path prefix for this file is obtained by a search of the directories passed in the `PATH` environment variable (see `environ(5)`). The environment is supplied typically by the shell. If the process image file is not a valid executable object file, `execlp()` and `execvp()` use the contents of that file as standard input to the shell. In this case, the shell becomes the new process image. In a standard-conforming application (see `standards(5)`), the `exec` family of functions use `/usr/bin/ksh` (see `ksh(1)`); otherwise, they use `/usr/bin/sh` (see `sh(1)`).

The arguments represented by *arg0*... are pointers to null-terminated character strings. These strings constitute the argument list available to the new process image. The list is terminated by a null pointer. The *arg0* argument should point to a filename that is associated with the process being started by one of the `exec` functions.

The *argv* argument is an array of character pointers to null-terminated strings. The last member of this array must be a null pointer. These strings constitute the argument list available to the new process image. The value in *argv*[0] should point to a filename that is associated with the process being started by one of the `exec` functions.

The *envp* argument is an array of character pointers to null-terminated strings. These strings constitute the environment for the new process image. The *envp* array is terminated by a null pointer. For `exec1()`, `execv()`, `execvp()`, and `execlp()`, the C-language run-time start-off routine places a pointer to the environment of the calling process in the global object `extern char **environ`, and it is used to pass the environment of the calling process to the new process image.

The number of bytes available for the new process's combined argument and environment lists is `ARG_MAX`. It is implementation-dependent whether null terminators, pointers, and/or any alignment bytes are included in this total.

The calling process must have read and execute access to the new process file or have the following in its set of effective privileges:

```
PRIV_FILE_DAC_SEARCH
PRIV_FILE_DAC_EXECUTE
PRIV_FILE_MAC_SEARCH
PRIV_FILE_MAC_READ
```

File descriptors open in the calling process image remain open in the new process image, except for those whose close-on-exec flag `FD_CLOEXEC` is set; (see `fcntl(2)`). For those file descriptors that remain open, all attributes of the open file description, including file locks, remain unchanged.

Directory streams open in the calling process image are closed in the new process image.

The state of conversion descriptors and message catalogue descriptors in the new process image is undefined. For the new process, the equivalent of:

`setlocale(LC_ALL, "C")` is executed at startup.

Signals set to the default action (`SIG_DFL`) in the calling process image are set to the default action in the new process image (see `signal(3C)`). Signals set to be ignored (`SIG_IGN`) by the calling process image are set to be ignored by the new process image. Signals set to be caught by the calling process image are set to the default action in the new process image (see `signal(3HEAD)`). After a successful call to any of the `exec` functions, alternate signal stacks are not preserved and the `SA_ONSTACK` flag is cleared for all signals.

After a successful call to any of the `exec` functions, any functions previously registered by `atexit(3C)` are no longer registered.

The saved resource limits in the new process image are set to be a copy of the process's corresponding hard and soft resource limits.

If the `ST_NOSUID` bit is set for the file system containing the new process image file, then the effective user ID, effective group ID, saved set-user-ID, and saved set-group-ID are unchanged in the new process image. If the set-user-ID mode bit of the new process image file is set (see `chmod(2)`), the effective user ID of the new process image is set to the owner ID of the new process image file. Similarly, if the set-group-ID mode bit of the new process image file is set, the effective group ID of the new process image is set to the group ID of the new process image file. The real user ID and real group ID of the new process image remain the same as those of the calling process image. The effective user ID and effective group ID of the new process image are saved (as the saved set-user-ID and the saved set-group-ID for use by `setuid(2)`).

If the process has the `PRIV_PROC_OWNER` privilege, the set-user-ID and set-group-ID bits will be honored when the process is being controlled by `ptrace()`.

Any shared memory segments attached to the calling process image will not be attached to the new process image (see `shmop(2)`). Any mappings established through `mmap()` are not preserved across an `exec`. Memory mappings created in the process are unmapped before the address space is rebuilt for the new process image. (see `mmap(2)`).

Memory locks established by the calling process via calls to `mlockall(3C)` or `mlock(3C)` are removed. If locked pages in the address space of the calling process are also mapped into the address spaces the locks established by the other processes will be unaffected by the call by this process to the `exec` function. If the `exec` function fails, the effect on memory locks is unspecified.

If `_XOPEN_REALTIME` is defined and has a value other than `-1`, any named semaphores open in the calling process are closed as if by appropriate calls to `sem_close(3RT)`

execvp(2)

Profiling is disabled for the new process; see `profil(2)`.

Timers created by the calling process with `timer_create(3RT)` are deleted before replacing the current process image with the new process image.

For the `SCHED_FIFO` and `SCHED_RR` scheduling policies, the policy and priority settings are not changed by a call to an `exec` function.

All open message queue descriptors in the calling process are closed, as described in `mq_close(3RT)`.

Any outstanding asynchronous I/O operations may be cancelled. Those asynchronous I/O operations that are not canceled will complete as if the `exec()` function had not yet occurred, but any associated signal notifications are suppressed. It is unspecified whether the `exec()` function itself blocks awaiting such I/O completion. In no event, however, will the new process image created by the `exec()` function be affected by the presence of outstanding asynchronous I/O operations at the time the `exec()` function is called.

The new process also inherits the following attributes from the calling process:

- nice value (see `nice(2)`)
- scheduler class and priority (see `priocntl(2)`)
- process ID
- parent process ID
- process group ID
- supplementary group IDs
- `semadj` values (see `semop(2)`)
- session membership (see `exit(2)` and `signal(3C)`)
- real user ID
- real group ID
- trace flag (see `ptrace(2)` request 0)
- time left until an alarm clock signal (see `alarm(2)`)
- current working directory
- root directory
- file mode creation mask (see `umask(2)`)
- file size limit (see `ulimit(2)`)
- resource limits (see `getrlimit(2)`)
- `tms_utime`, `tms_stime`, `tms_cutime`, and `tms_cstime` (see `times(2)`)
- file-locks (see `fcntl(2)` and `lockf(3C)`)
- controlling terminal
- process signal mask (see `sigprocmask(2)`)
- pending signals (see `sigpending(2)`)
- clearance (see `getclearance(2)`)
- sensitivity label (see `getcmwlabel(2)`)
- inheritable privilege set (see `getppriv(2)`)
- process attribute flags (see `getpattr(2)`)

execlp(2)

A call to any `exec` function from a process with more than one thread results in all threads being terminated and the new executable image being loaded and executed. No destructor functions will be called.

The four privilege sets of the new process are updated as described in the following equations where $E1, P1, S1, I1$ are the four privilege sets of the calling process; $E2, P2, S2, I2$ are the four privilege sets of the new process; and F and A are the forced set and the allowed set of the program file:

$$\begin{aligned} E2 &= P2 = (I1 \text{ union } F) \text{ intersect } A_i \\ S2 &= I1 \text{ intersect } A \\ I2 &= I1 \end{aligned}$$

When a script file is run, the resulting forced privileges are the combination of the forced privileges of the script and the forced privileges of the interpreter program; and the resulting allowed privileges are the allowed privileges of the interpreter program. The privilege update equations for a script executable could be expressed like this:

$$\begin{aligned} E2 &= P2 = (I1 \text{ union } F_s \text{ union } F_i) \text{ intersect } A_i \\ S2 &= I1 \text{ intersect } A_i \\ I2 &= I1 \text{ where} \end{aligned}$$

F_s is the forced privilege set of the script, F_i is the forced privilege set of the interpreter program, and A_i is the allowed privilege set of the interpreter program.

Upon successful completion, each of the functions in the `exec` family marks for update the `st_atime` field of the file. If an `exec` function failed but was able to locate the *process image file*, whether the `st_atime` field is marked for update is unspecified. Should the function succeed, the process image file is considered to have been opened with `open(2)`. The corresponding `close(2)` is considered to occur at a time after this open, but before process termination or successful completion of a subsequent call to one of the `exec` functions. The `argv[]` and `envp[]` arrays of pointers and the strings to which those arrays point will not be modified by a call to one of the `exec` functions, except as a consequence of replacing the process image. The `argv[]` and `envp[]` arrays of pointers and the strings to which those arrays point will not be modified by a call to one of the `exec` functions, except as a consequence of replacing the process image.

The saved resource limits in the new process image are set to be a copy of the process's corresponding hard and soft limits.

RETURN VALUES

If a function in the `exec` family returns to the calling process image, an error has occurred; the return value is `-1` and `errno` is set to indicate the error.

ERRORS

The `exec` functions will fail if:

<code>E2BIG</code>	The number of bytes in the new process's argument list is greater than the system-imposed limit of <code>ARG_MAX</code> bytes. The argument
--------------------	---

execlp(2)

	list limit is sum of the size of the argument list plus the size of the environment's exported shell variables.
EACCES	Search permission is denied for a directory listed in the new process file's path prefix; the new process file is not an ordinary file; or the new process file mode denies execute permission. Moreover, the calling process does not have <code>PRIV_FILE_DAC_SEARCH</code> and/or <code>PRIV_FILE_MAC_SEARCH</code> to override the restriction.
EAGAIN	Total amount of system memory available when reading using raw I/O is temporarily insufficient.
EFAULT	An argument points to an illegal address.
EINTR	A signal was caught during the execution of one of the functions in the <code>exec</code> family.
ELOOP	Too many symbolic links were encountered in translating <i>path</i> or <i>file</i> .
ENAMETOOLONG	The length of the <i>file</i> or <i>path</i> argument exceeds <code>PATH_MAX</code> , or the length of a <i>file</i> or <i>path</i> component exceeds <code>{NAME_MAX}</code> while <code>{_POSIX_NO_TRUNC}</code> is in effect.
ENOENT	One or more components of the new process path name of the file do not exist or is a null pathname.
ENOLINK	The <i>path</i> argument points to a remote machine and the link to that machine is no longer active.
ENOTDIR	A component of the new process path of the file prefix is not a directory.
The <code>exec</code> functions, except for <code>execlp()</code> and <code>execvp()</code> , will fail if:	
ENOEXEC	The new process image file has the appropriate access permission but is not in the proper format.
The <code>exec</code> functions may fail if:	
ENAMETOOLONG	Pathname resolution of a symbolic link produced an intermediate result whose length exceeds <code>PATH_MAX</code> .
ENOMEM	The new process image requires more memory than <code>RLIMIT_VMEM</code> , the limit imposed by <code>setrlimit()</code> (see <code>brk(2)</code>).
ETXTBSY	The new process image file is a pure procedure (shared text) file that is currently open for writing by some process.
USAGE	As the state of conversion descriptors and message catalogue descriptors in the new process image is undefined, portable applications should not rely on their use and should close them prior to calling one of the <code>exec</code> functions.

execlp(2)

Applications that require other than the default POSIX locale should call `setlocale(3C)` with the appropriate parameters to establish the locale of the new process.

The *environ* array should not be accessed directly by the application.

ATTRIBUTES See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	<code>execle()</code> and <code>execve()</code> are Async-Signal-Safe

SUMMARY OF TRUSTED SOLARIS CHANGES MAC search and execute permissions on the executable object are required. Process privilege sets are updated upon execution of the program. Other Trusted Solaris process attributes, such as clearance, sensitivity label, and process attribute flags, are unchanged.

Trusted Solaris 8 4/01 Reference Manual `chmod(2)`, `fcntl(2)`, `fork(2)`, `getrlimit(2)`, `nice(2)`, `prctl(2)`, `semop(2)`, `shmop(2)`, `setlocale(3C)`,
SunOS 5.8 Reference Manual `ksh(1)`, `ps(1)`, `sh(1)`, `alarm(2)`, `brk(2)`, `exit(2)`, `mmap(2)`, `profil(2)`, `ptrace(2)`, `sigpending(2)`, `sigprocmask(2)`, `times(2)`, `umask(2)`, `lockf(3C)`, `signal(3C)`, `system(3C)`, `timer_create(3RT)`, `a.out(4)`, `attributes(5)`, `environ(5)`, `standards(5)`

execv(2)

NAME	exec, execl, execv, execl, execve, execlp, execvp – execute a file
SYNOPSIS	<pre>#include <unistd.h> int execl(const char *path, const char *arg0, ..., const char *argn, char * /*NULL*/); int execv(const char *path, char *const argv[]); int execl(const char *path, const char *arg0, ..., const char *argn, char * /*NULL*/, char *const envp[]); int execve(const char *path, char *const argv[], char *const envp[]); int execlp(const char *file, const char *arg0, ..., const char *argn, char * /*NULL*/); int execvp(const char *file, char *const argv[]);</pre>
DESCRIPTION	<p>Each of the functions in the <code>exec</code> family replace the current process image with a new process image. The new image is constructed from a regular, executable file called the <i>new process image file</i>. This file is either an executable object file or a file of data for an interpreter. There is no return from a successful call to one of these functions because the calling process image is overlaid by the new process image.</p> <p>An interpreter file begins with a line of the form</p> <pre>#! pathname [arg]</pre> <p>where <i>pathname</i> is the path of the interpreter, and <i>arg</i> is an optional argument. When an interpreter file is executed, the system invokes the specified interpreter. The <i>pathname</i> specified in the interpreter file is passed as <i>arg0</i> to the interpreter. If <i>arg</i> was specified in the interpreter file, it is passed as <i>arg1</i> to the interpreter. The remaining arguments to the interpreter are <i>arg0</i> through <i>argn</i> of the originally <code>exec'd</code> file. The interpreter named by <i>pathname</i> must not be an interpreter file.</p> <p>When a C-language program is executed as a result of this call, it is entered as a C-language function call as follows:</p> <pre>int main (int argc, char *argv[], char *envp[]);</pre> <p>where <i>argc</i> is the argument count, <i>argv</i> is an array of character pointers to the arguments themselves, and <i>envp</i> is an array of character pointers to the environment strings. The <i>argv</i> and <i>environ</i> arrays are each terminated by a null pointer. The null pointer terminating the <i>argv</i> array is not counted in <i>argc</i>. As indicated, <i>argc</i> is at least one, and the first member of the array points to a string containing the name of the file.</p> <p>The arguments specified by a program with one of the <code>exec</code> functions are passed on to the new process image in the <code>main()</code> arguments.</p>

The *path* argument points to a path name that identifies the new process image file.

The *file* argument is used to construct a pathname that identifies the new process image file. If the *file* argument contains a slash character, it is used as the pathname for this file. Otherwise, the path prefix for this file is obtained by a search of the directories passed in the PATH environment variable (see `environ(5)`). The environment is supplied typically by the shell. If the process image file is not a valid executable object file, `exec1p()` and `execvp()` use the contents of that file as standard input to the shell. In this case, the shell becomes the new process image. In a standard-conforming application (see `standards(5)`), the `exec` family of functions use `/usr/bin/ksh` (see `ksh(1)`); otherwise, they use `/usr/bin/sh` (see `sh(1)`).

The arguments represented by *arg0...* are pointers to null-terminated character strings. These strings constitute the argument list available to the new process image. The list is terminated by a null pointer. The *arg0* argument should point to a filename that is associated with the process being started by one of the `exec` functions.

The *argv* argument is an array of character pointers to null-terminated strings. The last member of this array must be a null pointer. These strings constitute the argument list available to the new process image. The value in *argv*[0] should point to a filename that is associated with the process being started by one of the `exec` functions.

The *envp* argument is an array of character pointers to null-terminated strings. These strings constitute the environment for the new process image. The *envp* array is terminated by a null pointer. For `exec1()`, `execv()`, `execvp()`, and `exec1p()`, the C-language run-time start-off routine places a pointer to the environment of the calling process in the global object `extern char **environ`, and it is used to pass the environment of the calling process to the new process image.

The number of bytes available for the new process's combined argument and environment lists is ARG_MAX. It is implementation-dependent whether null terminators, pointers, and/or any alignment bytes are included in this total.

The calling process must have read and execute access to the new process file or have the following in its set of effective privileges:

```
PRIV_FILE_DAC_SEARCH
PRIV_FILE_DAC_EXECUTE
PRIV_FILE_MAC_SEARCH
PRIV_FILE_MAC_READ
```

File descriptors open in the calling process image remain open in the new process image, except for those whose close-on-exec flag `FD_CLOEXEC` is set; (see `fcntl(2)`). For those file descriptors that remain open, all attributes of the open file description, including file locks, remain unchanged.

Directory streams open in the calling process image are closed in the new process image.

execv(2)

The state of conversion descriptors and message catalogue descriptors in the new process image is undefined. For the new process, the equivalent of:

`setlocale(LC_ALL, "C")` is executed at startup.

Signals set to the default action (`SIG_DFL`) in the calling process image are set to the default action in the new process image (see `signal(3C)`). Signals set to be ignored (`SIG_IGN`) by the calling process image are set to be ignored by the new process image. Signals set to be caught by the calling process image are set to the default action in the new process image (see `signal(3HEAD)`). After a successful call to any of the `exec` functions, alternate signal stacks are not preserved and the `SA_ONSTACK` flag is cleared for all signals.

After a successful call to any of the `exec` functions, any functions previously registered by `atexit(3C)` are no longer registered.

The saved resource limits in the new process image are set to be a copy of the process's corresponding hard and soft resource limits.

If the `ST_NOSUID` bit is set for the file system containing the new process image file, then the effective user ID, effective group ID, saved set-user-ID, and saved set-group-ID are unchanged in the new process image. If the set-user-ID mode bit of the new process image file is set (see `chmod(2)`), the effective user ID of the new process image is set to the owner ID of the new process image file. Similarly, if the set-group-ID mode bit of the new process image file is set, the effective group ID of the new process image is set to the group ID of the new process image file. The real user ID and real group ID of the new process image remain the same as those of the calling process image. The effective user ID and effective group ID of the new process image are saved (as the saved set-user-ID and the saved set-group-ID for use by `setuid(2)`).

If the process has the `PRIV_PROC_OWNER` privilege, the set-user-ID and set-group-ID bits will be honored when the process is being controlled by `ptrace()`.

Any shared memory segments attached to the calling process image will not be attached to the new process image (see `shmop(2)`). Any mappings established through `mmap()` are not preserved across an `exec`. Memory mappings created in the process are unmapped before the address space is rebuilt for the new process image. (see `mmap(2)`).

Memory locks established by the calling process via calls to `mlockall(3C)` or `mlock(3C)` are removed. If locked pages in the address space of the calling process are also mapped into the address spaces the locks established by the other processes will be unaffected by the call by this process to the `exec` function. If the `exec` function fails, the effect on memory locks is unspecified.

If `_XOPEN_REALTIME` is defined and has a value other than `-1`, any named semaphores open in the calling process are closed as if by appropriate calls to `sem_close(3RT)`

Profiling is disabled for the new process; see `profil(2)`.

Timers created by the calling process with `timer_create(3RT)` are deleted before replacing the current process image with the new process image.

For the `SCHED_FIFO` and `SCHED_RR` scheduling policies, the policy and priority settings are not changed by a call to an `exec` function.

All open message queue descriptors in the calling process are closed, as described in `mq_close(3RT)`.

Any outstanding asynchronous I/O operations may be cancelled. Those asynchronous I/O operations that are not canceled will complete as if the `exec()` function had not yet occurred, but any associated signal notifications are suppressed. It is unspecified whether the `exec()` function itself blocks awaiting such I/O completion. In no event, however, will the new process image created by the `exec()` function be affected by the presence of outstanding asynchronous I/O operations at the time the `exec()` function is called.

The new process also inherits the following attributes from the calling process:

- nice value (see `nice(2)`)
- scheduler class and priority (see `priocntl(2)`)
- process ID
- parent process ID
- process group ID
- supplementary group IDs
- `semadj` values (see `semop(2)`)
- session membership (see `exit(2)` and `signal(3C)`)
- real user ID
- real group ID
- trace flag (see `ptrace(2)` request 0)
- time left until an alarm clock signal (see `alarm(2)`)
- current working directory
- root directory
- file mode creation mask (see `umask(2)`)
- file size limit (see `ulimit(2)`)
- resource limits (see `getrlimit(2)`)
- `tms_utime`, `tms_stime`, `tms_cutime`, and `tms_cstime` (see `times(2)`)
- file-locks (see `fcntl(2)` and `lockf(3C)`)
- controlling terminal
- process signal mask (see `sigprocmask(2)`)
- pending signals (see `sigpending(2)`)
- clearance (see `getclearance(2)`)
- sensitivity label (see `getcmwlabel(2)`)
- inheritable privilege set (see `getppriv(2)`)
- process attribute flags (see `getpattr(2)`)

execv(2)

A call to any `exec` function from a process with more than one thread results in all threads being terminated and the new executable image being loaded and executed. No destructor functions will be called.

The four privilege sets of the new process are updated as described in the following equations where $E1$, $P1$, $S1$, $I1$ are the four privilege sets of the calling process; $E2$, $P2$, $S2$, $I2$ are the four privilege sets of the new process; and F and A are the forced set and the allowed set of the program file:

$$\begin{aligned} E2 &= P2 = (I1 \text{ union } F) \text{ intersect } A_i \\ S2 &= I1 \text{ intersect } A \\ I2 &= I1 \end{aligned}$$

When a script file is run, the resulting forced privileges are the combination of the forced privileges of the script and the forced privileges of the interpreter program; and the resulting allowed privileges are the allowed privileges of the interpreter program. The privilege update equations for a script executable could be expressed like this:

$$\begin{aligned} E2 &= P2 = (I1 \text{ union } F_s \text{ union } F_i) \text{ intersect } A_i \\ S2 &= I1 \text{ intersect } A_i \\ I2 &= I1 \text{ where} \end{aligned}$$

F_s is the forced privilege set of the script, F_i is the forced privilege set of the interpreter program, and A_i is the allowed privilege set of the interpreter program.

Upon successful completion, each of the functions in the `exec` family marks for update the `st_atime` field of the file. If an `exec` function failed but was able to locate the *process image file*, whether the `st_atime` field is marked for update is unspecified. Should the function succeed, the process image file is considered to have been opened with `open(2)`. The corresponding `close(2)` is considered to occur at a time after this open, but before process termination or successful completion of a subsequent call to one of the `exec` functions. The `argv[]` and `envp[]` arrays of pointers and the strings to which those arrays point will not be modified by a call to one of the `exec` functions, except as a consequence of replacing the process image. The `argv[]` and `envp[]` arrays of pointers and the strings to which those arrays point will not be modified by a call to one of the `exec` functions, except as a consequence of replacing the process image.

The saved resource limits in the new process image are set to be a copy of the process's corresponding hard and soft limits.

RETURN VALUES

If a function in the `exec` family returns to the calling process image, an error has occurred; the return value is `-1` and `errno` is set to indicate the error.

ERRORS

The `exec` functions will fail if:

<code>E2BIG</code>	The number of bytes in the new process's argument list is greater than the system-imposed limit of <code>ARG_MAX</code> bytes. The argument
--------------------	---

execv(2)

	list limit is sum of the size of the argument list plus the size of the environment's exported shell variables.
EACCES	Search permission is denied for a directory listed in the new process file's path prefix; the new process file is not an ordinary file; or the new process file mode denies execute permission. Moreover, the calling process does not have <code>PRIV_FILE_DAC_SEARCH</code> and/or <code>PRIV_FILE_MAC_SEARCH</code> to override the restriction.
EAGAIN	Total amount of system memory available when reading using raw I/O is temporarily insufficient.
EFAULT	An argument points to an illegal address.
EINTR	A signal was caught during the execution of one of the functions in the <code>exec</code> family.
ELOOP	Too many symbolic links were encountered in translating <i>path</i> or <i>file</i> .
ENAMETOOLONG	The length of the <i>file</i> or <i>path</i> argument exceeds <code>PATH_MAX</code> , or the length of a <i>file</i> or <i>path</i> component exceeds <code>{NAME_MAX}</code> while <code>{_POSIX_NO_TRUNC}</code> is in effect.
ENOENT	One or more components of the new process path name of the file do not exist or is a null pathname.
ENOLINK	The <i>path</i> argument points to a remote machine and the link to that machine is no longer active.
ENOTDIR	A component of the new process path of the file prefix is not a directory.
The <code>exec</code> functions, except for <code>execvp()</code> and <code>execvp()</code> , will fail if:	
ENOEXEC	The new process image file has the appropriate access permission but is not in the proper format.
The <code>exec</code> functions may fail if:	
ENAMETOOLONG	Pathname resolution of a symbolic link produced an intermediate result whose length exceeds <code>PATH_MAX</code> .
ENOMEM	The new process image requires more memory than <code>RLIMIT_VMEM</code> , the limit imposed by <code>setrlimit()</code> (see <code>brk(2)</code>).
ETXTBSY	The new process image file is a pure procedure (shared text) file that is currently open for writing by some process.
USAGE	As the state of conversion descriptors and message catalogue descriptors in the new process image is undefined, portable applications should not rely on their use and should close them prior to calling one of the <code>exec</code> functions.

execv(2)

Applications that require other than the default POSIX locale should call `setlocale(3C)` with the appropriate parameters to establish the locale of the new process.

The *environ* array should not be accessed directly by the application.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	<code>execle()</code> and <code>execve()</code> are Async-Signal-Safe

SUMMARY OF TRUSTED SOLARIS CHANGES

MAC search and execute permissions on the executable object are required. Process privilege sets are updated upon execution of the program. Other Trusted Solaris process attributes, such as clearance, sensitivity label, and process attribute flags, are unchanged.

Trusted Solaris 8 4/01 Reference Manual

`chmod(2)`, `fcntl(2)`, `fork(2)`, `getrlimit(2)`, `nice(2)`, `priocntl(2)`, `semop(2)`, `shmop(2)`, `setlocale(3C)`,

SunOS 5.8 Reference Manual

`ksh(1)`, `ps(1)`, `sh(1)`, `alarm(2)`, `brk(2)`, `exit(2)`, `mmap(2)`, `profil(2)`, `ptrace(2)`, `sigpending(2)`, `sigprocmask(2)`, `times(2)`, `umask(2)`, `lockf(3C)`, `signal(3C)`, `system(3C)`, `timer_create(3RT)`, `a.out(4)`, `attributes(5)`, `environ(5)`, `standards(5)`

NAME	exec, execl, execv, execl, execve, execlp, execvp – execute a file
SYNOPSIS	<pre>#include <unistd.h> int execl(const char *path, const char *arg0, ..., const char *argn, char * /*NULL*/); int execv(const char *path, char *const argv[]); int execl(const char *path, const char *arg0, ..., const char *argn, char * /*NULL*/, char *const envp[]); int execve(const char *path, char *const argv[], char *const envp[]); int execlp(const char *file, const char *arg0, ..., const char *argn, char * /*NULL*/); int execvp(const char *file, char *const argv[]);</pre>
DESCRIPTION	<p>Each of the functions in the <code>exec</code> family replace the current process image with a new process image. The new image is constructed from a regular, executable file called the <i>new process image file</i>. This file is either an executable object file or a file of data for an interpreter. There is no return from a successful call to one of these functions because the calling process image is overlaid by the new process image.</p> <p>An interpreter file begins with a line of the form</p> <pre>#! pathname [arg]</pre> <p>where <i>pathname</i> is the path of the interpreter, and <i>arg</i> is an optional argument. When an interpreter file is executed, the system invokes the specified interpreter. The <i>pathname</i> specified in the interpreter file is passed as <i>arg0</i> to the interpreter. If <i>arg</i> was specified in the interpreter file, it is passed as <i>arg1</i> to the interpreter. The remaining arguments to the interpreter are <i>arg0</i> through <i>argn</i> of the originally <code>exec'd</code> file. The interpreter named by <i>pathname</i> must not be an interpreter file.</p> <p>When a C-language program is executed as a result of this call, it is entered as a C-language function call as follows:</p> <pre>int main (int argc, char *argv[], char *envp[]);</pre> <p>where <i>argc</i> is the argument count, <i>argv</i> is an array of character pointers to the arguments themselves, and <i>envp</i> is an array of character pointers to the environment strings. The <i>argv</i> and <i>environ</i> arrays are each terminated by a null pointer. The null pointer terminating the <i>argv</i> array is not counted in <i>argc</i>. As indicated, <i>argc</i> is at least one, and the first member of the array points to a string containing the name of the file.</p> <p>The arguments specified by a program with one of the <code>exec</code> functions are passed on to the new process image in the <code>main()</code> arguments.</p>

execve(2)

The *path* argument points to a path name that identifies the new process image file.

The *file* argument is used to construct a pathname that identifies the new process image file. If the *file* argument contains a slash character, it is used as the pathname for this file. Otherwise, the path prefix for this file is obtained by a search of the directories passed in the PATH environment variable (see *environ(5)*). The environment is supplied typically by the shell. If the process image file is not a valid executable object file, *execlp()* and *execvp()* use the contents of that file as standard input to the shell. In this case, the shell becomes the new process image. In a standard-conforming application (see *standards(5)*), the *exec* family of functions use */usr/bin/ksh* (see *ksh(1)*); otherwise, they use */usr/bin/sh* (see *sh(1)*).

The arguments represented by *arg0...* are pointers to null-terminated character strings. These strings constitute the argument list available to the new process image. The list is terminated by a null pointer. The *arg0* argument should point to a filename that is associated with the process being started by one of the *exec* functions.

The *argv* argument is an array of character pointers to null-terminated strings. The last member of this array must be a null pointer. These strings constitute the argument list available to the new process image. The value in *argv[0]* should point to a filename that is associated with the process being started by one of the *exec* functions.

The *envp* argument is an array of character pointers to null-terminated strings. These strings constitute the environment for the new process image. The *envp* array is terminated by a null pointer. For *execl()*, *execv()*, *execvp()*, and *execlp()*, the C-language run-time start-off routine places a pointer to the environment of the calling process in the global object `extern char **environ`, and it is used to pass the environment of the calling process to the new process image.

The number of bytes available for the new process's combined argument and environment lists is ARG_MAX. It is implementation-dependent whether null terminators, pointers, and/or any alignment bytes are included in this total.

The calling process must have read and execute access to the new process file or have the following in its set of effective privileges:

```
PRIV_FILE_DAC_SEARCH
PRIV_FILE_DAC_EXECUTE
PRIV_FILE_MAC_SEARCH
PRIV_FILE_MAC_READ
```

File descriptors open in the calling process image remain open in the new process image, except for those whose close-on-exec flag *FD_CLOEXEC* is set; (see *fcntl(2)*). For those file descriptors that remain open, all attributes of the open file description, including file locks, remain unchanged.

Directory streams open in the calling process image are closed in the new process image.

The state of conversion descriptors and message catalogue descriptors in the new process image is undefined. For the new process, the equivalent of:

`setlocale(LC_ALL, "C")` is executed at startup.

Signals set to the default action (`SIG_DFL`) in the calling process image are set to the default action in the new process image (see `signal(3C)`). Signals set to be ignored (`SIG_IGN`) by the calling process image are set to be ignored by the new process image. Signals set to be caught by the calling process image are set to the default action in the new process image (see `signal(3HEAD)`). After a successful call to any of the `exec` functions, alternate signal stacks are not preserved and the `SA_ONSTACK` flag is cleared for all signals.

After a successful call to any of the `exec` functions, any functions previously registered by `atexit(3C)` are no longer registered.

The saved resource limits in the new process image are set to be a copy of the process's corresponding hard and soft resource limits.

If the `ST_NOSUID` bit is set for the file system containing the new process image file, then the effective user ID, effective group ID, saved set-user-ID, and saved set-group-ID are unchanged in the new process image. If the set-user-ID mode bit of the new process image file is set (see `chmod(2)`), the effective user ID of the new process image is set to the owner ID of the new process image file. Similarly, if the set-group-ID mode bit of the new process image file is set, the effective group ID of the new process image is set to the group ID of the new process image file. The real user ID and real group ID of the new process image remain the same as those of the calling process image. The effective user ID and effective group ID of the new process image are saved (as the saved set-user-ID and the saved set-group-ID for use by `setuid(2)`).

If the process has the `PRIV_PROC_OWNER` privilege, the set-user-ID and set-group-ID bits will be honored when the process is being controlled by `ptrace()`.

Any shared memory segments attached to the calling process image will not be attached to the new process image (see `shmop(2)`). Any mappings established through `mmap()` are not preserved across an `exec`. Memory mappings created in the process are unmapped before the address space is rebuilt for the new process image. (see `mmap(2)`).

Memory locks established by the calling process via calls to `mlockall(3C)` or `mlock(3C)` are removed. If locked pages in the address space of the calling process are also mapped into the address spaces the locks established by the other processes will be unaffected by the call by this process to the `exec` function. If the `exec` function fails, the effect on memory locks is unspecified.

If `_XOPEN_REALTIME` is defined and has a value other than `-1`, any named semaphores open in the calling process are closed as if by appropriate calls to `sem_close(3RT)`

execve(2)

Profiling is disabled for the new process; see `profil(2)`.

Timers created by the calling process with `timer_create(3RT)` are deleted before replacing the current process image with the new process image.

For the `SCHED_FIFO` and `SCHED_RR` scheduling policies, the policy and priority settings are not changed by a call to an `exec` function.

All open message queue descriptors in the calling process are closed, as described in `mq_close(3RT)`.

Any outstanding asynchronous I/O operations may be cancelled. Those asynchronous I/O operations that are not canceled will complete as if the `exec()` function had not yet occurred, but any associated signal notifications are suppressed. It is unspecified whether the `exec()` function itself blocks awaiting such I/O completion. In no event, however, will the new process image created by the `exec()` function be affected by the presence of outstanding asynchronous I/O operations at the time the `exec()` function is called.

The new process also inherits the following attributes from the calling process:

- nice value (see `nice(2)`)
- scheduler class and priority (see `priocntl(2)`)
- process ID
- parent process ID
- process group ID
- supplementary group IDs
- `semadj` values (see `semop(2)`)
- session membership (see `exit(2)` and `signal(3C)`)
- real user ID
- real group ID
- trace flag (see `ptrace(2)` request 0)
- time left until an alarm clock signal (see `alarm(2)`)
- current working directory
- root directory
- file mode creation mask (see `umask(2)`)
- file size limit (see `ulimit(2)`)
- resource limits (see `getrlimit(2)`)
- `tms_utime`, `tms_stime`, `tms_cutime`, and `tms_cstime` (see `times(2)`)
- file-locks (see `fcntl(2)` and `lockf(3C)`)
- controlling terminal
- process signal mask (see `sigprocmask(2)`)
- pending signals (see `sigpending(2)`)
- clearance (see `getclearance(2)`)
- sensitivity label (see `getcmwlabel(2)`)
- inheritable privilege set (see `getppriv(2)`)
- process attribute flags (see `getpattr(2)`)

execve(2)

A call to any `exec` function from a process with more than one thread results in all threads being terminated and the new executable image being loaded and executed. No destructor functions will be called.

The four privilege sets of the new process are updated as described in the following equations where $E1, P1, S1, I1$ are the four privilege sets of the calling process; $E2, P2, S2, I2$ are the four privilege sets of the new process; and F and A are the forced set and the allowed set of the program file:

$$\begin{aligned} E2 &= P2 = (I1 \text{ union } F) \text{ intersect } A_i \\ S2 &= I1 \text{ intersect } A \\ I2 &= I1 \end{aligned}$$

When a script file is run, the resulting forced privileges are the combination of the forced privileges of the script and the forced privileges of the interpreter program; and the resulting allowed privileges are the allowed privileges of the interpreter program. The privilege update equations for a script executable could be expressed like this:

$$\begin{aligned} E2 &= P2 = (I1 \text{ union } F_s \text{ union } F_i) \text{ intersect } A_i \\ S2 &= I1 \text{ intersect } A_i \\ I2 &= I1 \text{ where} \end{aligned}$$

F_s is the forced privilege set of the script, F_i is the forced privilege set of the interpreter program, and A_i is the allowed privilege set of the interpreter program.

Upon successful completion, each of the functions in the `exec` family marks for update the `st_atime` field of the file. If an `exec` function failed but was able to locate the *process image file*, whether the `st_atime` field is marked for update is unspecified. Should the function succeed, the process image file is considered to have been opened with `open(2)`. The corresponding `close(2)` is considered to occur at a time after this open, but before process termination or successful completion of a subsequent call to one of the `exec` functions. The `argv[]` and `envp[]` arrays of pointers and the strings to which those arrays point will not be modified by a call to one of the `exec` functions, except as a consequence of replacing the process image. The `argv[]` and `envp[]` arrays of pointers and the strings to which those arrays point will not be modified by a call to one of the `exec` functions, except as a consequence of replacing the process image.

The saved resource limits in the new process image are set to be a copy of the process's corresponding hard and soft limits.

RETURN VALUES

If a function in the `exec` family returns to the calling process image, an error has occurred; the return value is `-1` and `errno` is set to indicate the error.

ERRORS

The `exec` functions will fail if:

<code>E2BIG</code>	The number of bytes in the new process's argument list is greater than the system-imposed limit of <code>ARG_MAX</code> bytes. The argument
--------------------	---

execve(2)

	list limit is sum of the size of the argument list plus the size of the environment's exported shell variables.
EACCES	Search permission is denied for a directory listed in the new process file's path prefix; the new process file is not an ordinary file; or the new process file mode denies execute permission. Moreover, the calling process does not have <code>PRIV_FILE_DAC_SEARCH</code> and/or <code>PRIV_FILE_MAC_SEARCH</code> to override the restriction.
EAGAIN	Total amount of system memory available when reading using raw I/O is temporarily insufficient.
EFAULT	An argument points to an illegal address.
EINTR	A signal was caught during the execution of one of the functions in the <code>exec</code> family.
ELOOP	Too many symbolic links were encountered in translating <i>path</i> or <i>file</i> .
ENAMETOOLONG	The length of the <i>file</i> or <i>path</i> argument exceeds <code>PATH_MAX</code> , or the length of a <i>file</i> or <i>path</i> component exceeds <code>{NAME_MAX}</code> while <code>{_POSIX_NO_TRUNC}</code> is in effect.
ENOENT	One or more components of the new process path name of the file do not exist or is a null pathname.
ENOLINK	The <i>path</i> argument points to a remote machine and the link to that machine is no longer active.
ENOTDIR	A component of the new process path of the file prefix is not a directory.
The <code>exec</code> functions, except for <code>execlp()</code> and <code>execvp()</code> , will fail if:	
ENOEXEC	The new process image file has the appropriate access permission but is not in the proper format.
The <code>exec</code> functions may fail if:	
ENAMETOOLONG	Pathname resolution of a symbolic link produced an intermediate result whose length exceeds <code>PATH_MAX</code> .
ENOMEM	The new process image requires more memory than <code>RLIMIT_VMEM</code> , the limit imposed by <code>setrlimit()</code> (see <code>brk(2)</code>).
ETXTBSY	The new process image file is a pure procedure (shared text) file that is currently open for writing by some process.
USAGE	As the state of conversion descriptors and message catalogue descriptors in the new process image is undefined, portable applications should not rely on their use and should close them prior to calling one of the <code>exec</code> functions.

execve(2)

Applications that require other than the default POSIX locale should call `setlocale(3C)` with the appropriate parameters to establish the locale of the new process.

The *environ* array should not be accessed directly by the application.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	<code>execle()</code> and <code>execve()</code> are Async-Signal-Safe

SUMMARY OF TRUSTED SOLARIS CHANGES

MAC search and execute permissions on the executable object are required. Process privilege sets are updated upon execution of the program. Other Trusted Solaris process attributes, such as clearance, sensitivity label, and process attribute flags, are unchanged.

Trusted Solaris 8 4/01 Reference Manual

`chmod(2)`, `fcntl(2)`, `fork(2)`, `getrlimit(2)`, `nice(2)`, `prctl(2)`, `semop(2)`, `shmop(2)`, `setlocale(3C)`,

SunOS 5.8 Reference Manual

`ksh(1)`, `ps(1)`, `sh(1)`, `alarm(2)`, `brk(2)`, `exit(2)`, `mmap(2)`, `profil(2)`, `ptrace(2)`, `sigpending(2)`, `sigprocmask(2)`, `times(2)`, `umask(2)`, `lockf(3C)`, `signal(3C)`, `system(3C)`, `timer_create(3RT)`, `a.out(4)`, `attributes(5)`, `environ(5)`, `standards(5)`

execvp(2)

NAME	exec, execl, execv, execl, execve, execlp, execvp – execute a file
SYNOPSIS	<pre>#include <unistd.h> int execl(const char *path, const char *arg0, ..., const char *argn, char * /*NULL*/); int execv(const char *path, char *const argv[]); int execl(const char *path, const char *arg0, ..., const char *argn, char * /*NULL*/, char *const envp[]); int execve(const char *path, char *const argv[], char *const envp[]); int execlp(const char *file, const char *arg0, ..., const char *argn, char * /*NULL*/); int execvp(const char *file, char *const argv[]);</pre>
DESCRIPTION	<p>Each of the functions in the <code>exec</code> family replace the current process image with a new process image. The new image is constructed from a regular, executable file called the <i>new process image file</i>. This file is either an executable object file or a file of data for an interpreter. There is no return from a successful call to one of these functions because the calling process image is overlaid by the new process image.</p> <p>An interpreter file begins with a line of the form</p> <pre>#! pathname [arg]</pre> <p>where <i>pathname</i> is the path of the interpreter, and <i>arg</i> is an optional argument. When an interpreter file is executed, the system invokes the specified interpreter. The <i>pathname</i> specified in the interpreter file is passed as <i>arg0</i> to the interpreter. If <i>arg</i> was specified in the interpreter file, it is passed as <i>arg1</i> to the interpreter. The remaining arguments to the interpreter are <i>arg0</i> through <i>argn</i> of the originally <code>exec'd</code> file. The interpreter named by <i>pathname</i> must not be an interpreter file.</p> <p>When a C-language program is executed as a result of this call, it is entered as a C-language function call as follows:</p> <pre>int main (int argc, char *argv[], char *envp[]);</pre> <p>where <i>argc</i> is the argument count, <i>argv</i> is an array of character pointers to the arguments themselves, and <i>envp</i> is an array of character pointers to the environment strings. The <i>argv</i> and <i>environ</i> arrays are each terminated by a null pointer. The null pointer terminating the <i>argv</i> array is not counted in <i>argc</i>. As indicated, <i>argc</i> is at least one, and the first member of the array points to a string containing the name of the file.</p> <p>The arguments specified by a program with one of the <code>exec</code> functions are passed on to the new process image in the <code>main()</code> arguments.</p>

The *path* argument points to a path name that identifies the new process image file.

The *file* argument is used to construct a pathname that identifies the new process image file. If the *file* argument contains a slash character, it is used as the pathname for this file. Otherwise, the path prefix for this file is obtained by a search of the directories passed in the PATH environment variable (see `environ(5)`). The environment is supplied typically by the shell. If the process image file is not a valid executable object file, `execvp()` and `execvp()` use the contents of that file as standard input to the shell. In this case, the shell becomes the new process image. In a standard-conforming application (see `standards(5)`), the `exec` family of functions use `/usr/bin/ksh` (see `ksh(1)`); otherwise, they use `/usr/bin/sh` (see `sh(1)`).

The arguments represented by *arg0...* are pointers to null-terminated character strings. These strings constitute the argument list available to the new process image. The list is terminated by a null pointer. The *arg0* argument should point to a filename that is associated with the process being started by one of the `exec` functions.

The *argv* argument is an array of character pointers to null-terminated strings. The last member of this array must be a null pointer. These strings constitute the argument list available to the new process image. The value in *argv*[0] should point to a filename that is associated with the process being started by one of the `exec` functions.

The *envp* argument is an array of character pointers to null-terminated strings. These strings constitute the environment for the new process image. The *envp* array is terminated by a null pointer. For `exec1()`, `execv()`, `execvp()`, and `execvp()`, the C-language run-time start-off routine places a pointer to the environment of the calling process in the global object `extern char **environ`, and it is used to pass the environment of the calling process to the new process image.

The number of bytes available for the new process's combined argument and environment lists is ARG_MAX. It is implementation-dependent whether null terminators, pointers, and/or any alignment bytes are included in this total.

The calling process must have read and execute access to the new process file or have the following in its set of effective privileges:

```
PRIV_FILE_DAC_SEARCH
PRIV_FILE_DAC_EXECUTE
PRIV_FILE_MAC_SEARCH
PRIV_FILE_MAC_READ
```

File descriptors open in the calling process image remain open in the new process image, except for those whose close-on-exec flag `FD_CLOEXEC` is set; (see `fcntl(2)`). For those file descriptors that remain open, all attributes of the open file description, including file locks, remain unchanged.

Directory streams open in the calling process image are closed in the new process image.

execvp(2)

The state of conversion descriptors and message catalogue descriptors in the new process image is undefined. For the new process, the equivalent of:

`setlocale(LC_ALL, "C")` is executed at startup.

Signals set to the default action (`SIG_DFL`) in the calling process image are set to the default action in the new process image (see `signal(3C)`). Signals set to be ignored (`SIG_IGN`) by the calling process image are set to be ignored by the new process image. Signals set to be caught by the calling process image are set to the default action in the new process image (see `signal(3HEAD)`). After a successful call to any of the `exec` functions, alternate signal stacks are not preserved and the `SA_ONSTACK` flag is cleared for all signals.

After a successful call to any of the `exec` functions, any functions previously registered by `atexit(3C)` are no longer registered.

The saved resource limits in the new process image are set to be a copy of the process's corresponding hard and soft resource limits.

If the `ST_NOSUID` bit is set for the file system containing the new process image file, then the effective user ID, effective group ID, saved set-user-ID, and saved set-group-ID are unchanged in the new process image. If the set-user-ID mode bit of the new process image file is set (see `chmod(2)`), the effective user ID of the new process image is set to the owner ID of the new process image file. Similarly, if the set-group-ID mode bit of the new process image file is set, the effective group ID of the new process image is set to the group ID of the new process image file. The real user ID and real group ID of the new process image remain the same as those of the calling process image. The effective user ID and effective group ID of the new process image are saved (as the saved set-user-ID and the saved set-group-ID for use by `setuid(2)`).

If the process has the `PRIV_PROC_OWNER` privilege, the set-user-ID and set-group-ID bits will be honored when the process is being controlled by `ptrace()`.

Any shared memory segments attached to the calling process image will not be attached to the new process image (see `shmop(2)`). Any mappings established through `mmap()` are not preserved across an `exec`. Memory mappings created in the process are unmapped before the address space is rebuilt for the new process image. (see `mmap(2)`).

Memory locks established by the calling process via calls to `mlockall(3C)` or `mlock(3C)` are removed. If locked pages in the address space of the calling process are also mapped into the address spaces the locks established by the other processes will be unaffected by the call by this process to the `exec` function. If the `exec` function fails, the effect on memory locks is unspecified.

If `_XOPEN_REALTIME` is defined and has a value other than `-1`, any named semaphores open in the calling process are closed as if by appropriate calls to `sem_close(3RT)`

Profiling is disabled for the new process; see `profil(2)`.

Timers created by the calling process with `timer_create(3RT)` are deleted before replacing the current process image with the new process image.

For the `SCHED_FIFO` and `SCHED_RR` scheduling policies, the policy and priority settings are not changed by a call to an `exec` function.

All open message queue descriptors in the calling process are closed, as described in `mq_close(3RT)`.

Any outstanding asynchronous I/O operations may be cancelled. Those asynchronous I/O operations that are not canceled will complete as if the `exec()` function had not yet occurred, but any associated signal notifications are suppressed. It is unspecified whether the `exec()` function itself blocks awaiting such I/O completion. In no event, however, will the new process image created by the `exec()` function be affected by the presence of outstanding asynchronous I/O operations at the time the `exec()` function is called.

The new process also inherits the following attributes from the calling process:

- nice value (see `nice(2)`)
- scheduler class and priority (see `prctl(2)`)
- process ID
- parent process ID
- process group ID
- supplementary group IDs
- `semadj` values (see `semop(2)`)
- session membership (see `exit(2)` and `signal(3C)`)
- real user ID
- real group ID
- trace flag (see `ptrace(2)` request 0)
- time left until an alarm clock signal (see `alarm(2)`)
- current working directory
- root directory
- file mode creation mask (see `umask(2)`)
- file size limit (see `ulimit(2)`)
- resource limits (see `getrlimit(2)`)
- `tms_utime`, `tms_stime`, `tms_cutime`, and `tms_cstime` (see `times(2)`)
- file-locks (see `fcntl(2)` and `lockf(3C)`)
- controlling terminal
- process signal mask (see `sigprocmask(2)`)
- pending signals (see `sigpending(2)`)
- clearance (see `getclearance(2)`)
- sensitivity label (see `getcmwlabel(2)`)
- inheritable privilege set (see `getppriv(2)`)
- process attribute flags (see `getpattr(2)`)

execvp(2)

A call to any `exec` function from a process with more than one thread results in all threads being terminated and the new executable image being loaded and executed. No destructor functions will be called.

The four privilege sets of the new process are updated as described in the following equations where $E1$, $P1$, $S1$, $I1$ are the four privilege sets of the calling process; $E2$, $P2$, $S2$, $I2$ are the four privilege sets of the new process; and F and A are the forced set and the allowed set of the program file:

$$\begin{aligned} E2 &= P2 = (I1 \text{ union } F) \text{ intersect } A_i \\ S2 &= I1 \text{ intersect } A \\ I2 &= I1 \end{aligned}$$

When a script file is run, the resulting forced privileges are the combination of the forced privileges of the script and the forced privileges of the interpreter program; and the resulting allowed privileges are the allowed privileges of the interpreter program. The privilege update equations for a script executable could be expressed like this:

$$\begin{aligned} E2 &= P2 = (I1 \text{ union } F_s \text{ union } F_i) \text{ intersect } A_i \\ S2 &= I1 \text{ intersect } A_i \\ I2 &= I1 \text{ where} \end{aligned}$$

F_s is the forced privilege set of the script, F_i is the forced privilege set of the interpreter program, and A_i is the allowed privilege set of the interpreter program.

Upon successful completion, each of the functions in the `exec` family marks for update the `st_atime` field of the file. If an `exec` function failed but was able to locate the *process image file*, whether the `st_atime` field is marked for update is unspecified. Should the function succeed, the process image file is considered to have been opened with `open(2)`. The corresponding `close(2)` is considered to occur at a time after this open, but before process termination or successful completion of a subsequent call to one of the `exec` functions. The `argv[]` and `envp[]` arrays of pointers and the strings to which those arrays point will not be modified by a call to one of the `exec` functions, except as a consequence of replacing the process image. The `argv[]` and `envp[]` arrays of pointers and the strings to which those arrays point will not be modified by a call to one of the `exec` functions, except as a consequence of replacing the process image.

The saved resource limits in the new process image are set to be a copy of the process's corresponding hard and soft limits.

RETURN VALUES

If a function in the `exec` family returns to the calling process image, an error has occurred; the return value is `-1` and `errno` is set to indicate the error.

ERRORS

The `exec` functions will fail if:

<code>E2BIG</code>	The number of bytes in the new process's argument list is greater than the system-imposed limit of <code>ARG_MAX</code> bytes. The argument
--------------------	---

	list limit is sum of the size of the argument list plus the size of the environment's exported shell variables.
EACCES	Search permission is denied for a directory listed in the new process file's path prefix; the new process file is not an ordinary file; or the new process file mode denies execute permission. Moreover, the calling process does not have <code>PRIV_FILE_DAC_SEARCH</code> and/or <code>PRIV_FILE_MAC_SEARCH</code> to override the restriction.
EAGAIN	Total amount of system memory available when reading using raw I/O is temporarily insufficient.
EFAULT	An argument points to an illegal address.
EINTR	A signal was caught during the execution of one of the functions in the <code>exec</code> family.
ELOOP	Too many symbolic links were encountered in translating <i>path</i> or <i>file</i> .
ENAMETOOLONG	The length of the <i>file</i> or <i>path</i> argument exceeds <code>PATH_MAX</code> , or the length of a <i>file</i> or <i>path</i> component exceeds <code>{NAME_MAX}</code> while <code>{_POSIX_NO_TRUNC}</code> is in effect.
ENOENT	One or more components of the new process path name of the file do not exist or is a null pathname.
ENOLINK	The <i>path</i> argument points to a remote machine and the link to that machine is no longer active.
ENOTDIR	A component of the new process path of the file prefix is not a directory.
The <code>exec</code> functions, except for <code>execvp()</code> and <code>execvp()</code> , will fail if:	
ENOEXEC	The new process image file has the appropriate access permission but is not in the proper format.
The <code>exec</code> functions may fail if:	
ENAMETOOLONG	Pathname resolution of a symbolic link produced an intermediate result whose length exceeds <code>PATH_MAX</code> .
ENOMEM	The new process image requires more memory than <code>RLIMIT_VMEM</code> , the limit imposed by <code>setrlimit()</code> (see <code>brk(2)</code>).
ETXTBSY	The new process image file is a pure procedure (shared text) file that is currently open for writing by some process.
USAGE	As the state of conversion descriptors and message catalogue descriptors in the new process image is undefined, portable applications should not rely on their use and should close them prior to calling one of the <code>exec</code> functions.

execvp(2)

Applications that require other than the default POSIX locale should call `setlocale(3C)` with the appropriate parameters to establish the locale of the new process.

The *environ* array should not be accessed directly by the application.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	<code>execle()</code> and <code>execve()</code> are Async-Signal-Safe

SUMMARY OF TRUSTED SOLARIS CHANGES

MAC search and execute permissions on the executable object are required. Process privilege sets are updated upon execution of the program. Other Trusted Solaris process attributes, such as clearance, sensitivity label, and process attribute flags, are unchanged.

Trusted Solaris 8 4/01 Reference Manual

`chmod(2)`, `fcntl(2)`, `fork(2)`, `getrlimit(2)`, `nice(2)`, `priocntl(2)`, `semop(2)`, `shmop(2)`, `setlocale(3C)`,

SunOS 5.8 Reference Manual

`ksh(1)`, `ps(1)`, `sh(1)`, `alarm(2)`, `brk(2)`, `exit(2)`, `mmap(2)`, `profil(2)`, `ptrace(2)`, `sigpending(2)`, `sigprocmask(2)`, `times(2)`, `umask(2)`, `lockf(3C)`, `signal(3C)`, `system(3C)`, `timer_create(3RT)`, `a.out(4)`, `attributes(5)`, `environ(5)`, `standards(5)`

NAME	acl, facl – Get or set a file's Access Control List (ACL)						
SYNOPSIS	<pre>#include <sys/acl.h> int acl(char *pathp, int cmd, int nentries, aclent_t *aclbufp); int facl(int fildes, int cmd, int nentries, aclent_t *aclbufp);</pre>						
DESCRIPTION	<p>The <code>acl()</code> and <code>facl()</code> functions get or set the ACL of a file whose name is given by <i>pathp</i> or referenced by the open file descriptor <i>fildes</i>. The <i>nentries</i> argument specifies how many ACL entries fit into buffer <i>aclbufp</i>. The <code>acl()</code> function is used to manipulate ACL on file system objects.</p> <p>The following values for <i>cmd</i> are supported:</p> <table> <tr> <td>SETACL</td><td><i>nentries</i> ACL entries, specified in buffer <i>aclbufp</i>, are stored in the file's ACL. This command can be executed only by a process that has an effective user ID equal to the owner of the file. To override this restriction, the calling process may assert the <code>PRIV_FILE_SETDAC</code> privilege.</td></tr> <tr> <td>GETACL</td><td>Buffer <i>aclbufp</i> is filled with the file's ACL entries. Read access to the file is not required, but all directories in the path name must be searchable.</td></tr> <tr> <td>GETACLCNT</td><td>The number of entries in the file's ACL is returned. Read access to the file is not required, but all directories in the path name must be searchable.</td></tr> </table>	SETACL	<i>nentries</i> ACL entries, specified in buffer <i>aclbufp</i> , are stored in the file's ACL. This command can be executed only by a process that has an effective user ID equal to the owner of the file. To override this restriction, the calling process may assert the <code>PRIV_FILE_SETDAC</code> privilege.	GETACL	Buffer <i>aclbufp</i> is filled with the file's ACL entries. Read access to the file is not required, but all directories in the path name must be searchable.	GETACLCNT	The number of entries in the file's ACL is returned. Read access to the file is not required, but all directories in the path name must be searchable.
SETACL	<i>nentries</i> ACL entries, specified in buffer <i>aclbufp</i> , are stored in the file's ACL. This command can be executed only by a process that has an effective user ID equal to the owner of the file. To override this restriction, the calling process may assert the <code>PRIV_FILE_SETDAC</code> privilege.						
GETACL	Buffer <i>aclbufp</i> is filled with the file's ACL entries. Read access to the file is not required, but all directories in the path name must be searchable.						
GETACLCNT	The number of entries in the file's ACL is returned. Read access to the file is not required, but all directories in the path name must be searchable.						
RETURN VALUES	<p>Upon successful completion, <code>acl()</code> and <code>facl()</code> return 0 if <i>cmd</i> is SETACL. If <i>cmd</i> is GETACL or GETACLCNT, the number of ACL entries is returned. Otherwise, -1 is returned and <code>errno</code> is set to indicate the error.</p> <p>The audit record has multiple events that represent the requested function. For SETACL, the audit record includes the old and new ACLs.</p>						
ERRORS	<p>The <code>acl()</code> function will fail if:</p> <table> <tr> <td>EACCESS</td><td> <p>The caller does not have access to a component of the pathname. To override this restriction, the calling process may assert one or both of these privileges: <code>PRIV_FILE_DAC_SEARCH</code> and <code>PRIV_FILE_MAC_SEARCH</code>.</p> <p>The <i>cmd</i> argument is SETACL and <i>nentries</i> is less than three.</p> <p>The <i>cmd</i> argument is SETACL and the ACL specified in <i>aclbufp</i> is not valid.</p> </td></tr> <tr> <td>EFAULT</td><td>The <i>pathp</i> or <i>aclbufp</i> argument points to an illegal address.</td></tr> <tr> <td>EINVAL</td><td> <p>The <i>cmd</i> argument is not GETACL, SETACL, or GETACLCNT; the <i>cmd</i> argument is SETACL and <i>nentries</i> is less than 3; or the <i>cmd</i> argument is SETACL and the ACL specified in <i>aclbufp</i> is not valid.</p> </td></tr> </table>	EACCESS	<p>The caller does not have access to a component of the pathname. To override this restriction, the calling process may assert one or both of these privileges: <code>PRIV_FILE_DAC_SEARCH</code> and <code>PRIV_FILE_MAC_SEARCH</code>.</p> <p>The <i>cmd</i> argument is SETACL and <i>nentries</i> is less than three.</p> <p>The <i>cmd</i> argument is SETACL and the ACL specified in <i>aclbufp</i> is not valid.</p>	EFAULT	The <i>pathp</i> or <i>aclbufp</i> argument points to an illegal address.	EINVAL	<p>The <i>cmd</i> argument is not GETACL, SETACL, or GETACLCNT; the <i>cmd</i> argument is SETACL and <i>nentries</i> is less than 3; or the <i>cmd</i> argument is SETACL and the ACL specified in <i>aclbufp</i> is not valid.</p>
EACCESS	<p>The caller does not have access to a component of the pathname. To override this restriction, the calling process may assert one or both of these privileges: <code>PRIV_FILE_DAC_SEARCH</code> and <code>PRIV_FILE_MAC_SEARCH</code>.</p> <p>The <i>cmd</i> argument is SETACL and <i>nentries</i> is less than three.</p> <p>The <i>cmd</i> argument is SETACL and the ACL specified in <i>aclbufp</i> is not valid.</p>						
EFAULT	The <i>pathp</i> or <i>aclbufp</i> argument points to an illegal address.						
EINVAL	<p>The <i>cmd</i> argument is not GETACL, SETACL, or GETACLCNT; the <i>cmd</i> argument is SETACL and <i>nentries</i> is less than 3; or the <i>cmd</i> argument is SETACL and the ACL specified in <i>aclbufp</i> is not valid.</p>						

facl(2)

EIO	A disk I/O error has occurred while storing or retrieving the ACL.
ENOENT	A component of the path does not exist.
ENOSPC	The <i>cmd</i> argument is GETACL and <i>nentries</i> is less than the number of entries in the file's ACL, or the <i>cmd</i> argument is SETACL and there is insufficient space in the file system to store the ACL.
ENOTDIR	A component of the path specified by <i>pathp</i> is not a directory, or the <i>cmd</i> argument is SETACL and an attempt is made to set a default ACL on a file type other than a directory.
ENOSYS	The <i>cmd</i> argument is SETACL and the file specified by <i>pathp</i> resides on a file system that does not support ACLs, or the <code>acl ()</code> function is not supported by this implementation.
EPERM	The <i>cmd</i> argument is SETACL and the effective user ID of the caller does not match the owner of the file. To override this restriction, the calling process may assert the <code>PRIV_FILE_SETDAC</code> privilege.
EROFS	The <i>cmd</i> argument is SETACL and the file specified by <i>pathp</i> resides on a file system that is mounted read-only.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Interface Stability	Evolving

**SUMMARY OF
TRUSTED
SOLARIS
CHANGES**

Appropriate privilege is required to override access or ownership checks.

The audit record has multiple events that represent the requested function. For SETACL the audit record includes the old and new ACLs.

**SunOS 5.8
Reference Manual**

`getfacl(1)`, `setfacl(1)`, `aclcheck(3SEC)`, `aclsort(3SEC)`, `attributes(5)`

NAME	chdir, fchdir – change working directory																		
SYNOPSIS	<pre>#include <unistd.h> int chdir(const char *path) ; int fchdir(int fildes) ;</pre>																		
DESCRIPTION	<p>The <code>chdir()</code> and <code>fchdir()</code> functions cause a directory pointed to by <i>path</i> or <i>fildes</i> to become the current working directory. The starting point for path searches for path names not beginning with / (slash). The <i>path</i> argument points to the path name of a directory. The <i>fildes</i> argument is an open file descriptor of a directory.</p> <p>For a directory to become the current directory, a process must have execute (search) access to the directory.</p>																		
RETURN VALUES	<p><code>chdir()</code> returns:</p> <p>0 On success.</p> <p>-1 On failure, and sets <code>errno</code> to indicate the error.</p>																		
ERRORS	<p>The <code>chdir()</code> function will fail if:</p> <table> <tr> <td>EACCES</td><td>Search permission is denied for some component of <i>path</i>. To override this restriction, the calling process may assert one or both of these privileges: <code>PRIV_FILE_DAC_SEARCH</code> and <code>PRIV_FILE_MAC_SEARCH</code>.</td></tr> <tr> <td>EFAULT</td><td>The <i>path</i> argument points to an illegal address.</td></tr> <tr> <td>EINTR</td><td>A signal was caught during the execution of the <code>chdir()</code> function.</td></tr> <tr> <td>EIO</td><td>An I/O error occurred while reading from or writing to the file system.</td></tr> <tr> <td>ELOOP</td><td>Too many symbolic links were encountered in translating <i>path</i>.</td></tr> <tr> <td>ENAMETOOLONG</td><td>The length of the <i>path</i> argument exceeds <code>PATH_MAX</code>, or the length of a <i>path</i> component exceeds <code>NAME_MAX</code> while <code>_POSIX_NO_TRUNC</code> is in effect.</td></tr> <tr> <td>ENOENT</td><td>Either a component of the path prefix or the directory named by <i>path</i> does not exist or is a null pathname.</td></tr> <tr> <td>ENOLINK</td><td>The <i>path</i> argument points to a remote machine and the link to that machine is no longer active.</td></tr> <tr> <td>ENOTDIR</td><td>A component of the path name is not a directory.</td></tr> </table> <p>The <code>fchdir()</code> function will fail if:</p>	EACCES	Search permission is denied for some component of <i>path</i> . To override this restriction, the calling process may assert one or both of these privileges: <code>PRIV_FILE_DAC_SEARCH</code> and <code>PRIV_FILE_MAC_SEARCH</code> .	EFAULT	The <i>path</i> argument points to an illegal address.	EINTR	A signal was caught during the execution of the <code>chdir()</code> function.	EIO	An I/O error occurred while reading from or writing to the file system.	ELOOP	Too many symbolic links were encountered in translating <i>path</i> .	ENAMETOOLONG	The length of the <i>path</i> argument exceeds <code>PATH_MAX</code> , or the length of a <i>path</i> component exceeds <code>NAME_MAX</code> while <code>_POSIX_NO_TRUNC</code> is in effect.	ENOENT	Either a component of the path prefix or the directory named by <i>path</i> does not exist or is a null pathname.	ENOLINK	The <i>path</i> argument points to a remote machine and the link to that machine is no longer active.	ENOTDIR	A component of the path name is not a directory.
EACCES	Search permission is denied for some component of <i>path</i> . To override this restriction, the calling process may assert one or both of these privileges: <code>PRIV_FILE_DAC_SEARCH</code> and <code>PRIV_FILE_MAC_SEARCH</code> .																		
EFAULT	The <i>path</i> argument points to an illegal address.																		
EINTR	A signal was caught during the execution of the <code>chdir()</code> function.																		
EIO	An I/O error occurred while reading from or writing to the file system.																		
ELOOP	Too many symbolic links were encountered in translating <i>path</i> .																		
ENAMETOOLONG	The length of the <i>path</i> argument exceeds <code>PATH_MAX</code> , or the length of a <i>path</i> component exceeds <code>NAME_MAX</code> while <code>_POSIX_NO_TRUNC</code> is in effect.																		
ENOENT	Either a component of the path prefix or the directory named by <i>path</i> does not exist or is a null pathname.																		
ENOLINK	The <i>path</i> argument points to a remote machine and the link to that machine is no longer active.																		
ENOTDIR	A component of the path name is not a directory.																		

fchdir(2)

EACCES	Search permission is denied for <i>fil</i> des. To override this restriction, the calling process may assert one or both of these privileges: PRIV_FILE_DAC_SEARCH and PRIV_FILE_MAC_SEARCH.
EBADF	The <i>fil</i> des argument is not an open file descriptor.
EINTR	A signal was caught during the execution of the fchdir () function.
EIO	An I/O error occurred while reading from or writing to the file system.
ENOLINK	The <i>fil</i> des argument points to a remote machine and the link to that machine is no longer active.
ENOTDIR	The open file descriptor <i>fil</i> des does not refer to a directory.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	chdir () is Async-Signal-Safe

SUMMARY OF
TRUSTED
SQUARIS
Trusted Squares
4/01 CRANGES
Manual
Reference Manual

Appropriate privilege is required to override access checks.

chroot(2)

attributes(5)

NAME	chmod, fchmod – change access permission mode of file																																													
SYNOPSIS	<pre>#include <sys/types.h> #include <sys/stat.h> int chmod(const char *path, mode_t mode); int fchmod(int fildes, mode_t mode);</pre>																																													
DESCRIPTION	<p>The chmod() and fchmod() functions set the access permission portion of the mode of the file whose name is given by <i>path</i> or referenced by the open file descriptor <i>fildes</i> to the bit pattern contained in <i>mode</i>. Access permission bits are interpreted as follows:</p> <table><tr><td>S_ISUID</td><td>04000</td><td>Set user ID on execution.</td></tr><tr><td>S_ISGID</td><td>020#0</td><td>Set group ID on execution if # is 7, 5, 3, or 1. Enable mandatory file/record locking if # is 6, 4, 2, or 0.</td></tr><tr><td>S_ISVTX</td><td>01000</td><td>Save text image after execution.</td></tr><tr><td>S_IRWXU</td><td>00700</td><td>Read, write, execute by owner.</td></tr><tr><td>S_IRUSR</td><td>00400</td><td>Read by owner.</td></tr><tr><td>S_IWUSR</td><td>00200</td><td>Write by owner.</td></tr><tr><td>S_IXUSR</td><td>00100</td><td>Execute (search if a directory) by owner.</td></tr><tr><td>S_IRWXG</td><td>00070</td><td>Read, write, execute by group.</td></tr><tr><td>S_IRGRP</td><td>00040</td><td>Read by group.</td></tr><tr><td>S_IWGRP</td><td>00020</td><td>Write by group.</td></tr><tr><td>S_IXGRP</td><td>00010</td><td>Execute by group.</td></tr><tr><td>S_IRWXO</td><td>00007</td><td>Read, write, execute (search) by others.</td></tr><tr><td>S_IROTH</td><td>00004</td><td>Read by others.</td></tr><tr><td>S_IWOTH</td><td>00002</td><td>Write by others.</td></tr><tr><td>S_IXOTH</td><td>00001</td><td>Execute by others.</td></tr></table> <p>Modes are constructed by the bitwise OR operation of the access permission bits.</p> <p>The effective user ID of the process must match the owner of the file or the process must have the PRIV_FILE_SETDAC privilege to change the mode of a file.</p> <p>If the process is not a privileged process and the file is not a directory, mode bit 01000 (save text image on execution) is cleared. The calling process may assert the PRIV_SYS_CONFIG privilege to override this restriction.</p>	S_ISUID	04000	Set user ID on execution.	S_ISGID	020#0	Set group ID on execution if # is 7, 5, 3, or 1. Enable mandatory file/record locking if # is 6, 4, 2, or 0.	S_ISVTX	01000	Save text image after execution.	S_IRWXU	00700	Read, write, execute by owner.	S_IRUSR	00400	Read by owner.	S_IWUSR	00200	Write by owner.	S_IXUSR	00100	Execute (search if a directory) by owner.	S_IRWXG	00070	Read, write, execute by group.	S_IRGRP	00040	Read by group.	S_IWGRP	00020	Write by group.	S_IXGRP	00010	Execute by group.	S_IRWXO	00007	Read, write, execute (search) by others.	S_IROTH	00004	Read by others.	S_IWOTH	00002	Write by others.	S_IXOTH	00001	Execute by others.
S_ISUID	04000	Set user ID on execution.																																												
S_ISGID	020#0	Set group ID on execution if # is 7, 5, 3, or 1. Enable mandatory file/record locking if # is 6, 4, 2, or 0.																																												
S_ISVTX	01000	Save text image after execution.																																												
S_IRWXU	00700	Read, write, execute by owner.																																												
S_IRUSR	00400	Read by owner.																																												
S_IWUSR	00200	Write by owner.																																												
S_IXUSR	00100	Execute (search if a directory) by owner.																																												
S_IRWXG	00070	Read, write, execute by group.																																												
S_IRGRP	00040	Read by group.																																												
S_IWGRP	00020	Write by group.																																												
S_IXGRP	00010	Execute by group.																																												
S_IRWXO	00007	Read, write, execute (search) by others.																																												
S_IROTH	00004	Read by others.																																												
S_IWOTH	00002	Write by others.																																												
S_IXOTH	00001	Execute by others.																																												

fchmod(2)

If neither the process is privileged, nor the file's group is a member of the process's supplementary group list, and the effective group ID of the process does not match the group ID of the file, mode bit 02000 (set group ID on execution) is cleared.

If a directory is writable and has `S_ISVTX` (the sticky bit) set, files within that directory can be removed or renamed only if one or more of the following is true (see `unlink(2)` and `rename(2)`):

- the user owns the file
- the user owns the directory
- the file is writable by the user
- the user is a privileged user

If a directory has the set group ID bit set, a given file created within that directory will have the same group ID as the directory, if that group ID is part of the group ID set of the process that created the file. Otherwise, the newly created file's group ID will be set to the effective group ID of the creating process.

If the mode bit 02000 (set group ID on execution) is set and the mode bit 00010 (execute or search by group) is not set, mandatory file/record locking will exist on a regular file. This may affect future calls to `open(2)`, `creat(2)`, `read(2)`, and `write(2)` on this file.

Upon successful completion, `chmod()` and `fchmod()` mark for update the `st_ctime` field of the file.

RETURN VALUES

`chmod()` returns:

- 0 On success.
- 1 On failure, and sets `errno` to indicate the error.

ERRORS

The `chmod()` function will fail if:

- | | |
|--------|--|
| EACCES | Search permission is denied on a component of the path prefix of <i>path</i> . To override this restriction, the calling process may assert the <code>PRIV_FILE_DAC_SEARCH</code> privilege and/or the <code>PRIV_FILE_MAC_SEARCH</code> privilege.

The calling process does not own the final object specified in <i>path</i> or does not own <i>files</i> . To override this restriction, the calling process may assert the <code>PRIV_FILE_SETDAC</code> privilege.

Write permission is denied on <i>path</i> or <i>files</i> . To override this restriction, the calling process may assert the <code>PRIV_FILE_DAC_WRITE</code> and/or the <code>PRIV_FILE_MAC_WRITE</code> privilege. |
| EFAULT | The <i>path</i> argument points to an illegal address. |

fchmod(2)

EINTR	A signal was caught during execution of the function.
EIO	An I/O error occurred while reading from or writing to the file system.
ELOOP	Too many symbolic links were encountered in translating <i>path</i> .
ENAMETOOLONG	The length of the <i>path</i> argument exceeds <code>PATH_MAX</code> , or the length of a <i>path</i> component exceeds <code>NAME_MAX</code> while <code>_POSIX_NO_TRUNC</code> is in effect.
ENOENT	Either a component of the path prefix or the file referred to by <i>path</i> does not exist or is a null pathname.
ENOLINK	The <i>fildev</i> argument points to a remote machine and the link to that machine is no longer active.
ENOTDIR	A component of the prefix of <i>path</i> is not a directory.
EPERM	The effective user ID does not match the owner of the file. To override this restriction, the calling process may assert the <code>PRIV_FILE_SETDAC</code> privilege.
EROFS	The file referred to by <i>path</i> resides on a read-only file system.

The `fchmod()` function will fail if:

EBADF	The <i>fildev</i> argument is not an open file descriptor
EIO	An I/O error occurred while reading from or writing to the file system.
EINTR	A signal was caught during execution of the <code>fchmod()</code> function.
ENOLINK	The <i>path</i> argument points to a remote machine and the link to that machine is no longer active.
EROFS	The file referred to by <i>fildev</i> resides on a read-only file system.

ATTRIBUTES See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	<code>chmod()</code> is Async-Signal-Safe

SUMMARY OF TRUSTED SOLARIS CHANGES

Appropriate privilege is required to override access checks.

- To override a search permission error, the calling process requires the `PRIV_FILE_MAC_SEARCH` privilege.

fchmod(2)

- To override a write permission error, the calling process requires the `PRIV_FILE_DAC_WRITE` and/or the `PRIV_FILE_MAC_WRITE` privilege.
- If the calling process does not own the object, the calling process requires the `PRIV_FILE_SETDAC` privilege.

To set the sticky bit on a file, the calling process may assert the `PRIV_SYS_CONFIG` privilege.

To set the set-user-ID on a user who is not the effective user ID of the calling process, the calling process may assert the `PRIV_FILE_SETID` privilege.

To set the set-group-ID bit on a group not in effective or supplementary group IDs of the calling process, the calling process may assert the `PRIV_FILE_SETID` privilege.

**Trusted Solaris 8
4/01 Reference
Manual**

`chmod(1)`, `chown(2)`, `creat(2)`, `fcntl(2)`, `mknod(2)`, `open(2)`, `read(2)`, `rename(2)`,
`stat(2)`, `write(2)`

**SunOS 5.8
Reference Manual**

`mkfifo(3C)`, `stat(3HEAD)`, `attributes(5)`

System Interface Guide

NOTES

If you use `chmod()` to change the file group owner permissions on a file with ACL entries, both the file group owner permissions and the ACL mask are changed to the new permissions. Be aware that the new ACL mask permissions may change the effective permissions for additional users and groups who have ACL entries on the file.

NAME	chown, lchown, fchown – change owner and group of a file
SYNOPSIS	<pre>#include <unistd.h> #include <sys/types.h> int chown(const char *path, uid_t owner, gid_t group); int lchown(const char *path, uid_t owner, gid_t group); int fchown(int fildes, uid_t owner, gid_t group);</pre>
DESCRIPTION	<p>The <code>chown()</code> function sets the owner ID and group ID of the file specified by <i>path</i> or referenced by the open file descriptor <i>fildes</i> to <i>owner</i> and <i>group</i> respectively. If <i>owner</i> or <i>group</i> is specified as <code>-1</code>, <code>chown()</code> does not change the corresponding ID of the file.</p> <p>The <code>lchown()</code> function sets the owner ID and group ID of the named file in the same manner as <code>chown()</code>, unless the named file is a symbolic link. In this case, <code>lchown()</code> changes the ownership of the symbolic link file itself, while <code>chown()</code> changes the ownership of the file or directory to which the symbolic link refers.</p> <p>If <code>chown()</code>, <code>lchown()</code>, or <code>fchown()</code> is invoked, the set-user-ID and set-group-ID bits of the file mode, <code>chmod(2)</code>, respectively, are cleared. See <code>chmod(2)</code>. To bypass this restriction, the process may assert the <code>PRIV_FILE_SETID</code> privilege.</p> <p>The operating system has a configuration option, <code>_POSIX_CHOWN_RESTRICTED</code>, to restrict ownership changes for the <code>chown()</code>, <code>lchown()</code>, and <code>fchown()</code> functions. When <code>_POSIX_CHOWN_RESTRICTED</code> is not in effect, the effective user ID of the process must match the owner of the file. To override this restriction, the calling process must assert the <code>PRIV_FILE_CHOWN</code> privilege. When <code>_POSIX_CHOWN_RESTRICTED</code> is not in effect, the effective user ID of the process must match the owner of the file or the process must be the super-user to change the ownership of a file. When <code>_POSIX_CHOWN_RESTRICTED</code> is in effect, the <code>chown()</code>, <code>lchown()</code>, and <code>fchown()</code> functions require that the calling process assert the <code>PRIV_FILE_CHOWN</code> privilege to change the user ID of a file. To change the group ID of a file, the process must be the owner of the file and the new group ID must be the group of the process ID or must be in the supplementary group list of the process. To override this restriction, the calling process may assert the <code>PRIV_FILE_CHOWN</code> privilege.</p> <p>set rstchown = 1</p> <p>To disable this option, include the following line in <code>/etc/system</code>:</p> <p>set rstchown = 0</p> <p>See <code>system(4)</code> and <code>fpathconf(2)</code>.</p> <p>Upon successful completion, <code>chown()</code>, <code>fchown()</code> and <code>lchown()</code> mark for update the <code>st_ctime</code> field of the file.</p>

fchown(2)

RETURN VALUES

chown() returns:

- 0 On success.
- 1 On failure, and sets errno to indicate the error.

ERRORS

The chown() and lchown() functions will fail if:

- EACCES Search permission is denied on a component of the path prefix of *path*. To override this restriction, the calling process may assert one or both of these privileges: PRIV_FILE_DAC_SEARCH and PRIV_FILE_MAC_SEARCH.

Write permission is denied on *path* or *filde*s. To override this restriction, the calling process may assert the PRIV_FILE_MAC_WRITE privilege.
 - EFAULT The *path* argument points to an illegal address.
 - EINTR A signal was caught during the execution of the chown() or lchown() function.
 - EINVAL The *group* or *owner* argument is out of range.
 - EIO An I/O error occurred while reading from or writing to the file system.
 - ELOOP Too many symbolic links were encountered in translating *path*.
 - ENAMETOOLONG The length of the *path* argument exceeds PATH_MAX, or the length of a *path* component exceeds NAME_MAX while _POSIX_NO_TRUNC is in effect.
 - ENOLINK The *path* argument points to a remote machine and the link to that machine is no longer active.
 - ENOENT Either a component of the path prefix or the file referred to by *path* does not exist or is a null pathname.
 - ENOTDIR A component of the path prefix of *path* is not a directory.
 - EPERM The effective user ID does not match the owner of the file. If _POSIX_CHOWN_RESTRICTED is set, the calling process must assert the PRIV_FILE_CHOWN privilege. If _POSIX_CHOWN_RESTRICTED is not set, the calling process may assert the PRIV_FILE_CHOWN privilege.
 - EROFS The named file resides on a read-only file system.
- The fchown() function will fail if:
- EBADF The *filde*s argument is not an open file descriptor.

fchown(2)

EIO	An I/O error occurred while reading from or writing to the file system.
EINTR	A signal was caught during execution of the function.
ENOLINK	The <i>fildev</i> argument points to a remote machine and the link to that machine is no longer active.
EINVAL	The <i>group</i> or <i>owner</i> argument is out of range.
EPERM	The effective user ID does not match the owner of the file, or the process is not the super-user and <code>_POSIX_CHOWN_RESTRICTED</code> indicates that such privilege is required.
EROFS	The named file referred to by <i>fildev</i> resides on a read-only file system.

ATTRIBUTES See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	chown () is Async-Signal-Safe

**SUMMARY OF
TRUSTED
SOLARIS
CHANGES**

Appropriate privilege is required to override access checks.

When the ownership of *path* and *fildev* is changed, the set-user-ID and set-group-ID bits are cleared. The calling process may assert the `PRIV_FILE_SETID` privilege to bypass this restriction.

To change the user ID of the file when the calling process does not own the file and `_POSIX_CHOWN_RESTRICTED` is not in effect, the calling process may assert the `PRIV_FILE_CHOWN` privilege.

To change the group ID of the file when the calling process does not own the file, and the new group ID is not in the group ID of the process or in the supplementary group list of the process, and `_POSIX_CHOWN_RESTRICTED` is not in effect, the calling process may assert the `PRIV_FILE_CHOWN` privilege.

**Trusted Solaris 8
4/01 Reference
Manual**

chgrp(1), chown(1), chmod(2)

attributes(5)

fchroot(2)

NAME	chroot, fchroot – change root directory										
SYNOPSIS	<pre>#include <unistd.h> int chroot(const char *path); int fchroot(int fildes);</pre>										
DESCRIPTION	<p>The <code>chroot()</code> and <code>fchroot()</code> functions cause a directory to become the root directory, the starting point for path searches for path names beginning with / (slash). The user's working directory is unaffected by the <code>chroot()</code> and <code>fchroot()</code> functions.</p> <p>The <i>path</i> argument points to a path name naming a directory. The <i>fildes</i> argument to <code>fchroot()</code> is the open file descriptor of the directory which is to become the root.</p> <p>The calling process must assert the <code>PRIV_PROC_CHROOT</code> privilege to use this system call. While it is always possible to change to the system root using the <code>fchroot()</code> function, it is not guaranteed to succeed in any other case, even should <i>fildes</i> be valid in all respects.</p> <p>The <code>"."</code> entry in the root directory is interpreted to mean the root directory itself. Therefore, <code>"."</code> cannot be used to access files outside the subtree rooted at the root directory. Instead, <code>fchroot()</code> can be used to reset the root to a directory that was opened before the root directory was changed.</p>										
RETURN VALUES	<p><code>chroot()</code> returns:</p> <p>0 On success.</p> <p>-1 On failure, and sets <code>errno</code> to indicate the error.</p>										
ERRORS	<p>The <code>chroot()</code> function will fail if:</p> <table> <tr> <td>EACCES</td><td>Search permission is denied for a component of the path prefix of <i>dirname</i>, or search permission is denied for the directory referred to by <i>dirname</i>. To override these restrictions, the calling process may assert one or both of these privileges: <code>PRIV_FILE_DAC_SEARCH</code> and <code>PRIV_FILE_MAC_SEARCH</code>.</td></tr> <tr> <td>EBADF</td><td>The descriptor is not valid.</td></tr> <tr> <td>EFAULT</td><td>The <i>path</i> argument points to an illegal address.</td></tr> <tr> <td>EINVAL</td><td>The <code>fchroot()</code> function attempted to change to a directory that is not the system root and external circumstances do not allow this.</td></tr> <tr> <td>EINTR</td><td>A signal was caught during the execution of the <code>chroot()</code> function.</td></tr> </table>	EACCES	Search permission is denied for a component of the path prefix of <i>dirname</i> , or search permission is denied for the directory referred to by <i>dirname</i> . To override these restrictions, the calling process may assert one or both of these privileges: <code>PRIV_FILE_DAC_SEARCH</code> and <code>PRIV_FILE_MAC_SEARCH</code> .	EBADF	The descriptor is not valid.	EFAULT	The <i>path</i> argument points to an illegal address.	EINVAL	The <code>fchroot()</code> function attempted to change to a directory that is not the system root and external circumstances do not allow this.	EINTR	A signal was caught during the execution of the <code>chroot()</code> function.
EACCES	Search permission is denied for a component of the path prefix of <i>dirname</i> , or search permission is denied for the directory referred to by <i>dirname</i> . To override these restrictions, the calling process may assert one or both of these privileges: <code>PRIV_FILE_DAC_SEARCH</code> and <code>PRIV_FILE_MAC_SEARCH</code> .										
EBADF	The descriptor is not valid.										
EFAULT	The <i>path</i> argument points to an illegal address.										
EINVAL	The <code>fchroot()</code> function attempted to change to a directory that is not the system root and external circumstances do not allow this.										
EINTR	A signal was caught during the execution of the <code>chroot()</code> function.										

fchroot(2)

	EIO	An I/O error occurred while reading from or writing to the file system.
	ELOOP	Too many symbolic links were encountered in translating <i>path</i> .
	ENAMETOOLONG	The length of the <i>path</i> argument exceeds <code>PATH_MAX</code> , or the length of a <i>path</i> component exceeds <code>NAME_MAX</code> while <code>_POSIX_NO_TRUNC</code> is in effect.
	ENOENT	The named directory does not exist or is a null pathname.
	ENOLINK	The <i>path</i> argument points to a remote machine and the link to that machine is no longer active.
	ENOTDIR	Any component of the path name is not a directory.
	EPERM	The calling process must assert the <code>PRIV_PROC_CHROOT</code> privilege to change the root directory.
SUMMARY OF TRUSTED SOLARIS CHANGES Trusted Solaris 8 4/01 Reference WARNINGS		Appropriate privilege is required to override access checks.
		The calling process must assert the <code>PRIV_PROC_CHROOT</code> privilege to change the root directory.
	chroot(1M)	
		The only use of <code>fchroot ()</code> that is appropriate is to change back to the system root.

fcntl(2)

NAME	fcntl – file control												
SYNOPSIS	<pre>#include <sys/types.h> #include <unistd.h> #include <fcntl.h> int fcntl(int <i>fildev</i>, int <i>cmd</i>, /* <i>arg</i> */ ...);</pre>												
DESCRIPTION	<p>The <code>fcntl()</code> function provides for control over open files. The <i>fildev</i> argument is an open file descriptor. [See <code>intro(2)</code>.]</p> <p>The <code>fcntl()</code> function may take a third argument, <i>arg</i>, whose data type, value and use depend upon the value of <i>cmd</i>. The <i>cmd</i> argument specifies the operation to be performed by <code>fcntl()</code>.</p> <p>The available values for <i>cmd</i> are defined in the header <code><fcntl.h></code>, which include:</p> <table> <tr> <td><code>F_DUPFD</code></td><td>Return a new file descriptor which is the lowest numbered available (that is, not already open) file descriptor greater than or equal to the third argument, <i>arg</i>, taken as an integer of type <code>int</code>. The new file descriptor refers to the same open file description as the original file descriptor, and shares any locks. The <code>FD_CLOEXEC</code> flag associated with the new file descriptor is cleared to keep the file open across calls to one of the <code>exec(2)</code> functions.</td></tr> <tr> <td><code>F_DUP2FD</code></td><td>Similar to <code>F_DUPFD</code>, but always returns <i>arg</i>. <code>F_DUP2FD</code> closes <i>arg</i> if it is open and not equal to <i>fildev</i>. <code>F_DUP2FD</code> is equivalent to <code>dup2(fildev, arg)</code>.</td></tr> <tr> <td><code>F_GETFD</code></td><td>Get the file descriptor flags defined in <code><fcntl.h></code> that are associated with the file descriptor <i>fildev</i>. File descriptor flags are associated with a single file descriptor and do not affect other file descriptors that refer to the same file.</td></tr> <tr> <td><code>F_SETFD</code></td><td>Set the file descriptor flags defined in <code><fcntl.h></code>, that are associated with <i>fildev</i>, to the third argument, <i>arg</i>, taken as type <code>int</code>. If the <code>FD_CLOEXEC</code> flag in the third argument is 0, the file will remain open across the <code>exec()</code> functions; otherwise the file will be closed upon successful execution of one of the <code>exec()</code> functions.</td></tr> <tr> <td><code>F_GETFL</code></td><td>Get the file status flags and file access modes, defined in <code><fcntl.h></code>, for the file description associated with <i>fildev</i>. The file access modes can be extracted from the return value using the mask <code>O_ACCMODE</code>, which is defined in <code><fcntl.h></code>. File status flags and file access modes are associated with the file description and do not affect other file descriptors that refer to the same file with different open file descriptions.</td></tr> <tr> <td><code>F_SETFL</code></td><td>Set the file status flags, defined in <code><fcntl.h></code>, for the file description associated with <i>fildev</i> from the corresponding bits in the third argument, <i>arg</i>, taken as type <code>int</code>. Bits corresponding to</td></tr> </table>	<code>F_DUPFD</code>	Return a new file descriptor which is the lowest numbered available (that is, not already open) file descriptor greater than or equal to the third argument, <i>arg</i> , taken as an integer of type <code>int</code> . The new file descriptor refers to the same open file description as the original file descriptor, and shares any locks. The <code>FD_CLOEXEC</code> flag associated with the new file descriptor is cleared to keep the file open across calls to one of the <code>exec(2)</code> functions.	<code>F_DUP2FD</code>	Similar to <code>F_DUPFD</code> , but always returns <i>arg</i> . <code>F_DUP2FD</code> closes <i>arg</i> if it is open and not equal to <i>fildev</i> . <code>F_DUP2FD</code> is equivalent to <code>dup2(fildev, arg)</code> .	<code>F_GETFD</code>	Get the file descriptor flags defined in <code><fcntl.h></code> that are associated with the file descriptor <i>fildev</i> . File descriptor flags are associated with a single file descriptor and do not affect other file descriptors that refer to the same file.	<code>F_SETFD</code>	Set the file descriptor flags defined in <code><fcntl.h></code> , that are associated with <i>fildev</i> , to the third argument, <i>arg</i> , taken as type <code>int</code> . If the <code>FD_CLOEXEC</code> flag in the third argument is 0, the file will remain open across the <code>exec()</code> functions; otherwise the file will be closed upon successful execution of one of the <code>exec()</code> functions.	<code>F_GETFL</code>	Get the file status flags and file access modes, defined in <code><fcntl.h></code> , for the file description associated with <i>fildev</i> . The file access modes can be extracted from the return value using the mask <code>O_ACCMODE</code> , which is defined in <code><fcntl.h></code> . File status flags and file access modes are associated with the file description and do not affect other file descriptors that refer to the same file with different open file descriptions.	<code>F_SETFL</code>	Set the file status flags, defined in <code><fcntl.h></code> , for the file description associated with <i>fildev</i> from the corresponding bits in the third argument, <i>arg</i> , taken as type <code>int</code> . Bits corresponding to
<code>F_DUPFD</code>	Return a new file descriptor which is the lowest numbered available (that is, not already open) file descriptor greater than or equal to the third argument, <i>arg</i> , taken as an integer of type <code>int</code> . The new file descriptor refers to the same open file description as the original file descriptor, and shares any locks. The <code>FD_CLOEXEC</code> flag associated with the new file descriptor is cleared to keep the file open across calls to one of the <code>exec(2)</code> functions.												
<code>F_DUP2FD</code>	Similar to <code>F_DUPFD</code> , but always returns <i>arg</i> . <code>F_DUP2FD</code> closes <i>arg</i> if it is open and not equal to <i>fildev</i> . <code>F_DUP2FD</code> is equivalent to <code>dup2(fildev, arg)</code> .												
<code>F_GETFD</code>	Get the file descriptor flags defined in <code><fcntl.h></code> that are associated with the file descriptor <i>fildev</i> . File descriptor flags are associated with a single file descriptor and do not affect other file descriptors that refer to the same file.												
<code>F_SETFD</code>	Set the file descriptor flags defined in <code><fcntl.h></code> , that are associated with <i>fildev</i> , to the third argument, <i>arg</i> , taken as type <code>int</code> . If the <code>FD_CLOEXEC</code> flag in the third argument is 0, the file will remain open across the <code>exec()</code> functions; otherwise the file will be closed upon successful execution of one of the <code>exec()</code> functions.												
<code>F_GETFL</code>	Get the file status flags and file access modes, defined in <code><fcntl.h></code> , for the file description associated with <i>fildev</i> . The file access modes can be extracted from the return value using the mask <code>O_ACCMODE</code> , which is defined in <code><fcntl.h></code> . File status flags and file access modes are associated with the file description and do not affect other file descriptors that refer to the same file with different open file descriptions.												
<code>F_SETFL</code>	Set the file status flags, defined in <code><fcntl.h></code> , for the file description associated with <i>fildev</i> from the corresponding bits in the third argument, <i>arg</i> , taken as type <code>int</code> . Bits corresponding to												

	the file access mode and the <i>oflag</i> values that are set in <i>arg</i> are ignored. If any bits in <i>arg</i> other than those mentioned here are changed by the application, the result is unspecified.
F_GETOWN	If <i>fildev</i> refers to a socket, get the process or process group ID specified to receive SIGURG signals when out-of-band data is available. Positive values indicate a process ID; negative values, other than -1, indicate a process group ID. If <i>fildev</i> does not refer to a socket, the results are unspecified.
F_SETOWN	If <i>fildev</i> refers to a socket, set the process or process group ID specified to receive SIGURG signals when out-of-band data is available, using the value of the third argument, <i>arg</i> , taken as type <code>int</code> . Positive values indicate a process ID; negative values, other than -1, indicate a process group ID. If <i>fildev</i> does not refer to a socket, the results are unspecified.
F_FREESP	Free storage space associated with a section of the ordinary file <i>fildev</i> . The section is specified by a variable of data type <code>struct flock</code> pointed to by <i>arg</i> . The data type <code>struct flock</code> is defined in the <code><fcntl.h></code> header (see <code>fcntl(5)</code>) and is described below. Note that all file systems might not support all possible variations of <code>F_FREESP</code> arguments. In particular, many file systems allow space to be freed only at the end of a file.
The following commands are available for advisory record locking. Record locking is supported for regular files, and may be supported for other files.	
F_GETLK	Get the first lock which blocks the lock description pointed to by the third argument, <i>arg</i> , taken as a pointer to type <code>struct flock</code> , defined in <code><fcntl.h></code> . The information retrieved overwrites the information passed to <code>fcntl()</code> in the structure <code>flock</code> . If no lock is found that would prevent this lock from being created, then the structure will be left unchanged except for the lock type which will be set to <code>F_UNLCK</code> .
F_GETLK64	Equivalent to <code>F_GETLK</code> , but takes a <code>struct flock64</code> argument rather than a <code>struct flock</code> argument.
F_SETLK	Set or clear a file segment lock according to the lock description pointed to by the third argument, <i>arg</i> , taken as a pointer to type <code>struct flock</code> , defined in <code><fcntl.h></code> . <code>F_SETLK</code> is used to establish shared (or read) locks (<code>F_RDLCK</code>) or exclusive (or write) locks (<code>F_WRLCK</code>), as well as to remove either type of lock (<code>F_UNLCK</code>). <code>F_RDLCK</code> , <code>F_WRLCK</code> and <code>F_UNLCK</code> are defined in <code><fcntl.h></code> . If a shared or exclusive lock cannot be set, <code>fcntl()</code> will return immediately with a return value of -1.
F_SETLK64	Equivalent to <code>F_SETLK</code> , but takes a <code>struct flock64</code> argument rather than a <code>struct flock</code> argument.

fcntl(2)

<code>F_SETLKW</code>	This command is the same as <code>F_SETLK</code> except that if a shared or exclusive lock is blocked by other locks, the process will wait until the request can be satisfied. If a signal that is to be caught is received while <code>fcntl()</code> is waiting for a region, <code>fcntl()</code> will be interrupted. Upon return from the process' signal handler, <code>fcntl()</code> will return <code>-1</code> with <code>errno</code> set to <code>EINTR</code> , and the lock operation will not be done.
<code>F_SETLKW64</code>	Equivalent to <code>F_SETLKW</code> , but takes a <code>struct flock64</code> argument rather than a <code>struct flock</code> argument.

When a shared lock is set on a segment of a file, other processes will be able to set shared locks on that segment or a portion of it. A shared lock prevents any other process from setting an exclusive lock on any portion of the protected area. A request for a shared lock will fail if the file descriptor was not opened with read access.

An exclusive lock will prevent any other process from setting a shared lock or an exclusive lock on any portion of the protected area. A request for an exclusive lock will fail if the file descriptor was not opened with write access.

The `flock` structure contains at least the following elements:

```
short  l_type;      /* lock operation type */
short  l_whence;    /* lock base indicator */
off_t  l_start;     /* starting offset from base */
off_t  l_len;       /* lock length; l_len == 0 means
                    until end of file */
long   l_sysid;     /* system ID running process holding lock */
pid_t  l_pid;       /* process ID of process holding lock */
```

The value of `l_whence` is `SEEK_SET`, `SEEK_CUR`, or `SEEK_END`, to indicate that the relative offset `l_start` bytes will be measured from the start of the file, current position or end of the file, respectively. The value of `l_len` is the number of consecutive bytes to be locked. The value of `l_len` may be negative (where the definition of `off_t` permits negative values of `l_len`). After a successful `F_GETLK` or `F_GETLK64` request, that is, one in which a lock was found, the value of `l_whence` will be `SEEK_SET`.

The `l_pid` and `l_sysid` fields are used only with `F_GETLK` or `F_GETLK64` to return the process ID of the process holding a blocking lock and to indicate which system is running that process.

If `l_len` is positive, the area affected starts at `l_start` and ends at `l_start + l_len - 1`. If `l_len` is negative, the area affected starts at `l_start + l_len` and ends at `l_start - 1`. Locks may start and extend beyond the current end of a file, but must not be negative relative to the beginning of the file. A lock will be set to extend to the largest possible value of the file offset for that file by setting `l_len` to 0. If such a lock also has `l_start` set to 0 and `l_whence` is set to `SEEK_SET`, the whole file will be locked.

If a process has an existing lock in which `l_len` is 0 and which includes the last byte of the requested segment, and an unlock (`F_UNLCK`) request is made in which `l_len` is non-zero and the offset of the last byte of the requested segment is the maximum

value for an object of type `off_t`, then the `F_UNLCK` request will be treated as a request to unlock from the start of the requested segment with an `l_len` equal to 0. Otherwise, the request will attempt to unlock only the requested segment.

There will be at most one type of lock set for each byte in the file. Before a successful return from an `F_SETLK`, `F_SETLK64`, `F_SETLKW`, or `F_SETLKW64` request when the calling process has previously existing locks on bytes in the region specified by the request, the previous lock type for each byte in the specified region will be replaced by the new lock type. As specified above under the descriptions of shared locks and exclusive locks, an `F_SETLK`, `F_SETLK64`, `F_SETLKW`, or `F_SETLKW64` request will (respectively) fail or block when another process has existing locks on bytes in the specified region and the type of any of those locks conflicts with the type specified in the request.

All locks associated with a file for a given process are removed when a file descriptor for that file is closed by that process or the process holding that file descriptor terminates. Locks are not inherited by a child process created using `fork(2)`.

When mandatory file and record locking is active on a file [see `chmod(2)`, `creat(2)`, `open(2)`, `read(2)` and `write(2)`], functions issued on the file will be affected by the record locks in effect. When mandatory file and record locking is active on a file, it cannot be memory mapped.

A potential for deadlock occurs if a process controlling a locked region is put to sleep by attempting to lock another process' locked region. If the system detects that sleeping until a locked region is unlocked would cause a deadlock, `fcntl()` will fail with an `EDEADLK` error.

The following values for `cmd` are used for file share reservations. A share reservation is placed on an entire file to allow cooperating processes to control access to the file.

<code>F_SHARE</code>	Sets a share reservation on a file with the specified access mode and designates which types of access to deny.
<code>F_UNSHARE</code>	Remove an existing share reservation.

File share reservations are an advisory form of access control among cooperating processes, on both local and remote machines. They are most often used by DOS or Windows emulators and DOS based NFS clients. However, native UNIX versions of DOS or Windows applications may also choose to use this form of access control.

A share reservation is described by an `fshare` structure defined in `<sys/fcntl.h>`, which is included in `<fcntl.h>` as follows:

```
typedef struct fshare {
    short    f_access;
    short    f_deny;
    long     f_id;
} fshare_t;
```

fcntl(2)

A share reservation specifies the type of access, *f_access*, to be requested on the open file descriptor. If access is granted, it further specifies what type of access to deny other processes, *f_deny*. A single process on the same file may hold multiple non-conflicting reservations by specifying an identifier, *f_id*, unique to the process, with each request.

An `F_UNSHARE` request releases the reservation with the specified *f_id*. The *f_access* and *f_deny* fields are ignored.

Valid *f_access* values are:

<code>F_RDACC</code>	Set a file share reservation for read-only access.
<code>F_WRACC</code>	Set a file share reservation for write-only access.
<code>F_RWACC</code>	Set a file share reservation for read and write access.

Valid *f_deny* values are:

<code>F_COMPAT</code>	Set a file share reservation to compatibility mode.
<code>F_RDDNY</code>	Set a file share reservation to deny read access to other processes.
<code>F_WRDNY</code>	Set a file share reservation to deny write access to other processes.
<code>F_RWDNY</code>	Set a file share reservation to deny read and write access to other processes.
<code>F_NODNY</code>	Do not deny read or write access to any other process.

RETURN VALUES

Upon successful completion, the value returned depends on *cmd* as follows:

<code>F_DUPFD</code>	A new file descriptor.
<code>F_GETFD</code>	Value of flags defined in <code><fcntl.h></code> . The return value will not be negative.
<code>F_SETFD</code>	Value other than <code>-1</code> .
<code>F_GETFL</code>	Value of file status flags and access modes. The return value will not be negative.
<code>F_SETFL</code>	Value other than <code>-1</code> .
<code>F_GETOWN</code>	Value of the socket owner process or process group; this will not be <code>-1</code> .
<code>F_SETOWN</code>	Value other than <code>-1</code> .
<code>F_FREESP</code>	Value of <code>0</code> .
<code>F_GETLK</code>	Value other than <code>-1</code> .
<code>F_GETLK64</code>	Value other than <code>-1</code> .
<code>F_SETLK</code>	Value other than <code>-1</code> .

F_SETLK64	Value other than -1.
F_SETLKW	Value other than -1.
F_SETLKW64	Value other than -1.
F_SHARE	Value other than -1.
F_UNSHARE	Value other than -1.

Otherwise, -1 is returned and `errno` is set to indicate the error.

ERRORS

The `fcntl()` function will fail if:

EAGAIN	<p>The <i>cmd</i> argument is <code>F_SETLK</code> or <code>F_SETLK64</code>, the type of lock (<i>l_type</i>) is a shared (<code>F_RDLCK</code>) or exclusive (<code>F_WRLCK</code>) lock, and the segment of a file to be locked is already exclusive-locked by another process; or the type is an exclusive lock and some portion of the segment of a file to be locked is already shared-locked or exclusive-locked by another process.</p> <p>The <i>cmd</i> argument is <code>F_FREESP</code>, the file exists, mandatory file/record locking is set, and there are outstanding record locks on the file; or the <i>cmd</i> argument is <code>F_SETLK</code>, <code>F_SETLK64</code>, <code>F_SETLKW</code>, or <code>F_SETLKW64</code>, mandatory file/record locking is set, and the file is currently being mapped to virtual memory using <code>mmap(2)</code>.</p> <p>The <i>cmd</i> argument is <code>F_SHARE</code> and <i>f_access</i> conflicts with an existing <i>f_deny</i> share reservation.</p>
EBADF	<p>The <i>fildev</i> argument is not a valid open file descriptor; or the <i>cmd</i> argument is <code>F_SETLK</code>, <code>F_SETLK64</code>, <code>F_SETLKW</code>, or <code>F_SETLKW64</code>, the type of lock, <i>l_type</i>, is a shared lock (<code>F_RDLCK</code>), and <i>fildev</i> is not a valid file descriptor open for reading; or the type of lock <i>l_type</i> is an exclusive lock (<code>F_WRLCK</code>) and <i>fildev</i> is not a valid file descriptor open for writing.</p> <p>The <i>cmd</i> argument is <code>F_FREESP</code> and <i>fildev</i> is not a valid file descriptor open for writing.</p> <p>The <i>cmd</i> argument is <code>F_DUP2FD</code>, and <i>arg</i> is negative or is not less than the current resource limit for <code>RLIMIT_NOFILE</code>.</p> <p>The <i>cmd</i> argument is <code>F_SHARE</code>, the <i>f_access</i> share reservation is for write access, and <i>fildev</i> is not a valid file descriptor open for writing.</p> <p>The <i>cmd</i> argument is <code>F_SHARE</code>, the <i>f_access</i> share reservation is for read access, and <i>fildev</i> is not a valid file descriptor open for reading.</p>

fcntl(2)

EFAULT	<p>The <i>cmd</i> argument is F_GETLK, F_GETLK64, F_SETLK, F_SETLK64, F_SETLKW, F_SETLKW64, or F_FREESP and the <i>arg</i> argument points to an illegal address.</p> <p>The <i>cmd</i> argument is F_SHARE or F_UNSHARE and <i>arg</i> points to an illegal address.</p>
EINTR	The <i>cmd</i> argument is F_SETLKW or F_SETLKW64 and the function was interrupted by a signal.
EINVAL	<p>The <i>cmd</i> argument is invalid; or the <i>cmd</i> argument is F_DUPFD and <i>arg</i> is negative or greater than or equal to OPEN_MAX; or the <i>cmd</i> argument is F_GETLK, F_GETLK64, F_SETLK, F_SETLK64, F_SETLKW, or F_SETLKW64 and the data pointed to by <i>arg</i> is not valid; or <i>fildev</i> refers to a file that does not support locking.</p> <p>The <i>cmd</i> argument is F_UNSHARE and a reservation with this <i>f_id</i> for this process does not exist.</p>
EIO	An I/O error occurred while reading from or writing to the file system.
EMFILE	The <i>cmd</i> argument is F_DUPFD and either OPEN_MAX file descriptors are currently open in the calling process, or no file descriptors greater than or equal to <i>arg</i> are available.
ENOLCK	The <i>cmd</i> argument is F_SETLK, F_SETLK64, F_SETLKW, or F_SETLKW64 and satisfying the lock or unlock request would result in the number of locked regions in the system exceeding a system-imposed limit.
ENOLINK	Either the <i>fildev</i> argument is on a remote machine and the link to that machine is no longer active; or the <i>cmd</i> argument is F_FREESP, the file is on a remote machine, and the link to that machine is no longer active.
EOVERFLOW	<p>One of the values to be returned cannot be represented correctly.</p> <p>The <i>cmd</i> argument is F_GETLK, F_SETLK, or F_SETLKW and the smallest or, if <i>l_len</i> is non-zero, the largest, offset of any byte in the requested segment cannot be represented correctly in an object of type <code>off_t</code>.</p> <p>The <i>cmd</i> argument is F_GETLK64, F_SETLK64, or F_SETLKW64 and the smallest or, if <i>l_len</i> is non-zero, the largest, offset of any byte in the requested segment cannot be represented correctly in an object of type <code>off64_t</code>.</p>
The <code>fcntl()</code> function may fail if:	

EAGAIN	The <i>cmd</i> argument is F_SETLK, F_SETLK64, F_SETLKW, or F_SETLKW64, and the file is currently being mapped to virtual memory using <code>mmap(2)</code> .
EDEADLK	<p>The <i>cmd</i> argument is F_SETLKW or F_SETLKW64, the lock is blocked by some lock from another process and putting the calling process to sleep, waiting for that lock to become free would cause a deadlock.</p> <p>The <i>cmd</i> argument is F_FREESP, mandatory record locking is enabled, O_NDELAY and O_NONBLOCK are clear and a deadlock condition was detected.</p>

ATTRIBUTES See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Async-Signal Safe

SUMMARY OF TRUSTED SOLARIS CHANGES

For the F_GETLK operation, when the requested lock is blocked, mandatory access checks are required to ensure that the sensitivity label of the calling process that is requesting the lock dominates the sensitivity label of the process holding the blocking lock. This is done to prevent the transmission of lock information from a process holding the blocking lock which dominates the sensitivity label of the calling process making the F_GETLK request. If the calling process fails this MAC check, then fixed results are returned indicating that the entire file is locked, and with zeroes for the process ID and system ID. The calling process may assert the PRIV_FILE_LOCK privilege to bypass this check.

Trusted Solaris 8 4/01 Reference Manual NOTES

`lockd(1M)`, `chmod(2)`, `creat(2)`, `exec(2)`, `fork(2)`, `open(2)`, `read(2)`, `write(2)`

`close(2)`, `dup(2)`, `pipe(2)`, `attributes(5)`, `fcntl(5)`

In the past, the variable `errno` was set to `EACCES` rather than `EAGAIN` when a section of a file is already locked by another process. Therefore, portable application programs should expect and test for either value.

Advisory locks allow cooperating processes to perform consistent operations on files, but do not guarantee exclusive access. Files can be accessed without advisory locks, but inconsistencies may result. The network share locking protocol does not support the `f_deny` value of `F_COMPAT`. For network file systems, if `f_access` is `F_RDACC`, `f_deny` is mapped to `F_RDDNY`. Otherwise, it is mapped to `F_RWDNY`.

To prevent possible file corruption, the system may reject `mmap()` requests for advisory locked files, or it may reject advisory locking requests for mapped files. Applications that require a file be both locked and mapped should lock the entire file (`l_start` and `l_len` both set to 0). If a file is mapped, the system may reject an unlock request, resulting in a lock that does not cover the entire file.

fcntl(2)

If the file server crashes and has to be rebooted, the lock manager (see `lockd(1M)`) attempts to recover all locks that were associated with that server. If a lock cannot be reclaimed, the process that held the lock is issued a `SIGLOST` signal.

`read(2)` and `write(2)` system calls on files are affected by mandatory file and record locks. [See `chmod(2)`.]

NAME	getcmwfsrange, fgetcmwfsrange – Get file system sensitivity label range														
SYNOPSIS	<pre>cc [flags...] file... -ltsol #include <tsol/label.h> int getcmwfsrange(char *path, brange_t *range_p); int fgetcmwfsrange(int fd, brange_t *range_p);</pre>														
DESCRIPTION	<p>getcmwfsrange() returns the sensitivity label range of a mounted file system. <i>path</i> is the path name of any file within the mounted filesystem. <i>range_p</i> is a pointer to a sensitivity label range structure defined as follows:</p> <pre>struct binary_level_range { blevel_t lower_bound; blevel_t upper_bound; }; typedef struct binary_level_range brange_t; /* Level Range */</pre> <p>fgetcmwfsrange() returns the same information about an open file referred to by descriptor <i>fd</i>.</p>														
RETURN VALUES	<p>getcmwfsrange() and fgetcmwfsrange() return:</p> <p>0 On success.</p> <p>-1 On failure, and set <i>errno</i> to indicate the error.</p>														
ERRORS	<p>getcmwfsrange() fails if one or more of the following are true:</p> <table> <tr> <td>EACCES</td><td>Search permission is denied for a component of the path prefix of <i>path</i>. To override this restriction, the calling process may assert the PRIV_FILE_DAC_SEARCH privilege and/or the PRIV_FILE_MAC_SEARCH privilege.</td></tr> <tr> <td>EFAULT</td><td><i>range_p</i> or <i>path</i> points to an invalid address.</td></tr> <tr> <td>EIO</td><td>An I/O error occurred while reading from or writing to the file system.</td></tr> <tr> <td>ELOOP</td><td>Too many symbolic links were encountered in translating <i>path</i>.</td></tr> <tr> <td>ENAMETOOLONG</td><td>The length of the path argument exceeds PATH_MAX. A pathname component is longer than NAME_MAX (see sysconf(3C)) while _POSIX_NO_TRUNC is in effect (see pathconf(2V)).</td></tr> <tr> <td>ENOENT</td><td>The file referred to by <i>path</i> does not exist.</td></tr> <tr> <td>ENOTDIR</td><td>A component of the path prefix of <i>path</i> is not a directory.</td></tr> </table> <p>fgetcmwfsrange() fails if one or more of the following are true:</p>	EACCES	Search permission is denied for a component of the path prefix of <i>path</i> . To override this restriction, the calling process may assert the PRIV_FILE_DAC_SEARCH privilege and/or the PRIV_FILE_MAC_SEARCH privilege.	EFAULT	<i>range_p</i> or <i>path</i> points to an invalid address.	EIO	An I/O error occurred while reading from or writing to the file system.	ELOOP	Too many symbolic links were encountered in translating <i>path</i> .	ENAMETOOLONG	The length of the path argument exceeds PATH_MAX. A pathname component is longer than NAME_MAX (see sysconf(3C)) while _POSIX_NO_TRUNC is in effect (see pathconf(2V)).	ENOENT	The file referred to by <i>path</i> does not exist.	ENOTDIR	A component of the path prefix of <i>path</i> is not a directory.
EACCES	Search permission is denied for a component of the path prefix of <i>path</i> . To override this restriction, the calling process may assert the PRIV_FILE_DAC_SEARCH privilege and/or the PRIV_FILE_MAC_SEARCH privilege.														
EFAULT	<i>range_p</i> or <i>path</i> points to an invalid address.														
EIO	An I/O error occurred while reading from or writing to the file system.														
ELOOP	Too many symbolic links were encountered in translating <i>path</i> .														
ENAMETOOLONG	The length of the path argument exceeds PATH_MAX. A pathname component is longer than NAME_MAX (see sysconf(3C)) while _POSIX_NO_TRUNC is in effect (see pathconf(2V)).														
ENOENT	The file referred to by <i>path</i> does not exist.														
ENOTDIR	A component of the path prefix of <i>path</i> is not a directory.														

fgetcmwfsrange(2)

EBADF	<i>fd</i> is not a valid open file descriptor.
EFAULT	<i>range_p</i> points to an invalid address.
EINVAL	<i>fd</i> refers to a socket, not a file.
EIO	An I/O error occurred while reading from the file system.

Trusted Solaris 8
4/01 Reference
Manual
SunOS 5.8
Reference Manual

pathconf(2)
sysconf(3C)

NAME	getcmwlabel, lgetcmwlabel, fgetcmwlabel – get file CMW label
SYNOPSIS	<pre>cc [flags...] file ... -ltsol [library...] #include <tsol/label.h> int getcmwlabel(char *path, bclabel_t *label_p); int lgetcmwlabel(char *path, bclabel_t *label_p); int fgetcmwlabel(int fd, bclabel_t *label_p);</pre>
DESCRIPTION	<p>getcmwlabel() obtains the CMW label of the file named by <i>path</i>. Mandatory read access to the final component of <i>path</i> is required or the calling process must have PRIV_FILE_MAC_READ in its set of effective privileges. Discretionary read, write or execute permission to the final component of <i>path</i> is not required, but all directories in the path prefix of <i>path</i> must be searchable.</p> <p>lgetcmwlabel() is like getcmwlabel() except in the case where the final component of <i>path</i> is a symbolic link, in which case lgetcmwlabel() returns the CMW label of the link, while getcmwlabel() returns the CMW label of the file to which the link refers.</p> <p>fgetcmwlabel() obtains the CMW label of an open file referred to by the argument descriptor, such as would be obtained by an open(2) call. If the descriptor is only open for writing, then mandatory read access to the object is required or the calling process must have PRIV_FILE_MAC_READ in its set of effective privileges.</p> <p><i>label_p</i> is a pointer to an opaque CMW label structure.</p> <p>An exception to the access rules applies in the case of pty pseudo-terminals (/dev/ptyp* and /dev/ttyp*). Normally mandatory read access is required or the calling process must have PRIV_FILE_MAC_READ in its set of effective privileges. If the specified file is a pty device file and the calling process does not have mandatory read access or PRIV_FILE_MAC_READ is not in its set of effective privileges, each function returns success and sets <i>label_p</i> to ADMIN_LOW.</p>
RETURN VALUES	<p>getcmwlabel(), lgetcmwlabel() and fgetcmwlabel() return:</p> <p>0 On success.</p> <p>-1 On failure, and set errno to indicate the error.</p>
ERRORS	<p>getcmwlabel() and lgetcmwlabel() fail if one or more of the following are true:</p> <p>EACCES Search permission is denied for a component of the path prefix of <i>path</i>. To override this restriction, the calling process may assert the PRIV_FILE_DAC_SEARCH privilege and/or the PRIV_FILE_MAC_SEARCH privilege.</p> <p> The calling process does not have mandatory read access to <i>path</i> because the sensitivity label of the calling process does not</p>

fgetcmwlabel(2)

	dominate the sensitivity label of the final component of <i>path</i> and the calling process does not have PRIV_FILE_MAC_READ in its set of effective privileges.
EFAULT	<i>label_p</i> or <i>path</i> points to an invalid address.
EIO	An I/O error occurred while reading from or writing to the file system.
ELOOP	Too many symbolic links were encountered in translating <i>path</i> .
ENAMETOOLONG	The length of the path argument exceeds PATH_MAX. A pathname component is longer than NAME_MAX while _POSIX_NO_TRUNC is in effect (see pathconf(2)).
ENOENT	The file referred to by <i>path</i> does not exist.
ENOTDIR	A component of the path prefix of <i>path</i> is not a directory.
EPERM	The calling process does not have mandatory read access to <i>path</i> because the sensitivity label of <i>path</i> is outside the calling process' clearance and the calling process does not have PRIV_FILE_MAC_READ in its set of effective privileges.
fgetcmwlabel() fails if one or more of the following are true:	
EACCES	The descriptor is only open for writing and the calling process does not have mandatory read access to the object referred to by the descriptor because the sensitivity label of the calling process does not dominate the sensitivity label of the object and the calling process does not have PRIV_FILE_MAC_READ in its set of effective privileges.
EBADF	<i>fd</i> is not a valid open file descriptor.
EFAULT	<i>label_p</i> points to an invalid address.
EIO	An I/O error occurred while reading from or writing to the file system.

pathconf(2), open(2), setcmwlabel(2)

NAME	getfattrflag, fsetfattrflag, fgetfattrflag, setfattrflag, mldgetfattrflag, mldsetfattrflag – set/get the security attribute flags of a file
SYNOPSIS	<pre>cc [flags...] file... -ltsol #include <tsol/secflgs.h> int getfattrflag(const char *path, secflgs_t *flags); int setfattrflag(const char *path, secflgs_t which, secflgs_t flags); int fgetfattrflag(int fildes, secflgs_t *flags); int fsetfattrflag(int fildes, secflgs_t which, secflgs_t flags); int mldgetfattrflag(const char *path, secflgs_t *flags); int mldsetfattrflag(const char *path, secflgs_t which, secflgs_t flags);</pre>
DESCRIPTION	<p>setfattrflag(), fsetfattrflag(), and mldsetfattrflag() set the security flags of the file whose name is given by <i>path</i> or referred to by the open file descriptor <i>fildes</i>. The bit pattern contained in <i>which</i> is used to indicate which flags are being affected. The corresponding bits in <i>flags</i> are set to 1 or 0 to indicate whether the affected flags are being set or unset respectively.</p> <p>getfattrflag(), fgetfattrflag(), and mldgetfattrflag() get the security flags of the file whose name is given by <i>path</i> or referred to by the open file descriptor <i>fildes</i> and store it in the location pointed to by <i>flags</i>.</p> <p>Attribute bits are interpreted as follows:</p> <p>FAF_MLD Directory has MLD semantics.</p> <p>FAF_PUBLIC Filesystem object is a public object.</p> <p>FAF_SLD Directory is an SLD.</p> <p>Attribute flags are constructed by OR'ing the attribute flag bits.</p> <p>FAF_MLD is the only flag that may be modified without privilege if the directory is empty, the effective user ID of the process matches the directory owner, and the process has mandatory as well as discretionary write access. The FAF_MLD flag, once set, cannot be unset. Additionally, the FAF_MLD flag may only be set via the mldsetfattrflag interface. The FAF_PUBLIC flag can only be read or modified by a process possessing the PRIV_FILE_AUDIT privilege. A process attempting to read the FAF_PUBLIC flag without the PRIV_FILE_AUDIT privilege in effect will not fail. However the value of FAF_PUBLIC will be returned as unset. The FAF_SLD flag can never be set. The ability to read any flag is dependant upon the process having mandatory and discretionary read access to the file. The ability to set any flag is dependant upon the process having mandatory and discretionary write access to the file.</p>

fgetfattrflag(2)

If *path* is a symbolic link, the target's attribute flags are affected rather than the link's. If *path* is a multilevel directory, `getfattrflag()` and `setfattrflag()` will affect the underlying single-level directory beneath (unless *path* is adorned). `mltgetfattrflag()` and `mltsetfattrflag()` do not translate multi-level directories to underlying single-level directories. `fgetfattrflag()` and `fsetsattrflag()` affect only the file referred to by *fildev*.

RETURN VALUES

These functions return:

- 0 On success.
- 1 On failure, and set `errno` to indicate the error.

ERRORS

`getfattrflag()` and `mltgetfattrflag()` will fail if one or more of the following are true:

- EACCES Search permission is denied on a component of the path prefix of *path*. To override this restriction, the calling process may assert the `PRIV_FILE_DAC_SEARCH` privilege and/or the `PRIV_FILE_MAC_SEARCH` privilege.
 - EACCES Read permission is denied the final component of *path*. To override this restriction, the calling process may assert the `PRIV_FILE_MAC_READ` privilege.
 - EFAULT *path* points to an illegal address.
 - EINTR A signal was caught during execution of the function.
 - EIO An I/O error occurred while reading from the file system.
 - ELOOP Too many symbolic links were encountered in translating *path*.
 - EMULTIHOP Components of *path* require hopping to multiple remote machines and file system type does not allow it.
 - ENAMETOOLONG The length of the *path* argument exceeds `PATH_MAX`, or the length of a *path* component exceeds `NAME_MAX` while `_POSIX_NO_TRUNC` is in effect.
 - ENOENT Either a component of the path prefix, or the file referred to by *path* does not exist or is a null pathname.
 - ENOLINK *fildev* points to a remote machine and the link to that machine is no longer active.
 - ENOTDIR A component of the prefix of *path* is not a directory.
- `fgetfattrflag()` fails and the file mode is unchanged if:
- EACCES Read permission is denied on *fildev*. To override this restriction, the calling process may assert the `PRIV_FILE_MAC_READ` privilege.
 - EBADF *fildev* is not an open file descriptor.

EIO	An I/O error occurred while reading from the file system.
EINTR	A signal was caught during execution of the <code>fgetfattrflag()</code> function.
<code>setfattrflag()</code> and <code>lchsetfattrflag()</code> will fail and the file mode is unchanged if one or more of the following are true:	
EACCES	Search permission is denied on a component of the path prefix of <i>path</i> . To override this restriction, the calling process may assert the <code>PRIV_FILE_DAC_SEARCH</code> privilege and/or the <code>PRIV_FILE_MAC_SEARCH</code> privilege.
EACCES	Write permission is denied <i>path</i> . To override this restriction, the calling process may assert the <code>PRIV_FILE_MAC_WRITE</code> privilege.
EACCES	The calling process does not own <i>files</i> . To override this restriction, the calling process may assert the <code>PRIV_FILE_OWNER</code> privilege.
EFAULT	<i>path</i> points to an illegal address.
EINTR	A signal was caught during execution of the function.
EINVAL	<i>path</i> is not a valid pathname. When setting <code>FAF_MLD</code> , <i>path</i> must refer to an empty directory.
EIO	An I/O error occurred while writing to the file system.
ELOOP	Too many symbolic links were encountered in translating <i>path</i> .
EMULTIHOP	Components of <i>path</i> require hopping to multiple remote machines and filesystem type does not allow it.
ENAMETOOLONG	The length of the <i>path</i> argument exceeds <code>PATH_MAX</code> , or the length of a <i>path</i> component exceeds <code>NAME_MAX</code> while <code>_POSIX_NO_TRUNC</code> is in effect.
ENOENT	Either a component of the path prefix, or the file referred to by <i>path</i> does not exist or is a null pathname.
ENOLINK	<i>path</i> points to a remote machine and the link to that machine is no longer active.
ENOTDIR	A component of the prefix of <i>path</i> is not a directory.
EPERM	The effective user ID does not match the owner of the file and the process does not possess the privilege <code>PRIV_FILE_OWNER</code> .
EPERM	The process does not possess the privilege <code>PRIV_FILE_AUDIT</code> and is attempting to set the <code>FAF_PUBLIC</code> flag.
EROFS	The file referred to by <i>path</i> resides on a read-only file system.
<code>fsetfattrflag()</code> fails and the file mode is unchanged if:	

fgetfattrflag(2)

EACCES	The calling process does not own <i>fil</i> <i>des</i> . To override this restriction, the calling process may assert the PRIV_FILE_OWNER privilege.
EACCES	Write access is denied on <i>fil</i> <i>des</i> . To override this restriction, the calling process may assert the PRIV_FILE_MAC_WRITE privilege.
EINVAL	<i>fil</i> <i>des</i> is not a valid pathname. When setting FAF_MLD, <i>fil</i> <i>des</i> must refer to an empty directory.
EBADF	<i>fil</i> <i>des</i> is not an open file descriptor.
EIO	An I/O error occurred while writing to the file system.
EINTR	A signal was caught during execution of the fsetfattrflag() function.
EPERM	The process does not possess the privilege PRIV_FILE_AUDIT and is attempting to set the FAF_PUBLIC flag.
EROFS	The file referred to by <i>fil</i> <i>des</i> resides on a read-only file system.

**Trusted Solaris 8
4/01 Reference
Manual**

setfattrflag(1), getfattrflag(1)

Trusted Solaris Developer's Guide

NAME	getfpriv, fgetfpriv, setfpriv, fsetfpriv – return or set a privilege set associated with a file										
SYNOPSIS	<pre>cc [flags...] file... -ltsol int getfpriv(char *path, priv_ftype_t type, priv_set_t *priv_set); int setfpriv(char *path, priv_op_t op, priv_ftype_t type, priv_set_t *priv_set); int fgetfpriv(int fd, priv_ftype_t type, priv_set_t *priv_set); int fsetfpriv(int fd, priv_op_t op, priv_ftype_t type, priv_set_t *priv_set);</pre>										
DESCRIPTION	<p>Set or get privileges of the file that is named by <i>path</i> or referred to by <i>fd</i>. <i>fgetfpriv()</i> and <i>fsetfpriv()</i> function exactly like <i>getfpriv()</i> and <i>setfpriv()</i> respectively, except that they require an open reference to a file as their argument.</p> <p><i>getfpriv()</i> copies the privilege set indicated by <i>type</i> and associated with the named file into the address specified by <i>priv_set</i>. Values for <i>type</i> are:</p> <table> <tr> <td>PRIV_FORCED</td><td>The forced privilege set.</td></tr> <tr> <td>PRIV_ALLOWED</td><td>The allowed privilege set.</td></tr> </table> <p>MAC read permission is required for the named file unless the privilege PRIV_FILE_MAC_READ is effective.</p> <p><i>setfpriv()</i> sets/modifies the privilege set (the target set) indicated by <i>type</i> and associated with the named file. Modification occurs according to the value of <i>op</i> and the privilege set specified by <i>priv_set</i> (the specified set). Values for <i>op</i> are:</p> <table> <tr> <td>PRIV_ON</td><td>Each privilege asserted in the specified set is asserted in the target set.</td></tr> <tr> <td>PRIV_OFF</td><td>Each privilege asserted in the specified set is cleared in the target set.</td></tr> <tr> <td>PRIV_SET</td><td>The target set is set exactly equal to the specified set.</td></tr> </table> <p>Values for <i>type</i> are the same as those used for <i>getfpriv()</i>.</p> <p>In all cases, the privilege PRIV_FILE_SETPRIV must be effective. In addition, only the owner of a file may change its privilege sets, unless the privilege PRIV_FILE_OWNER is effective.</p> <p>The invoking process must have MAC write permission for the named file (unless the privilege PRIV_FILE_MAC_WRITE is effective). DAC write access is not required.</p> <p>It is an error to attempt to assert a forced privilege if the corresponding allowed privilege is not present. For this reason, it is recommended that the allowed privilege set be modified first whenever both privilege sets are to be modified.</p>	PRIV_FORCED	The forced privilege set.	PRIV_ALLOWED	The allowed privilege set.	PRIV_ON	Each privilege asserted in the specified set is asserted in the target set.	PRIV_OFF	Each privilege asserted in the specified set is cleared in the target set.	PRIV_SET	The target set is set exactly equal to the specified set.
PRIV_FORCED	The forced privilege set.										
PRIV_ALLOWED	The allowed privilege set.										
PRIV_ON	Each privilege asserted in the specified set is asserted in the target set.										
PRIV_OFF	Each privilege asserted in the specified set is cleared in the target set.										
PRIV_SET	The target set is set exactly equal to the specified set.										

fgetfpriv(2)

If the target set is the allowed set, all privileges cleared from the target set are also automatically cleared from the forced set.

Normally MAC read permission is required or the privilege `PRIV_FILE_MAC_READ` must be effective for `getfpriv()` to complete its operation successfully unless the named file is a pty pseudo-terminal. If the named file is a pseudo-terminal (`/dev/ptyp*` or `/dev/ttyp*`) and the label of the process invoking `getfpriv()` does not dominate the label of the named file and the privilege `PRIV_FILE_MAC_READ` is not effective then `getfpriv()` returns success but sets the privilege fields of `priv_set` to zero.

RETURN VALUES

These routines return:

- 0 On success.
- 1 On failure, and set `errno` to indicate the error.

ERRORS

These routines fail and the target set is not modified if:

- `EINVAL` An illegal or undefined value is supplied for *size* or *type*.
- `EFAULT` *priv_set* refers to an invalid address.

Additionally, `getfpriv()` and `setfpriv()` fail if:

- `EACCES` Search permission is denied a component of *path*. To override this restriction, the calling process may assert the `PRIV_FILE_DAC_SEARCH` privilege and/or the `PRIV_FILE_MAC_SEARCH` privilege.

`getfpriv()` and `fgetfpriv()` fail if:

- `EACCES` MAC read permission is denied for the named file, and privilege `PRIV_FILE_MAC_READ` is not effective.
- `ENOENT` A component of the specified path does not exist.
- `ENOTDIR` A component of the specified path prefix is not a directory.
- `ENAMETOOLONG` The length of the path argument exceeds `PATH_MAX`, or a pathname component is longer than `NAME_MAX` while `_POSIX_NO_TRUNC` is in effect.

`setfpriv()` and `fsetfpriv()` fail and the target set is not modified if:

- `EACCES` MAC write permission is denied for the named file, privilege `PRIV_FILE_MAC_WRITE` is not effective, and the user's clearance dominates the sensitivity label of the file.
- `EINVAL` (1) The named file resides on a file system that does not support privileges (that is, a file system other than NFS, TMPFS) or (2) an illegal or undefined value is supplied for *op*. Also if privilege `PRIV_FILE_MAC_WRITE` is not effective.

fgetfpriv(2)

EPERM MAC write permission is denied for the named file, and the user's clearance does not dominate the label of the named file, or (2) PRIV_FILE_SETPRIV is not effective, or (3) the effective uid does not match the owner of the named file and privilege PRIV_FILE_OWNER is not effective.

EROFS The named file resides on a read-only file system.

getppriv(2), setppriv(2), priv_macros(5)

fgetfsattr(2)

NAME	getfsattr, fgetfsattr – Get filesystem security attributes																
SYNOPSIS	<pre>cc [flags...] file... -ltsol #include <tsol/fsattr.h> int getfsattr(char *path, u_long type, void *buf_p, int len); int fgetfsattr(int fd, u_long type, void *buf_p, int len);</pre>																
DESCRIPTION	<p>getfsattr() returns the file system security attributes of a mounted file system. <i>path</i> is the pathname of any file within the mounted file system. <i>type</i> is the type of attribute requested. Values for <i>type</i> are:</p> <table> <tr> <td>FSA_ACLCNT</td><td>The file system access ACL count.</td></tr> <tr> <td>FSA_ACL</td><td>The file system access ACL.</td></tr> <tr> <td>FSA_APRIV</td><td>The file system allowed privilege set.</td></tr> <tr> <td>FSA_FPRIV</td><td>The file system forced privilege set.</td></tr> <tr> <td>FSA_LABEL</td><td>The file system CMW label.</td></tr> <tr> <td>FSA_AFLAGS</td><td>The file system attribute flags.</td></tr> <tr> <td>FSA_LBLRNG</td><td>The file system label range.</td></tr> <tr> <td>FSA_MLDPFX</td><td>The file system MLD prefix string.</td></tr> </table> <p><i>buf_p</i> is a pointer to a buffer to hold the requested attribute, and <i>len</i> is the buffer length.</p> <p>fgetfsattr() returns the same information, but for an open file referred to by descriptor <i>fd</i>. <i>type</i>, <i>buf_p</i>, and <i>len</i> are the same as for getfsattr(). The information label of <i>path</i> or <i>fd</i> is unchanged.</p>	FSA_ACLCNT	The file system access ACL count.	FSA_ACL	The file system access ACL.	FSA_APRIV	The file system allowed privilege set.	FSA_FPRIV	The file system forced privilege set.	FSA_LABEL	The file system CMW label.	FSA_AFLAGS	The file system attribute flags.	FSA_LBLRNG	The file system label range.	FSA_MLDPFX	The file system MLD prefix string.
FSA_ACLCNT	The file system access ACL count.																
FSA_ACL	The file system access ACL.																
FSA_APRIV	The file system allowed privilege set.																
FSA_FPRIV	The file system forced privilege set.																
FSA_LABEL	The file system CMW label.																
FSA_AFLAGS	The file system attribute flags.																
FSA_LBLRNG	The file system label range.																
FSA_MLDPFX	The file system MLD prefix string.																
RETURN VALUES	<p>getfsattr() and fgetfsattr() return:</p> <table> <tr> <td>0</td><td>On success.</td></tr> <tr> <td>-1</td><td>On failure and set errno to indicate the error.</td></tr> </table>	0	On success.	-1	On failure and set errno to indicate the error.												
0	On success.																
-1	On failure and set errno to indicate the error.																
ERRORS	<p>getfsattr() fails if one or more of the following are true:</p> <table> <tr> <td>EACCES</td><td>Search permission is denied for a component of the path prefix of <i>path</i>. To override this restriction, the calling process may assert the PRIV_FILE_DAC_SEARCH privilege and/or the PRIV_FILE_MAC_SEARCH privilege.</td></tr> <tr> <td>EFAULT</td><td><i>buf_p</i> or <i>path</i> points to an invalid address.</td></tr> <tr> <td>EINVAL</td><td>The requested attributed is not set.</td></tr> <tr> <td>EIO</td><td>An I/O error occurred while reading from the file system.</td></tr> <tr> <td>ELOOP</td><td>Too many symbolic links were encountered in translating <i>path</i>.</td></tr> </table>	EACCES	Search permission is denied for a component of the path prefix of <i>path</i> . To override this restriction, the calling process may assert the PRIV_FILE_DAC_SEARCH privilege and/or the PRIV_FILE_MAC_SEARCH privilege.	EFAULT	<i>buf_p</i> or <i>path</i> points to an invalid address.	EINVAL	The requested attributed is not set.	EIO	An I/O error occurred while reading from the file system.	ELOOP	Too many symbolic links were encountered in translating <i>path</i> .						
EACCES	Search permission is denied for a component of the path prefix of <i>path</i> . To override this restriction, the calling process may assert the PRIV_FILE_DAC_SEARCH privilege and/or the PRIV_FILE_MAC_SEARCH privilege.																
EFAULT	<i>buf_p</i> or <i>path</i> points to an invalid address.																
EINVAL	The requested attributed is not set.																
EIO	An I/O error occurred while reading from the file system.																
ELOOP	Too many symbolic links were encountered in translating <i>path</i> .																

ENAMETOOLONG	The length of the path argument exceeds <code>PATH_MAX</code> . A pathname component is longer than <code>NAME_MAX</code> (see <code>sysconf(3C)</code>) while <code>_POSIX_NO_TRUNC</code> is in effect (see <code>pathconf(2)</code>).
ENOENT	The file referred to by <i>path</i> does not exist.
ENOTDIR	A component of the path prefix of <i>path</i> is not a directory.
<code>fgetfsattr()</code> fails if one or more of the following are true:	
EBADF	<i>fd</i> is not a valid open file descriptor.
EFAULT	<i>buf_p</i> points to an invalid address.
EINVAL	<i>fd</i> refers to a socket, not a file; or the requested attribute is not set.
EIO	An I/O error occurred while reading from the file system.

fgetmldadorn(2)

NAME	getmldadorn, fgetmldadorn – Get file system multilevel directory adornment
SYNOPSIS	<pre>cc [flags...] file... -ltsol #include <tsol/mld.h> int getmldadorn(char *path_name, char adorn_buf[MLD_ADORN_MAX]); int fgetmldadorn(intfd, char adorn_buf[MLD_ADORN_MAX]);</pre>
DESCRIPTION	<p>getmldadorn() returns the MLD adornment of the file system on which <i>path_name</i> resides. <i>path_name</i> is the path name of any file within the mounted filesystem. <i>adorn_buf</i> is a pointer to a buffer of at least MLD_ADORN_MAX bytes in which the null-terminated MLD adornment is returned.</p> <p>fgetmldadorn() returns the same information about an open file referred to by descriptor <i>fd</i>.</p> <p>The information label of <i>path_name</i> or <i>fd</i> is unchanged. The information label of the calling process is also unchanged.</p>
RETURN VALUES	<p>getmldadorn() and fgetmldadorn() return:</p> <ul style="list-style-type: none"> 0 On success. -1 On failure and set errno to indicate the error.
ERRORS	<p>getmldadorn() fails if one or more of the following are true:</p> <ul style="list-style-type: none"> EACCES Search permission is denied for a component of the path prefix of <i>path_name</i>. To override this restriction, the calling process may assert the PRIV_FILE_DAC_SEARCH privilege and/or the PRIV_FILE_MAC_SEARCH privilege. EFAULT <i>adorn_buf</i> or <i>path_name</i> points to an invalid address. EIO An I/O error occurred while reading from or writing to the file system. ELOOP Too many symbolic links were encountered in translating <i>path_name</i>. ENAMETOOLONG The length of the path argument exceeds PATH_MAX. A pathname component is longer than NAME_MAX (see sysconf(3C)) while _POSIX_NO_TRUNC is in effect (see pathconf(2)). ENOENT The file referred to by <i>path_name</i> does not exist. ENOTDIR A component of the path prefix of <i>path_name</i> is not a directory. <p>fgetmldadorn() fails if one or more of the following are true:</p> <ul style="list-style-type: none"> EBADF <i>fd</i> is not a valid open file descriptor.

EFAULT	<i>adorn_buf</i> points to an invalid address.
EINVAL	<i>fd</i> refers to a socket, not a file.
EIO	An I/O error occurred while reading from the file system.

WARNINGS

If the filesystem of the *fd* does not support MLDs and no `mld_prefix` attribute was specified at mount time, no error is returned, and a zero-length string is returned in the *adorn_buf* buffer.

**Trusted Solaris 8
4/01 Reference
Manual**

fgetslldname(2), getslldname(2)

fgetslldname(2)

NAME	getslldname, fgetslldname – Get file system single-level directory name				
SYNOPSIS	<pre>cc [flags...] file... -ltsol #include <tsol/label.h> int getslldname(char *path_name, bslabel_t *slabel_p, char *name_buf, const int length); int fgetslldname(int fd, const bslabel_t *slabel_p, char *name_buf, const int length);</pre>				
DESCRIPTION	<p>getslldname() returns the SLD name associated with the sensitivity label to which <i>slabel_p</i> refers within the context of the file system on which <i>path_name</i> resides. <i>path_name</i> is the path name of any multilevel directory within the mounted filesystem. <i>name_buf</i> is a pointer to a buffer of at least SLD_NAME_MAX bytes.</p> <p>fgetslldname() returns the SLD name associated with the sensitivity label to which <i>slabel_p</i> refers if the MLD to which descriptor <i>fd</i> refers was opened by the directory name (not by the fully adorned, multilevel directory name.) If the MLD to which descriptor <i>fd</i> refers was opened using the fully adorned, multilevel directory name, fgetslldname() returns the MLD and the SLD name associated with the sensitivity label to which <i>slabel_p</i> refers.</p> <p>If it does not exist, the single-level directory that corresponds to <i>slabel_p</i> is created with the attributes of the parent multilevel directory, the specified sensitivity label, and an ADMIN_LOW information label. If the sensitivity label of the calling process is equal to <i>slabel_p</i>, no additional privileges are needed. If the sensitivity label of the calling process is strictly dominated by <i>slabel_p</i>, the calling process may assert the PRIV_FILE_UPGRADE_SL privilege to create the directory. Otherwise, the calling process may assert the PRIV_FILE_DOWNGRADE_SL privilege to create the directory.</p>				
ATTRIBUTES	<p>See for descriptions of the following attributes:</p> <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>Availability</td><td>SUNWtsu</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
Availability	SUNWtsu				
RETURN VALUES	<p>getslldname() and fgetslldname() return:</p> <p>0 On success.</p> <p>-1 On failure and set errno to indicate the error.</p>				
ERRORS	<p>getslldname() fails if any of these conditions is true:</p> <p>EACCES Search permission is denied for a component of the path prefix of <i>path_name</i>. To override this restriction, the calling process may assert one or both of these privileges: PRIV_FILE_DAC_SEARCH and PRIV_FILE_MAC_SEARCH.</p>				

	The single-level directory specified does not exist, the system is configured to require write access to create a single-level directory, and the calling process does not have discretionary write access to <i>path_name</i> . To override this restriction, the calling process may assert the <code>PRIV_FILE_DAC_WRITE</code> privilege.
EFAULT	<i>name_buf</i> , <i>path_name</i> , or <i>slabel_p</i> points to an invalid address.
EIO	An I/O error occurred while reading from or writing to the file system
ELOOP	Too many symbolic links were encountered in translating <i>path_name</i> .
ENAMETOOLONG	The length of the path argument exceeds <code>PATH_MAX</code> . A pathname component is longer than <code>NAME_MAX</code> [see <code>sysconf(3C)</code>] while <code>_POSIX_NO_TRUNC</code> is in effect. [See <code>pathconf(2)</code> .]
ENOENT	The file to which <i>path_name</i> refers does not exist.
ENOTDIR	A component of the path prefix of <i>path_name</i> is not a directory.
EPERM	The SLD that corresponds to <i>slabel_p</i> does not exist and one of these conditions is true: the sensitivity label of the calling process is strictly dominated by <i>slabel_p</i> and the calling process has not asserted the <code>PRIV_FILE_DOWNGRADE</code> privilege; the sensitivity label of the calling process is not dominated by <i>slabel_p</i> and the calling process has not asserted the <code>PRIV_FILE_DOWNGRADE_SL</code> privilege.
<code>fgetslldname()</code> fails if any of these conditions is true:	
EBADF	<i>fd</i> is not a valid open file descriptor.
EFAULT	<i>name_buf</i> or <i>slabel_p</i> points to an invalid address.
EINVAL	<i>fd</i> does not refer to a multilevel directory.
EIO	An I/O error occurred while reading from the file system.
EPERM	The SLD that corresponds to <i>slabel_p</i> does not exist and one of these conditions is true: the sensitivity label of the calling process is strictly dominated by <i>slabel_p</i> and the calling process has not asserted the <code>PRIV_FILE_UPGRADE_SL</code> privilege; the sensitivity label of the calling process is not dominated by <i>slabel_p</i> and the calling process has not asserted the <code>PRIV_FILE_DOWNGRADE_SL</code> privilege.
WARNINGS	If the file system that contains <i>path_name</i> or the object referred to by <i>fd</i> does not support MLDs, no error is returned and the first <code>SLD_NAME_MAX</code> bytes in the <i>name_buf</i> are cleared.

fgetslidname(2)

Trusted Solaris 8
4/01 Reference
SunOS 5.8
Reference Manual

fgetmldadorn(2), getmldadorn(2)

sysconf(3C)

NAME	fork, fork1 – create a new process
SYNOPSIS	<pre>#include <sys/types.h> #include <unistd.h> pid_t fork(void); pid_t fork1(void);</pre>
DESCRIPTION	<p>The <code>fork()</code> and <code>fork1()</code> functions create a new process. The new process (child process) is an exact copy of the calling process (parent process). The child process inherits the following attributes from the parent process:</p> <ul style="list-style-type: none"> ■ real user ID, real group ID, effective user ID, effective group ID ■ environment ■ open file descriptors ■ close-on-exec flags (see <code>exec(2)</code>) ■ signal handling settings (that is, <code>SIG_DFL</code>, <code>SIG_IGN</code>, <code>SIG_HOLD</code>, function address) ■ supplementary group IDs ■ set-user-ID mode bit ■ set-group-ID mode bit ■ profiling on/off status ■ nice value (see <code>nice(2)</code>) ■ scheduler class (see <code>priocntl(2)</code>) ■ all attached shared memory segments (see <code>shmop(2)</code>) ■ process group ID -- memory mappings (see <code>mmap(2)</code>) ■ session ID (see <code>exit(2)</code>) ■ current working directory ■ root directory ■ file mode creation mask (see <code>umask(2)</code>) ■ resource limits (see <code>getrlimit(2)</code>) ■ controlling terminal ■ saved user ID and group ID ■ process attribute flags [See <code>getpattr(2)</code>.] ■ clearance [See <code>intro(2)</code>.] ■ sensitivity label [See <code>intro(2)</code>.] <p>Scheduling priority and any per-process scheduling parameters that are specific to a given scheduling class may or may not be inherited according to the policy of that particular class (see <code>priocntl(2)</code>). The child process differs from the parent process in the following ways:</p>

fork1(2)

- The child process has a unique process ID which does not match any active process group ID.
- The child process has a different parent process ID (that is, the process ID of the parent process).
- The child process has its own copy of the parent's file descriptors and directory streams. Each of the child's file descriptors shares a common file pointer with the corresponding file descriptor of the parent.
- Each shared memory segment remains attached and the value of `shm_nattach` is incremented by 1.
- All `semadj` values are cleared (see `semop(2)`).
- Process locks, text locks, data locks, and other memory locks are not inherited by the child (see `plock(3C)` and `memcntl(2)`).
- The child process's `tms` structure is cleared: `tms_utime`, `stime`, `cutime`, and `cstime` are set to 0 (see `times(2)`).
- The child processes resource utilizations are set to 0; see `getrlimit(2)`. The `it_value` and `it_interval` values for the `ITIMER_REAL` timer are reset to 0; see `getitimer(2)`.
- The set of signals pending for the child process is initialized to the empty set.
- Timers created by `timer_create(3RT)` are not inherited by the child process.
- No asynchronous input or asynchronous output operations are inherited by the child.

Record locks set by the parent process are not inherited by the child process (see `fcntl(2)`).

Solaris Threads

In applications that use the Solaris threads API rather than the POSIX threads API (applications linked with `-lthread` but not `-lpthread`), `fork()` duplicates in the child process all threads (see `thr_create(3THR)`) and LWPs in the parent process. The `fork1()` function duplicates only the calling thread (LWP) in the child process.

POSIX Threads

In applications that use the POSIX threads API rather than the Solaris threads API (applications linked with `-lpthread`, whether or not linked with `-lthread`), a call to `fork()` is like a call to `fork1()`, which replicates only the calling thread. There is no call that forks a child with all threads and LWPs duplicated in the child.

Note that if a program is linked with both libraries (`-lthread` and `-lpthread`), the POSIX semantic of `fork()` prevails.

fork() safety

If a Solaris threads application calls `fork1()` or a POSIX threads application calls `fork()`, and the child does more than simply call `exec()`, there is a possibility of deadlock occurring in the child. The application should use `pthread_atfork(3THR)` to ensure safety with respect to this deadlock. A Solaris threads application must explicitly link with `-lpthread` to access `pthread_atfork()`. Should there be any outstanding mutexes throughout the process, the application should call

	pthread_atfork() to wait for and acquire those mutexes prior to calling fork() or fork1(). See "MT-Level of Libraries" on the attributes(5) manual page.				
RETURN VALUES	Upon successful completion, fork() and fork1() return 0 to the child process and return the process ID of the child process to the parent process. Otherwise, (pid_t)-1 is returned to the parent process, no child process is created, and errno is set to indicate the error.				
ERRORS	The fork() function will fail if: <table> <tr> <td>EAGAIN</td><td>The system-imposed limit on the total number of processes under execution by a single user has been exceeded, and the calling process does not have the PRIV_SYS_MAXPROC effective privilege, or the total amount of system memory available is temporarily insufficient to duplicate this process.</td></tr> <tr> <td>ENOMEM</td><td>There is not enough swap space.</td></tr> </table>	EAGAIN	The system-imposed limit on the total number of processes under execution by a single user has been exceeded, and the calling process does not have the PRIV_SYS_MAXPROC effective privilege, or the total amount of system memory available is temporarily insufficient to duplicate this process.	ENOMEM	There is not enough swap space.
EAGAIN	The system-imposed limit on the total number of processes under execution by a single user has been exceeded, and the calling process does not have the PRIV_SYS_MAXPROC effective privilege, or the total amount of system memory available is temporarily insufficient to duplicate this process.				
ENOMEM	There is not enough swap space.				
ATTRIBUTES	See attributes(5) for descriptions of the following attributes: <table> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> <tr> <td>MT-Level</td><td>fork() is Async-Signal-Safe</td></tr> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	fork() is Async-Signal-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	fork() is Async-Signal-Safe				
SUMMARY OF TRUSTED SOLARIS CHANGES	Process attributes introduced by Trusted Solaris are all inheritable by the child process. A calling process with the PRIV_SYS_MAXPROC privilege is able to override the limit on the number of processes a user may have.				
Trusted Solaris 8 4/01 Reference Manual	exec(2), fcntl(2), getrlimit(2), nice(2), priocntl(2), semop(2), shmop(2), plock(3C)				
SunOS 5.8 Reference Manual	alarm(2), exit(2), getitimer(2), memcntl(2), mmap(2), ptrace(2), times(2), umask(2), wait(2), exit(3C), pthread_atfork(3THR), signal(3C), system(3C), thr_create(3THR), timer_create(3RT), attributes(5), standards(5)				
NOTES	<p>An applications should call _exit() rather than exit(3C) if it cannot execve(), since exit() will flush and close standard I/O channels and thereby corrupt the parent process's standard I/O data structures. Using exit(3C) will flush buffered data twice. See exit(2).</p> <p>The thread (or LWP) in the child that calls fork1() must not depend on any resources held by threads (or LWPs) that no longer exist in the child. In particular, locks held by these threads (or LWPs) will not be released.</p> <p>In a multithreaded process, fork() or fork1() can cause blocking system calls to be interrupted and return with an EINTR error.</p> <p>The fork() and fork1() functions suspend all threads in the process before proceeding. Threads that are executing in the kernel and are in an uninterruptible wait</p>				

`fork1(2)`

cannot be suspended immediately; and therefore cause a delay before `fork()` and `fork1()` can complete. During this delay, since all other threads will have already been suspended, the process will appear "hung."

NAME	fork, fork1 – create a new process
SYNOPSIS	<pre>#include <sys/types.h> #include <unistd.h> pid_t fork(void); pid_t fork1(void);</pre>
DESCRIPTION	<p>The <code>fork()</code> and <code>fork1()</code> functions create a new process. The new process (child process) is an exact copy of the calling process (parent process). The child process inherits the following attributes from the parent process:</p> <ul style="list-style-type: none"> ■ real user ID, real group ID, effective user ID, effective group ID ■ environment ■ open file descriptors ■ close-on-exec flags (see <code>exec(2)</code>) ■ signal handling settings (that is, <code>SIG_DFL</code>, <code>SIG_IGN</code>, <code>SIG_HOLD</code>, function address) ■ supplementary group IDs ■ set-user-ID mode bit ■ set-group-ID mode bit ■ profiling on/off status ■ nice value (see <code>nice(2)</code>) ■ scheduler class (see <code>priocntl(2)</code>) ■ all attached shared memory segments (see <code>shmop(2)</code>) ■ process group ID -- memory mappings (see <code>mmap(2)</code>) ■ session ID (see <code>exit(2)</code>) ■ current working directory ■ root directory ■ file mode creation mask (see <code>umask(2)</code>) ■ resource limits (see <code>getrlimit(2)</code>) ■ controlling terminal ■ saved user ID and group ID ■ process attribute flags [See <code>getpattr(2)</code>.] ■ clearance [See <code>intro(2)</code>.] ■ sensitivity label [See <code>intro(2)</code>.] <p>Scheduling priority and any per-process scheduling parameters that are specific to a given scheduling class may or may not be inherited according to the policy of that particular class (see <code>priocntl(2)</code>). The child process differs from the parent process in the following ways:</p>

fork(2)

- The child process has a unique process ID which does not match any active process group ID.
- The child process has a different parent process ID (that is, the process ID of the parent process).
- The child process has its own copy of the parent's file descriptors and directory streams. Each of the child's file descriptors shares a common file pointer with the corresponding file descriptor of the parent.
- Each shared memory segment remains attached and the value of `shm_nattach` is incremented by 1.
- All `semadj` values are cleared (see `semop(2)`).
- Process locks, text locks, data locks, and other memory locks are not inherited by the child (see `plock(3C)` and `memcntl(2)`).
- The child process's `tms` structure is cleared: `tms_utime`, `stime`, `cutime`, and `cstime` are set to 0 (see `times(2)`).
- The child processes resource utilizations are set to 0; see `getrlimit(2)`. The `it_value` and `it_interval` values for the `ITIMER_REAL` timer are reset to 0; see `getitimer(2)`.
- The set of signals pending for the child process is initialized to the empty set.
- Timers created by `timer_create(3RT)` are not inherited by the child process.
- No asynchronous input or asynchronous output operations are inherited by the child.

Record locks set by the parent process are not inherited by the child process (see `fcntl(2)`).

Solaris Threads

In applications that use the Solaris threads API rather than the POSIX threads API (applications linked with `-lthread` but not `-lpthread`), `fork()` duplicates in the child process all threads (see `thr_create(3THR)`) and LWPs in the parent process. The `fork1()` function duplicates only the calling thread (LWP) in the child process.

POSIX Threads

In applications that use the POSIX threads API rather than the Solaris threads API (applications linked with `-lpthread`, whether or not linked with `-lthread`), a call to `fork()` is like a call to `fork1()`, which replicates only the calling thread. There is no call that forks a child with all threads and LWPs duplicated in the child.

Note that if a program is linked with both libraries (`-lthread` and `-lpthread`), the POSIX semantic of `fork()` prevails.

fork() safety

If a Solaris threads application calls `fork1()` or a POSIX threads application calls `fork()`, and the child does more than simply call `exec()`, there is a possibility of deadlock occurring in the child. The application should use `pthread_atfork(3THR)` to ensure safety with respect to this deadlock. A Solaris threads application must explicitly link with `-lpthread` to access `pthread_atfork()`. Should there be any outstanding mutexes throughout the process, the application should call

	pthread_atfork() to wait for and acquire those mutexes prior to calling fork() or fork1(). See "MT-Level of Libraries" on the attributes(5) manual page.				
RETURN VALUES	Upon successful completion, fork() and fork1() return 0 to the child process and return the process ID of the child process to the parent process. Otherwise, (pid_t)-1 is returned to the parent process, no child process is created, and errno is set to indicate the error.				
ERRORS	The fork() function will fail if: <table> <tr> <td>EAGAIN</td><td>The system-imposed limit on the total number of processes under execution by a single user has been exceeded, and the calling process does not have the PRIV_SYS_MAXPROC effective privilege, or the total amount of system memory available is temporarily insufficient to duplicate this process.</td></tr> <tr> <td>ENOMEM</td><td>There is not enough swap space.</td></tr> </table>	EAGAIN	The system-imposed limit on the total number of processes under execution by a single user has been exceeded, and the calling process does not have the PRIV_SYS_MAXPROC effective privilege, or the total amount of system memory available is temporarily insufficient to duplicate this process.	ENOMEM	There is not enough swap space.
EAGAIN	The system-imposed limit on the total number of processes under execution by a single user has been exceeded, and the calling process does not have the PRIV_SYS_MAXPROC effective privilege, or the total amount of system memory available is temporarily insufficient to duplicate this process.				
ENOMEM	There is not enough swap space.				
ATTRIBUTES	See attributes(5) for descriptions of the following attributes: <table> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> <tr> <td>MT-Level</td><td>fork() is Async-Signal-Safe</td></tr> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	fork() is Async-Signal-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	fork() is Async-Signal-Safe				
SUMMARY OF TRUSTED SOLARIS CHANGES	Process attributes introduced by Trusted Solaris are all inheritable by the child process. A calling process with the PRIV_SYS_MAXPROC privilege is able to override the limit on the number of processes a user may have.				
Trusted Solaris 8 4/01 Reference Manual	exec(2), fcntl(2), getrlimit(2), nice(2), priocntl(2), semop(2), shmop(2), plock(3C)				
SunOS 5.8 Reference Manual	alarm(2), exit(2), getitimer(2), memcntl(2), mmap(2), ptrace(2), times(2), umask(2), wait(2), exit(3C), pthread_atfork(3THR), signal(3C), system(3C), thr_create(3THR), timer_create(3RT), attributes(5), standards(5)				
NOTES	<p>An applications should call _exit() rather than exit(3C) if it cannot execve(), since exit() will flush and close standard I/O channels and thereby corrupt the parent process's standard I/O data structures. Using exit(3C) will flush buffered data twice. See exit(2).</p> <p>The thread (or LWP) in the child that calls fork1() must not depend on any resources held by threads (or LWPs) that no longer exist in the child. In particular, locks held by these threads (or LWPs) will not be released.</p> <p>In a multithreaded process, fork() or fork1() can cause blocking system calls to be interrupted and return with an EINTR error.</p> <p>The fork() and fork1() functions suspend all threads in the process before proceeding. Threads that are executing in the kernel and are in an uninterruptible wait</p>				

`fork(2)`

cannot be suspended immediately; and therefore cause a delay before `fork()` and `fork1()` can complete. During this delay, since all other threads will have already been suspended, the process will appear "hung."

NAME

SYNOPSIS

DESCRIPTION

fpathconf, pathconf – Get configurable pathname variables

#include <unistd.h>

long int **fpathconf**(int *filde*, int *name*);

long int **pathconf**(const char **path*, int *name*);

The `fpathconf()` and `pathconf()` functions provide a method for the application to determine the current value of a configurable limit or option I (variable) that is associated with a file or directory.

For `pathconf()` , the *path* argument points to the pathname of a file or directory.

For `fpathconf()` , the *filde* argument is an open file descriptor.

The *name* argument represents the variable to be queried relative to that file or directory. The variables in the following table come from <limits.h> or <unistd.h> and the symbolic constants, defined in <unistd.h>, are the corresponding values used for *name*:

Variable	Value of <i>name</i>	Notes
FILESIZEBITS	_PC_FILESIZEBITS	3,4
LINK_MAX	_PC_LINK_MAX	1
MAX_CANON	_PC_MAX_CANON	2
MAX_INPUT	_PC_MAX_INPUT	2
NAME_MAX	_PC_NAME_MAX	3,4
PATH_MAX	_PC_PATH_MAX	4,5
PIPE_BUF	_PC_PIPE_BUF	6
_POSIX_CHOWN_RESTRICTED	_PC_CHOWN_RESTRICTED	7
_POSIX_NO_TRUNC	_PC_NO_TRUNC	3,4
_POSIX_VDISABLE	_PC_VDISABLE	2
_POSIX_ASYNC_IO	_PC_ASYNC_IO	8
_POSIX_PRIO_IO	_PC_PRIO_IO	8
_POSIX_SYNC_IO	_PC_SYNC_IO	8

Notes:

1. If *path* or *filde* refers to a directory, the value returned applies to the directory itself.

2. If *path* or *filde* does not refer to a terminal file, it is unspecified whether an implementation supports an association of the variable name with the specified

fpathconf(2)

file.

3. If *path* or *fildes* refers to a directory, the value returned applies to filenames within the directory.
4. If *path* or *fildes* does not refer to a directory, it is unspecified whether an implementation supports an association of the variable name with the specified file.
5. If *path* or *fildes* refers to a directory, the value returned is the maximum length of a relative pathname when the specified directory is the working directory.
6. If *path* refers to a FIFO, or *fildes* refers to a pipe or FIFO, the value returned applies to the referenced object. If *path* or *fildes* refers to a directory, the value returned applies to any FIFO that exists or can be created within the directory. If *path* or *fildes* refers to any other type of file, it is unspecified whether an implementation supports an association of the variable name with the specified file.
7. If *path* or *fildes* refers to a directory, the value returned applies to any files, other than directories, that exist or can be created within the directory.
8. If *path* or *fildes* refers to a directory, it is unspecified whether an implementation supports an association of the variable name with the specified file.

RETURN VALUES

If *name* is an invalid value, both `pathconf()` and `fpathconf()` return `-1` and `errno` is set to indicate the error.

If the variable corresponding to *name* has no limit for the *path* or file descriptor, both `pathconf()` and `fpathconf()` return `-1` without changing `errno`. If the implementation needs to use *path* to determine the value of *name* and the implementation does not support the association of *name* with the file specified by *path*, or if the process did not have appropriate privileges to query the *appropriate privileges* file specified by *path*, or *path* does not exist, `pathconf()` returns `-1` and `errno` is set to indicate the error.

If the implementation needs to use *fildes* to determine the value of *name* and the implementation does not support the association of *name* with the file specified by *fildes*, or if *fildes* is an invalid file descriptor, `fpathconf()` will return `-1` and `errno` is set to indicate the error.

Otherwise `pathconf()` or `fpathconf()` returns the current variable value for the file or directory without changing `errno`. The value returned will not be more restrictive than the corresponding value available to the application when it was compiled with the implementation's `<limits.h>` or `<unistd.h>`.

ERRORS

The `pathconf()` function will fail if:

EINVAL

The value of *name* is not valid.

ELOOP

Too many symbolic links were encountered in resolving *path*.

The `pathconf()` function may fail if:

EACCES	Search permission is denied for a component of the path prefix.
EINVAL	The implementation does not support an association of the variable <i>name</i> with the specified file.
ENAMETOOLONG	The length of the <i>path</i> argument exceeds PATH_MAX or a pathname component is longer than NAME_MAX.
ENAMETOOLONG	Pathname resolution of a symbolic link produced an intermediate result whose length exceeds PATH_MAX.
ENOENT	A component of <i>path</i> does not name an existing file or <i>path</i> is an empty string.
ENOTDIR	A component of the path prefix is not a directory.

The fpathconf () function will fail if:

EINVAL	The value of <i>name</i> is not valid.
EACCES	<i>fildev</i> is open only for writing and the calling process does not have mandatory read access to the object to which the descriptor refers. To override this restriction, the calling process may assert the PRIV_FILE_MAC_READ privilege.

The fpathconf () function may fail if:

EACCES	Search permission is denied for a component of the path prefix. To override this restriction, the calling process may assert one or both of these privileges: PRIV_FILE_DAC_SEARCH and PRIV_FILE_MAC_SEARCH. The calling process does not have mandatory read access to <i>path</i> . To override this restriction, the calling process may assert the PRIV_FILE_MAC_READ privilege.
EBADF	The <i>fildev</i> argument is not a valid file descriptor.
EINVAL	The implementation does not support an association of the variable <i>name</i> with the specified file.

ATTRIBUTES See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	pathconf () is Async-Signal-Safe

fpathconf(2)

**SUMMARY OF
TRUSTED
SOURCES
CHANGES
Reference Manual**

Appropriate privilege is required to override access checks.

sysconf(3C), limits(4), attributes(5), standards(5)

NAME	setcmwlabel, fsetcmwlabel, lsetcmwlabel – Set CMW label of a file
SYNOPSIS	<pre>cc [flags...] file ... -ltsol [library...] #include <tsol/label.h> int setcmwlabel(const char *path, const bclabel_t *label_p, const setting_flag_t flag); int fsetcmwlabel(int fd, const bclabel_t *label_p, const setting_flag_t flag); int lsetcmwlabel(const char *path, const bclabel_t *label_p, const setting_flag_t flag);</pre>
DESCRIPTION	<p>The file that is named by <i>path</i> or referred to by <i>fd</i> has its CMW label changed as specified provided the file resides on a file system that supports the setting of labels on individual objects.</p> <p>If <i>flag</i> equals SETCL_ALL, then both parts of the file's CMW label are to be set and the following checks must be made:</p> <ul style="list-style-type: none"> ■ The sensitivity label of <i>label_p</i> must be in the sensitivity label range of the containing file system. ■ If the sensitivity label of <i>label_p</i> equals the existing sensitivity label, then neither PRIV_FILE_UPGRADE_SL nor PRIV_FILE_DOWNGRADE_SL is required. ■ If the sensitivity label of <i>label_p</i> dominates but does not equal the existing sensitivity label (an upgrade), then the calling process must have PRIV_FILE_UPGRADE_SL in its set of effective privileges. ■ If the sensitivity label of <i>label_p</i> does not dominate the existing sensitivity label (a downgrade), then the calling process must have PRIV_FILE_DOWNGRADE_SL in its set of effective privileges. ■ If the sensitivity label operation is a downgrade and the calling process is not the owner of the file, then the calling process must have PRIV_FILE_OWNER in its set of effective privileges. <p>If <i>flag</i> equals SETCL_SL, then the sensitivity label of the file's CMW label is to be set and the following checks must be made:</p> <ul style="list-style-type: none"> ■ The sensitivity label of <i>label_p</i> must be in the sensitivity label range of the containing file system. ■ If the sensitivity label of <i>label_p</i> equals the existing sensitivity label, then neither PRIV_FILE_UPGRADE_SL nor PRIV_FILE_DOWNGRADE_SL is required. ■ If the sensitivity label of <i>label_p</i> dominates but does not equal the existing sensitivity label (an upgrade), then the calling process must have PRIV_FILE_UPGRADE_SL in its set of effective privileges. ■ If the sensitivity label of <i>label_p</i> does not dominate the existing sensitivity label (a downgrade), then the calling process must have PRIV_FILE_DOWNGRADE_SL in

fsetcmwlabel(2)

	<p>its set of effective privileges.</p> <ul style="list-style-type: none"> ■ If the operation is a sensitivity label downgrade and the calling process is not the owner of the file, then the calling process must have <code>PRIV_FILE_OWNER</code> in its set of effective privileges. <p>There are several checks that are applicable if the sensitivity label is being changed:</p> <ul style="list-style-type: none"> ■ The calling process must have discretionary write access to the file. ■ If there is an open file descriptor reference to the file, then the calling process must have <code>PRIV_PROC_TRANQUIL</code> in its set of effective privileges. <p><code>setcmwlabel()</code> and <code>lsetcmwlabel()</code> function identically except when the final component is a symbolic link. If the final component is a symbolic link, <code>lsetcmwlabel()</code> sets the CMW label of the symbolic link, but <code>setcmwlabel()</code> sets the CMW label of the object referred to by the symbolic link.</p>								
NOTES	<p>If the sensitivity label is being set, then the calling process is responsible for verifying that sensitivity label is within the accreditation range of the system.</p>								
RETURN VALUES	<p><code>setcmwlabel()</code>, <code>fsetcmwlabel()</code>, and <code>lsetcmwlabel()</code> return:</p> <p>0 On success.</p> <p>-1 On failure, and set <code>errno</code> to indicate the error.</p>								
ERRORS	<p><code>setcmwlabel()</code> and <code>lsetcmwlabel()</code> fail and the file is unchanged if any of these conditions prevails:</p> <table> <tr> <td>EACCES</td><td> <p>Search permission is denied for a component of the path prefix of <i>path</i>.</p> <p>The calling process does not have mandatory write access to the final component of <i>path</i> because the sensitivity label of the final component of <i>path</i> does not dominate the sensitivity label of the calling process and the calling process does not have <code>PRIV_FILE_MAC_WRITE</code> in its set of effective privileges.</p> <p>The calling process does not have discretionary write access to the final component of <i>path</i>.</p> </td></tr> <tr> <td>EBUSY</td><td> <p>There is an open file descriptor reference to the final component of <i>path</i> and the calling process does not have <code>PRIV_PROC_TRANQUIL</code> in its set of effective privileges.</p> </td></tr> <tr> <td>EFAULT</td><td> <p><i>path</i> or <i>label_p</i> points outside the allocated address space of the process.</p> </td></tr> <tr> <td>EINVAL</td><td> <p><i>path</i> does not reside on a file system that supports the setting of labels on individual objects.</p> </td></tr> </table>	EACCES	<p>Search permission is denied for a component of the path prefix of <i>path</i>.</p> <p>The calling process does not have mandatory write access to the final component of <i>path</i> because the sensitivity label of the final component of <i>path</i> does not dominate the sensitivity label of the calling process and the calling process does not have <code>PRIV_FILE_MAC_WRITE</code> in its set of effective privileges.</p> <p>The calling process does not have discretionary write access to the final component of <i>path</i>.</p>	EBUSY	<p>There is an open file descriptor reference to the final component of <i>path</i> and the calling process does not have <code>PRIV_PROC_TRANQUIL</code> in its set of effective privileges.</p>	EFAULT	<p><i>path</i> or <i>label_p</i> points outside the allocated address space of the process.</p>	EINVAL	<p><i>path</i> does not reside on a file system that supports the setting of labels on individual objects.</p>
EACCES	<p>Search permission is denied for a component of the path prefix of <i>path</i>.</p> <p>The calling process does not have mandatory write access to the final component of <i>path</i> because the sensitivity label of the final component of <i>path</i> does not dominate the sensitivity label of the calling process and the calling process does not have <code>PRIV_FILE_MAC_WRITE</code> in its set of effective privileges.</p> <p>The calling process does not have discretionary write access to the final component of <i>path</i>.</p>								
EBUSY	<p>There is an open file descriptor reference to the final component of <i>path</i> and the calling process does not have <code>PRIV_PROC_TRANQUIL</code> in its set of effective privileges.</p>								
EFAULT	<p><i>path</i> or <i>label_p</i> points outside the allocated address space of the process.</p>								
EINVAL	<p><i>path</i> does not reside on a file system that supports the setting of labels on individual objects.</p>								

	The sensitivity label of <i>label_p</i> is not in the sensitivity label range of the containing file system.
EIO	An I/O error occurred while reading from or writing to the file system.
ELOOP	Too many symbolic links were encountered in translating <i>path</i> .
ENAMETOOLONG	The length of the path argument exceeds <code>PATH_MAX</code> . A pathname component is longer than <code>NAME_MAX</code> [see <code>sysconf(3C)</code>] while <code>_POSIX_NO_TRUNC</code> is in effect. See <code>pathconf(2)</code> .
ENOENT	The file referred to by <i>path</i> does not exist.
ENOTDIR	A component of the path prefix of <i>path</i> is not a directory.
EPERM	The calling process does not have mandatory write access to the final component of <i>path</i> because the sensitivity label of the final component of <i>path</i> is outside the clearance of the calling process and the calling process does not have <code>PRIV_FILE_MAC_WRITE</code> in its set of effective privileges. A calling process that is not the owner of the file attempted to downgrade the sensitivity label associated with the final component of <i>path</i> but did not have <code>PRIV_FILE_OWNER</code> in its set of effective privileges. The calling process attempted to upgrade the sensitivity label associated with the final component of <i>path</i> but did not have <code>PRIV_FILE_UPGRADE_SL</code> in its set of effective privileges. The calling process attempted to downgrade the sensitivity label associated with the final component of <i>path</i> but did not have <code>PRIV_FILE_DOWNGRADE_SL</code> in its set of effective privileges.
EROFS	The file referred to by <i>path</i> resides on a read-only file system.
<code>fsetcmwlabel()</code> fails if any of these conditions prevails:	
EBADF	<i>fd</i> does not refer to a valid descriptor.
EBUSY	There is an open file descriptor reference to the object referred to by the descriptor and the calling process does not have <code>PRIV_PROC_TRANQUIL</code> in its set of effective privileges.
EFAULT	<i>label_p</i> points outside the allocated address space of the process.
EINVAL	<i>fd</i> refers to a socket, not a file. <i>fd</i> does not refer to a file on a file system that supports the setting of labels on individual objects.

fsetcmwlabel(2)

	<p>The sensitivity label of <i>label_p</i> is not in the sensitivity label range of the containing file system.</p>
EIO	<p>An I/O error occurred while reading from or writing to the file system.</p> <p>The calling process is not the owner of the file, attempted to downgrade the sensitivity label associated with the file, but did not have <code>PRIV_FILE_OWNER</code> in its set of effective privileges.</p> <p>The calling process attempted to upgrade the sensitivity label associated with the file but did not have <code>PRIV_FILE_UPGRADE_SL</code> in its set of effective privileges.</p> <p>The calling process attempted to downgrade the sensitivity label associated with the file but did not have <code>PRIV_FILE_DOWNGRADE_SL</code> in its set of effective privileges.</p>
EPERM	<p>The calling process does not have mandatory write access to the object referred to by <i>fd</i> because the sensitivity label of the object referred to by <i>fd</i> is outside the clearance of the calling process and the calling process does not have <code>PRIV_FILE_MAC_WRITE</code> in its set of effective privileges.</p> <p>A calling process that is not the owner of the file attempted to downgrade the sensitivity label associated with the object referred to by <i>fd</i> but did not have <code>PRIV_FILE_OWNER</code> in its set of effective privileges.</p> <p>The calling process attempted to upgrade the sensitivity label associated with the object referred to by <i>fd</i> but did not have <code>PRIV_FILE_UPGRADE_SL</code> in its set of effective privileges.</p> <p>The calling process attempted to downgrade the sensitivity label associated with the object referred to by <i>fd</i> but did not have <code>PRIV_FILE_DOWNGRADE_SL</code> in its set of effective privileges.</p>
EROFS	<p>The file referred to by <i>fd</i> resides on a read-only file system.</p>
getcmwfsrange(2), getcmwlabel(2)	

Trusted Solaris 8
4/01 Reference
Manual

NAME	getfattrflag, fsetfattrflag, fgetfattrflag, setfattrflag, mldgetfattrflag, mldsetfattrflag – set/get the security attribute flags of a file
SYNOPSIS	<pre>cc [flags...] file... -ltso1 #include <tso1/secflgs.h> int getfattrflag(const char *path, secflgs_t *flags); int setfattrflag(const char *path, secflgs_t which, secflgs_t flags); int fgetfattrflag(int fildes, secflgs_t *flags); int fsetfattrflag(int fildes, secflgs_t which, secflgs_t flags); int mldgetfattrflag(const char *path, secflgs_t *flags); int mldsetfattrflag(const char *path, secflgs_t which, secflgs_t flags);</pre>
DESCRIPTION	<p>setfattrflag(), fsetfattrflag(), and mldsetfattrflag() set the security flags of the file whose name is given by <i>path</i> or referred to by the open file descriptor <i>fildes</i>. The bit pattern contained in <i>which</i> is used to indicate which flags are being affected. The corresponding bits in <i>flags</i> are set to 1 or 0 to indicate whether the affected flags are being set or unset respectively.</p> <p>getfattrflag(), fgetfattrflag(), and mldgetfattrflag() get the security flags of the file whose name is given by <i>path</i> or referred to by the open file descriptor <i>fildes</i> and store it in the location pointed to by <i>flags</i>.</p> <p>Attribute bits are interpreted as follows:</p> <p>FAF_MLD Directory has MLD semantics.</p> <p>FAF_PUBLIC Filesystem object is a public object.</p> <p>FAF_SLD Directory is an SLD.</p> <p>Attribute flags are constructed by OR'ing the attribute flag bits.</p> <p>FAF_MLD is the only flag that may be modified without privilege if the directory is empty, the effective user ID of the process matches the directory owner, and the process has mandatory as well as discretionary write access. The FAF_MLD flag, once set, cannot be unset. Additionally, the FAF_MLD flag may only be set via the mldsetfattrflag interface. The FAF_PUBLIC flag can only be read or modified by a process possessing the PRIV_FILE_AUDIT privilege. A process attempting to read the FAF_PUBLIC flag without the PRIV_FILE_AUDIT privilege in effect will not fail. However the value of FAF_PUBLIC will be returned as unset. The FAF_SLD flag can never be set. The ability to read any flag is dependant upon the process having mandatory and discretionary read access to the file. The ability to set any flag is dependant upon the process having mandatory and discretionary write access to the file.</p>

fsetattrflag(2)

If *path* is a symbolic link, the target's attribute flags are affected rather than the link's. If *path* is a multilevel directory, `getfattrflag()` and `setattrflag()` will affect the underlying single-level directory beneath (unless *path* is adorned). `mltgetfattrflag()` and `mltsetattrflag()` do not translate multi-level directories to underlying single-level directories. `fgetfattrflag()` and `fsetattrflag()` affect only the file referred to by *fildev*.

RETURN VALUES

These functions return:

- 0 On success.
- 1 On failure, and set `errno` to indicate the error.

ERRORS

`getfattrflag()` and `mltgetfattrflag()` will fail if one or more of the following are true:

- EACCES Search permission is denied on a component of the path prefix of *path*. To override this restriction, the calling process may assert the `PRIV_FILE_DAC_SEARCH` privilege and/or the `PRIV_FILE_MAC_SEARCH` privilege.
 - EACCES Read permission is denied the final component of *path*. To override this restriction, the calling process may assert the `PRIV_FILE_MAC_READ` privilege.
 - EFAULT *path* points to an illegal address.
 - EINTR A signal was caught during execution of the function.
 - EIO An I/O error occurred while reading from the file system.
 - ELOOP Too many symbolic links were encountered in translating *path*.
 - EMULTIHOP Components of *path* require hopping to multiple remote machines and file system type does not allow it.
 - ENAMETOOLONG The length of the *path* argument exceeds `PATH_MAX`, or the length of a *path* component exceeds `NAME_MAX` while `_POSIX_NO_TRUNC` is in effect.
 - ENOENT Either a component of the path prefix, or the file referred to by *path* does not exist or is a null pathname.
 - ENOLINK *fildev* points to a remote machine and the link to that machine is no longer active.
 - ENOTDIR A component of the prefix of *path* is not a directory.
- `fgetfattrflag()` fails and the file mode is unchanged if:
- EACCES Read permission is denied on *fildev*. To override this restriction, the calling process may assert the `PRIV_FILE_MAC_READ` privilege.
 - EBADF *fildev* is not an open file descriptor.

EIO	An I/O error occurred while reading from the file system.
EINTR	A signal was caught during execution of the <code>fgetfattrflag()</code> function.
<code>setfattrflag()</code> and <code>ltdsetfattrflag()</code> will fail and the file mode is unchanged if one or more of the following are true:	
EACCES	Search permission is denied on a component of the path prefix of <i>path</i> . To override this restriction, the calling process may assert the <code>PRIV_FILE_DAC_SEARCH</code> privilege and/or the <code>PRIV_FILE_MAC_SEARCH</code> privilege.
EACCES	Write permission is denied <i>path</i> . To override this restriction, the calling process may assert the <code>PRIV_FILE_MAC_WRITE</code> privilege.
EACCES	The calling process does not own <i>files</i> . To override this restriction, the calling process may assert the <code>PRIV_FILE_OWNER</code> privilege.
EFAULT	<i>path</i> points to an illegal address.
EINTR	A signal was caught during execution of the function.
EINVAL	<i>path</i> is not a valid pathname. When setting <code>FAF_MLD</code> , <i>path</i> must refer to an empty directory.
EIO	An I/O error occurred while writing to the file system.
ELOOP	Too many symbolic links were encountered in translating <i>path</i> .
EMULTIHOP	Components of <i>path</i> require hopping to multiple remote machines and filesystem type does not allow it.
ENAMETOOLONG	The length of the <i>path</i> argument exceeds <code>PATH_MAX</code> , or the length of a <i>path</i> component exceeds <code>NAME_MAX</code> while <code>_POSIX_NO_TRUNC</code> is in effect.
ENOENT	Either a component of the path prefix, or the file referred to by <i>path</i> does not exist or is a null pathname.
ENOLINK	<i>path</i> points to a remote machine and the link to that machine is no longer active.
ENOTDIR	A component of the prefix of <i>path</i> is not a directory.
EPERM	The effective user ID does not match the owner of the file and the process does not possess the privilege <code>PRIV_FILE_OWNER</code> .
EPERM	The process does not possess the privilege <code>PRIV_FILE_AUDIT</code> and is attempting to set the <code>FAF_PUBLIC</code> flag.
EROFS	The file referred to by <i>path</i> resides on a read-only file system.
<code>fsetfattrflag()</code> fails and the file mode is unchanged if:	

fsetfattrflag(2)

EACCES	The calling process does not own <i>filde</i> s. To override this restriction, the calling process may assert the PRIV_FILE_OWNER privilege.
EACCES	Write access is denied on <i>filde</i> s. To override this restriction, the calling process may assert the PRIV_FILE_MAC_WRITE privilege.
EINVAL	<i>filde</i> s is not a valid pathname. When setting FAF_MLD, <i>filde</i> s must refer to an empty directory.
EBADF	<i>filde</i> s is not an open file descriptor.
EIO	An I/O error occurred while writing to the file system.
EINTR	A signal was caught during execution of the fsetfattrflag() function.
EPERM	The process does not possess the privilege PRIV_FILE_AUDIT and is attempting to set the FAF_PUBLIC flag.
EROFS	The file referred to by <i>filde</i> s resides on a read-only file system.

**Trusted Solaris 8
4/01 Reference
Manual**

setfattrflag(1), getfattrflag(1)

Trusted Solaris Developer's Guide

NAME	getfpriv, fgetfpriv, setfpriv, fsetfpriv – return or set a privilege set associated with a file										
SYNOPSIS	<pre>cc [flags...] file... -ltsol int getfpriv(char *path, priv_ftype_t type, priv_set_t *priv_set); int setfpriv(char *path, priv_op_t op, priv_ftype_t type, priv_set_t *priv_set); int fgetfpriv(int fd, priv_ftype_t type, priv_set_t *priv_set); int fsetfpriv(int fd, priv_op_t op, priv_ftype_t type, priv_set_t *priv_set);</pre>										
DESCRIPTION	<p>Set or get privileges of the file that is named by <i>path</i> or referred to by <i>fd</i>. <i>fgetfpriv()</i> and <i>fsetfpriv()</i> function exactly like <i>getfpriv()</i> and <i>setfpriv()</i> respectively, except that they require an open reference to a file as their argument.</p> <p><i>getfpriv()</i> copies the privilege set indicated by <i>type</i> and associated with the named file into the address specified by <i>priv_set</i>. Values for <i>type</i> are:</p> <table> <tr> <td>PRIV_FORCED</td><td>The forced privilege set.</td></tr> <tr> <td>PRIV_ALLOWED</td><td>The allowed privilege set.</td></tr> </table> <p>MAC read permission is required for the named file unless the privilege PRIV_FILE_MAC_READ is effective.</p> <p><i>setfpriv()</i> sets/modifies the privilege set (the target set) indicated by <i>type</i> and associated with the named file. Modification occurs according to the value of <i>op</i> and the privilege set specified by <i>priv_set</i> (the specified set). Values for <i>op</i> are:</p> <table> <tr> <td>PRIV_ON</td><td>Each privilege asserted in the specified set is asserted in the target set.</td></tr> <tr> <td>PRIV_OFF</td><td>Each privilege asserted in the specified set is cleared in the target set.</td></tr> <tr> <td>PRIV_SET</td><td>The target set is set exactly equal to the specified set.</td></tr> </table> <p>Values for <i>type</i> are the same as those used for <i>getfpriv()</i>.</p> <p>In all cases, the privilege PRIV_FILE_SETPRIV must be effective. In addition, only the owner of a file may change its privilege sets, unless the privilege PRIV_FILE_OWNER is effective.</p> <p>The invoking process must have MAC write permission for the named file (unless the privilege PRIV_FILE_MAC_WRITE is effective). DAC write access is not required.</p> <p>It is an error to attempt to assert a forced privilege if the corresponding allowed privilege is not present. For this reason, it is recommended that the allowed privilege set be modified first whenever both privilege sets are to be modified.</p>	PRIV_FORCED	The forced privilege set.	PRIV_ALLOWED	The allowed privilege set.	PRIV_ON	Each privilege asserted in the specified set is asserted in the target set.	PRIV_OFF	Each privilege asserted in the specified set is cleared in the target set.	PRIV_SET	The target set is set exactly equal to the specified set.
PRIV_FORCED	The forced privilege set.										
PRIV_ALLOWED	The allowed privilege set.										
PRIV_ON	Each privilege asserted in the specified set is asserted in the target set.										
PRIV_OFF	Each privilege asserted in the specified set is cleared in the target set.										
PRIV_SET	The target set is set exactly equal to the specified set.										

fsetfpriv(2)

If the target set is the allowed set, all privileges cleared from the target set are also automatically cleared from the forced set.

Normally MAC read permission is required or the privilege `PRIV_FILE_MAC_READ` must be effective for `getfpriv()` to complete its operation successfully unless the named file is a pty pseudo-terminal. If the named file is a pseudo-terminal (`/dev/ptyp*` or `/dev/ttyp*`) and the label of the process invoking `getfpriv()` does not dominate the label of the named file and the privilege `PRIV_FILE_MAC_READ` is not effective then `getfpriv()` returns success but sets the privilege fields of *priv_set* to zero.

RETURN VALUES

These routines return:

- 0 On success.
- 1 On failure, and set `errno` to indicate the error.

ERRORS

These routines fail and the target set is not modified if:

- `EINVAL` An illegal or undefined value is supplied for *size* or *type*.
- `EFAULT` *priv_set* refers to an invalid address.

Additionally, `getfpriv()` and `setfpriv()` fail if:

- `EACCES` Search permission is denied a component of *path*. To override this restriction, the calling process may assert the `PRIV_FILE_DAC_SEARCH` privilege and/or the `PRIV_FILE_MAC_SEARCH` privilege.

`getfpriv()` and `fgetfpriv()` fail if:

- `EACCES` MAC read permission is denied for the named file, and privilege `PRIV_FILE_MAC_READ` is not effective.
- `ENOENT` A component of the specified path does not exist.
- `ENOTDIR` A component of the specified path prefix is not a directory.
- `ENAMETOOLONG` The length of the path argument exceeds `PATH_MAX`, or a pathname component is longer than `NAME_MAX` while `_POSIX_NO_TRUNC` is in effect.

`setfpriv()` and `fsetfpriv()` fail and the target set is not modified if:

- `EACCES` MAC write permission is denied for the named file, privilege `PRIV_FILE_MAC_WRITE` is not effective, and the user's clearance dominates the sensitivity label of the file.
- `EINVAL` (1) The named file resides on a file system that does not support privileges (that is, a file system other than NFS, TMPFS) or (2) an illegal or undefined value is supplied for *op*. Also if privilege `PRIV_FILE_MAC_WRITE` is not effective.

		fsetfpriv(2)
EPERM	MAC write permission is denied for the named file, and the user's clearance does not dominate the label of the named file, or (2) PRIV_FILE_SETPRIV is not effective, or (3) the effective uid does not match the owner of the named file and privilege PRIV_FILE_OWNER is not effective.	
EROFS	The named file resides on a read-only file system.	
	getppriv(2), setppriv(2), priv_macros(5)	

fstat(2)

NAME	stat, lstat, fstat – get file status				
SYNOPSIS	<pre>#include <sys/types.h> #include <sys/stat.h> int stat(const char *path, struct stat *buf); int lstat(const char *path, struct stat *buf); int fstat(int fildes, struct stat *buf);</pre>				
DESCRIPTION	<p>The <code>stat()</code> function obtains information about the file pointed to by <i>path</i>. Read, write, or execute permission of the named file is not required, but all directories listed in the path name leading to the file must be searchable.</p> <p>The <code>lstat()</code> function obtains file attributes similar to <code>stat()</code>, except when the named file is a symbolic link; in that case <code>lstat()</code> returns information about the link, while <code>stat()</code> returns information about the file the link references.</p> <p>The <code>fstat()</code> function obtains information about an open file known by the file descriptor <i>fildes</i>, obtained from a successful <code>open(2)</code>, <code>creat(2)</code>, <code>dup(2)</code>, <code>fcntl(2)</code>, or <code>pipe(2)</code> function.</p> <p>The <i>buf</i> argument is a pointer to a <code>stat</code> structure into which information is placed concerning the file. A <code>stat</code> structure includes the following members:</p> <pre>mode_t st_mode; /* File mode (see mknod(2)) */ ino_t st_ino; /* Inode number */ dev_t st_dev; /* ID of device containing */ /* a directory entry for this file */ dev_t st_rdev; /* ID of device */ /* This entry is defined only for */ /* char special or block special files */ nlink_t st_nlink; /* Number of links */ uid_t st_uid; /* User ID of the file's owner */ gid_t st_gid; /* Group ID of the file's group */ off_t st_size; /* File size in bytes */ time_t st_atime; /* Time of last access */ time_t st_mtime; /* Time of last data modification */ time_t st_ctime; /* Time of last file status change */ /* Times measured in seconds since */ /* 00:00:00 UTC, Jan. 1, 1970 */ long st_blksize; /* Preferred I/O block size */ blkcnt_t st_blocks; /* Number of 512 byte blocks allocated*/</pre> <p>Descriptions of structure members are as follows:</p> <table><tr><td>st_mode</td><td>The mode of the file as described in <code>mknod(2)</code>. In addition to the modes described in <code>mknod()</code>, the mode of a file may also be <code>S_IFLNK</code> if the file is a symbolic link. <code>S_IFLNK</code> may only be returned by <code>lstat()</code>.</td></tr><tr><td>st_ino</td><td>This field uniquely identifies the file in a given file system. The pair <code>st_ino</code> and <code>st_dev</code> uniquely identifies regular files.</td></tr></table>	st_mode	The mode of the file as described in <code>mknod(2)</code> . In addition to the modes described in <code>mknod()</code> , the mode of a file may also be <code>S_IFLNK</code> if the file is a symbolic link. <code>S_IFLNK</code> may only be returned by <code>lstat()</code> .	st_ino	This field uniquely identifies the file in a given file system. The pair <code>st_ino</code> and <code>st_dev</code> uniquely identifies regular files.
st_mode	The mode of the file as described in <code>mknod(2)</code> . In addition to the modes described in <code>mknod()</code> , the mode of a file may also be <code>S_IFLNK</code> if the file is a symbolic link. <code>S_IFLNK</code> may only be returned by <code>lstat()</code> .				
st_ino	This field uniquely identifies the file in a given file system. The pair <code>st_ino</code> and <code>st_dev</code> uniquely identifies regular files.				

<code>st_dev</code>	This field uniquely identifies the file system that contains the file. Its value may be used as input to the <code>ustat()</code> function to determine more information about this file system. No other meaning is associated with this value.
<code>st_rdev</code>	This field should be used only by administrative commands. It is valid only for block special or character special files and only has meaning on the system where the file was configured.
<code>st_nlink</code>	This field should be used only by administrative commands.
<code>st_uid</code>	The user ID of the file's owner.
<code>st_gid</code>	The group ID of the file's group.
<code>st_size</code>	For regular files, this is the address of the end of the file. For block special or character special, this is not defined. See also <code>pipe(2)</code> .
<code>st_atime</code>	Time when file data was last accessed. Changed by the following functions: <code>creat()</code> , <code>mknod()</code> , <code>pipe()</code> , <code>utime(2)</code> , and <code>read(2)</code> .
<code>st_mtime</code>	Time when data was last modified. Changed by the following functions: <code>creat()</code> , <code>mknod()</code> , <code>pipe()</code> , <code>utime()</code> , and <code>write(2)</code> .
<code>st_ctime</code>	Time when file status was last changed. Changed by the following functions: <code>chmod()</code> , <code>chown()</code> , <code>creat()</code> , <code>link(2)</code> , <code>mknod()</code> , <code>pipe()</code> , <code>unlink(2)</code> , <code>utime()</code> , and <code>write()</code> .
<code>st_blksize</code>	A hint as to the "best" unit size for I/O operations. This field is not defined for block special or character special files.
<code>st_blocks</code>	The total number of physical blocks of size 512 bytes actually allocated on disk. This field is not defined for block special or character special files.

`stat()`, `lstat()`, and `fstat()` require mandatory read access to the final component of *path*. If the file descriptor is open only for writing, `fstat()` requires mandatory read access to the object to which the descriptor refers. To override these restrictions, the calling process may assert the `PRIV_FILE_MAC_READ` privilege in its set of effective privileges.

If the calling process does not have mandatory read access, `stat()`, `lstat()`, and `fstat()` return fixed values for some elements of the `stat` structure.

RETURN VALUES

Upon successful completion, 0 is returned. Otherwise, -1 is returned and `errno` is set to indicate the error.

ERRORS

The `stat()`, `fstat()`, and `lstat()` functions will fail if:

<code>EOVERFLOW</code>	The file size in bytes or the number of blocks allocated to the file or the file serial number cannot be represented correctly in the structure pointed to by <i>buf</i> .
------------------------	--

fstat(2)

The `stat()` and `lstat()` functions will fail if:

EACCES	Search permission is denied for a component of the path prefix. To override this restriction, the calling process may assert one or both of these privileges: <code>PRIV_FILE_DAC_SEARCH</code> and <code>PRIV_FILE_MAC_SEARCH</code> .
EFAULT	The <i>buf</i> or <i>path</i> argument points to an illegal address.
EINTR	A signal was caught during the execution of the <code>stat()</code> or <code>lstat()</code> function.
ELOOP	Too many symbolic links were encountered in translating <i>path</i> .
ENAMETOOLONG	The length of the <i>path</i> argument exceeds <code>PATH_MAX</code> , or the length of a <i>path</i> component exceeds <code>NAME_MAX</code> while <code>_POSIX_NO_TRUNC</code> is in effect.
ENOENT	The named file does not exist or is the null pathname.
ENOLINK	The <i>path</i> argument points to a remote machine and the link to that machine is no longer active.
ENOTDIR	A component of the path prefix is not a directory.
EOVERFLOW	A component is too large to store in the structure pointed to by <i>buf</i> .

The `fstat()` function will fail if:

EBADF	The <i>filides</i> argument is not a valid open file descriptor.
EFAULT	The <i>buf</i> argument points to an illegal address.
EINTR	A signal was caught during the execution of the <code>fstat()</code> function.
ENOLINK	The <i>filides</i> argument points to a remote machine and the link to that machine is no longer active.
EOVERFLOW	A component is too large to store in the structure pointed to by <i>buf</i> .

USAGE The `stat()`, `fstat()`, and `lstat()` functions have transitional interfaces for 64-bit file offsets. See `lf64(5)`.

ATTRIBUTES See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	<code>stat()</code> and <code>fstat()</code> are Async-Signal-Safe

SUMMARY OF TRUSTED SOLARIS CHANGES `stat()`, `lstat()`, and `fstat()` require mandatory read access to the final component of *path*. If the file descriptor is open only for writing, `fstat()` requires

fstat(2)

mandatory read access to the object to which the descriptor refers. To override these restrictions, the calling process may assert the `PRIV_FILE_MAC_READ` privilege in its set of effective privileges.

To override access restrictions, the calling process of `stat()` or `lstat()` may also assert one or both of these privileges: `PRIV_FILE_DAC_SEARCH` and `PRIV_FILE_MAC_SEARCH`.

Certain uses of this interface may present a covert channel. If a covert channel is exploited, the execution of the process may be delayed. To bypass this delay, the process may assert the `PRIV_PROC_NODELAY` privilege.

`chmod(2)`, `chown(2)`, `creat(2)`, `fcntl(2)`, `link(2)`, `mknod(2)`, `open(2)`, `read(2)`, `unlink(2)`, `write(2)`

`dup(2)`, `pipe(2)`, `time(2)`, `utime(2)`, `fattach(3C)`, `stat(3HEAD)`, `attributes(5)`

If you use `chmod(2)` to change the file group owner permissions on a file with ACL entries, both the file group owner permissions and the ACL mask are changed to the new permissions. Be aware that the new ACL mask permissions may change the effective permissions for additional users and groups who have ACL entries on the file.

fstatvfs(2)

NAME	statvfs, fstatvfs – get file system information
SYNOPSIS	<pre>#include <sys/types.h> #include <sys/statvfs.h> int statvfs(const char *path, struct statvfs *buf); int fstatvfs(int fildes, struct statvfs *buf);</pre>
DESCRIPTION	<p>The <code>statvfs()</code> function returns a “generic superblock” describing a file system; it can be used to acquire information about mounted file systems. The <i>buf</i> argument is a pointer to a structure (described below) that is filled by the function.</p> <p>The <i>path</i> argument should name a file that resides on that file system. The file system type is known to the operating system. Read, write, or execute permission for the named file is not required, but all directories listed in the path name leading to the file must be searchable.</p> <p>The <code>statvfs</code> structure pointed to by <i>buf</i> includes the following members:</p> <pre>u_long f_bsize; /* preferred file system block size */ u_long f_frsize; /* fundamental filesystem block (size if supported) */ fsblkcnt_t f_blocks; /* total # of blocks on file system in units of f_frsize */ fsblkcnt_t f_bfree; /* total # of free blocks */ fsblkcnt_t f_bavail; /* # of free blocks avail to non-super-user */ fsfilcnt_t f_files; /* total # of file nodes (inodes) */ fsfilcnt_t f_ffree; /* total # of free file nodes */ fsfilcnt_t f_favail; /* # of inodes avail to non-super-user */ u_long f_fsid; /* file system id (dev for now) */ char f_basetype[FSTYP SZ]; /* target fs type name, null-terminated */ u_long f_flag; /* bit mask of flags */ u_long f_namemax; /* maximum file name length */ char f_fstr[32]; /* file system specific string */ u_long f_filler[16]; /* reserved for future expansion */</pre> <p>The <code>f_basetype</code> member contains a null-terminated FSType name of the mounted target.</p> <p>The following values can be returned in the <code>f_flag</code> field:</p> <pre>ST_RDONLY 0x01 /* read-only file system */ ST_NOSUID 0x02 /* does not support setuid/setgid semantics */ ST_NOTRUNC 0x04 /* does not truncate file names longer than NAME_MAX */</pre> <p>The <code>fstatvfs()</code> function is similar to <code>statvfs()</code>, except that the file named by <i>path</i> in <code>statvfs()</code> is instead identified by an open file descriptor <i>fildes</i> obtained from a successful <code>open(2)</code>, <code>creat(2)</code>, <code>dup(2)</code>, <code>fcntl(2)</code>, or <code>pipe(2)</code> function call.</p>

RETURN VALUES

statvfs() returns:

- 0 On success.
- 1 On failure, and sets errno to indicate the error.

ERRORS

The statvfs() and fstatvfs() functions will fail if:

- EOVERFLOW One of the values to be returned cannot be represented correctly in the structure pointed to by *buf*.

The statvfs() function will fail if:

- EACCES Search permission is denied on a component of the path prefix. To override this restriction, the calling process may assert one or both of these privileges: PRIV_FILE_DAC_SEARCH and PRIV_FILE_MAC_SEARCH

The calling process does not have mandatory read access to *path_name*. To override this restriction, the calling process may assert the PRIV_FILE_MAC_READ privilege.

- EFAULT The *path* or *buf* argument points to an illegal address.
 - EINTR A signal was caught during the execution of the statvfs() function.
 - EIO An I/O error occurred while reading the file system.
 - ELOOP Too many symbolic links were encountered in translating *path*.
 - ENAMETOOLONG The length of a *path* component exceeds NAME_MAX characters, or the length of *path* exceeds PATH_MAX characters.
 - ENOENT Either a component of the path prefix or the file referred to by *path* does not exist.
 - ENOLINK The *path* argument points to a remote machine and the link to that machine is no longer active.
 - ENOTDIR A component of the path prefix of *path* is not a directory.
- The fstatvfs() function will fail if:
- EACCES The descriptor is open only for writing and the calling process does not have mandatory read access to the object to which the descriptor refers. To override this restriction, the calling process may assert the PRIV_FILE_MAC_READ privilege.
 - EBADF The *fdes* argument is not an open file descriptor.
 - EFAULT The *buf* argument points to an illegal address.
 - EINTR A signal was caught during the execution of the fstatvfs() function.

fstatvfs(2)

	EIO	An I/O error occurred while reading the file system.
USAGE	The statvfs() and fstatvfs() functions have transitional interfaces for 64-bit file offsets. See lf64(5).	
SUMMARY OF TRUSTED SQUARES CHANGES	Appropriate privilege is required to override access checks.	
Trusted Squares 4/01 Reference Manual	chmod(2), chown(2), creat(2), fcntl(2), link(2), mknod(2), open(2), read(2), unlink(2), write(2)	
SunOS 5.8 Reference Manual	dup(2), pipe(2), time(2), utime(2)	
BUGS	The values returned for f_files, f_ffree, and f_favail may not be valid for NFS mounted file systems.	

NAME	getaudit, setaudit, getaudit_addr, setaudit_addr – Get and set process audit information
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lbsm -lsocket -lnsl -lintl [<i>library</i> ...] #include <sys/param.h> #include <bsm/audit.h> int getaudit(struct auditinfo *<i>info</i>); int setaudit(struct auditinfo *<i>info</i>); int getaudit_addr(struct auditinfo_addr *<i>info</i>, int <i>length</i>); int setaudit_addr(struct auditinfo_addr *<i>info</i>, int <i>length</i>);</pre>
DESCRIPTION	<p>getaudit() gets the audit ID, the preselection mask, the terminal ID, and the audit session ID of the current process.</p> <p>Note that getaudit() may fail and return an E2BIG errno if the address field in the terminal ID is larger than 32 bits. In this case, getaudit_addr() should be used.</p> <p>setaudit() sets the audit ID, the preselection mask, the terminal ID, and the audit session ID for the current process.</p> <p>The getaudit_addr() function returns a variable length auditinfo_addr structure that contains the audit ID, the preselection mask, the terminal ID, and the audit session ID for the current process. The terminal ID contains a size field that indicates the size of the network address.</p> <p>The setaudit_addr() function sets the audit ID, the preselection mask, the terminal ID, and the audit session ID for the current process. The values are taken from the variable length structure auditinfo_addr. The terminal ID contains a size field that indicates the size of the network address.</p> <p>The info structure used to pass the process audit information contains the following members:</p> <pre>au_id_t ai_auid; /* audit user ID */ au_mask_t ai_mask; /* preselection mask */ au_tid_t ai_termid; /* terminal ID */ au_asid_t ai_asid; /* audit session ID */</pre> <p>To execute these commands successfully, a process needs certain privileges in its set of effective privileges: for getaudit(), a process needs PRIV_SYS_AUDIT, PRIV_PROC_AUDIT_TCB, or PRIV_PROC_AUDIT_APPL; for setaudit(), PRIV_SYS_AUDIT.</p>
RETURN VALUES	<p>getaudit() and setaudit() return:</p> <p>0 On success.</p> <p>-1 On failure, and set errno to indicate the error.</p>

getaudit(2)

ERRORS

The `getaudit()` and `setaudit()` functions will fail if:

- | | |
|---------------------|---|
| <code>EFAULT</code> | The <i>info</i> parameter points outside the process's allocated address space. |
| <code>EPERM</code> | The process did not have the appropriate privilege. |

USAGE

Only processes with the appropriate privileges may successfully execute these calls.

**SUMMARY OF
TRUSTED
SOLARIS
CHANGES**

This functionality is active only if auditing is enabled. By default, auditing is enabled in the Trusted Solaris environment. See *Trusted Solaris Audit Administration Manual* for more information.

As explained in DESCRIPTION, privileges are needed to run this command successfully.

**Trusted Solaris 8
4/01 Reference
Manual**

`audit(2)`

NAME	getaudit, setaudit, getaudit_addr, setaudit_addr – Get and set process audit information
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lbsm -lsocket -lnsl -lintl [<i>library</i> ...] #include <sys/param.h> #include <bsm/audit.h> int getaudit(struct auditinfo *<i>info</i>); int setaudit(struct auditinfo *<i>info</i>); int getaudit_addr(struct auditinfo_addr *<i>info</i>, int <i>length</i>); int setaudit_addr(struct auditinfo_addr *<i>info</i>, int <i>length</i>);</pre>
DESCRIPTION	<p>getaudit() gets the audit ID, the preselection mask, the terminal ID, and the audit session ID of the current process.</p> <p>Note that getaudit() may fail and return an E2BIG errno if the address field in the terminal ID is larger than 32 bits. In this case, getaudit_addr() should be used.</p> <p>setaudit() sets the audit ID, the preselection mask, the terminal ID, and the audit session ID for the current process.</p> <p>The getaudit_addr() function returns a variable length auditinfo_addr structure that contains the audit ID, the preselection mask, the terminal ID, and the audit session ID for the current process. The terminal ID contains a size field that indicates the size of the network address.</p> <p>The setaudit_addr() function sets the audit ID, the preselection mask, the terminal ID, and the audit session ID for the current process. The values are taken from the variable length structure auditinfo_addr. The terminal ID contains a size field that indicates the size of the network address.</p> <p>The info structure used to pass the process audit information contains the following members:</p> <pre>au_id_t ai_auid; /* audit user ID */ au_mask_t ai_mask; /* preselection mask */ au_tid_t ai_termid; /* terminal ID */ au_asid_t ai_asid; /* audit session ID */</pre> <p>To execute these commands successfully, a process needs certain privileges in its set of effective privileges: for getaudit(), a process needs PRIV_SYS_AUDIT, PRIV_PROC_AUDIT_TCB, or PRIV_PROC_AUDIT_APPL; for setaudit(), PRIV_SYS_AUDIT.</p>
RETURN VALUES	<p>getaudit() and setaudit() return:</p> <p>0 On success.</p> <p>-1 On failure, and set errno to indicate the error.</p>

getaudit_addr(2)

ERRORS

The `getaudit()` and `setaudit()` functions will fail if:

- | | |
|---------------------|---|
| <code>EFAULT</code> | The <i>info</i> parameter points outside the process's allocated address space. |
| <code>EPERM</code> | The process did not have the appropriate privilege. |

USAGE

Only processes with the appropriate privileges may successfully execute these calls.

**SUMMARY OF
TRUSTED
SOLARIS
CHANGES**

This functionality is active only if auditing is enabled. By default, auditing is enabled in the Trusted Solaris environment. See *Trusted Solaris Audit Administration Manual* for more information.

As explained in `DESCRIPTION`, privileges are needed to run this command successfully.

**Trusted Solaris 8
4/01 Reference
Manual**

`audit(2)`

NAME	getaudit, setaudit – Get and set user audit identity				
SYNOPSIS	<pre>cc [flags...] file ... -lbsm -lsocket -lnsl -lintl [library...] #include <sys/param.h> #include <bsm/audit.h> int getaudit(au_id_t *auid); int setaudit(au_id_t *auid);</pre>				
DESCRIPTION	<p>The <code>getaudit()</code> function returns the audit user ID for the current process. This value is initially set at login time and inherited by all child processes. This value does not change when the real/effective user IDs change, so it can be used to identify the logged-in user even when running a setuid program. The audit user ID governs audit decisions for a process.</p> <p>The <code>setaudit()</code> function sets the audit user ID for the current process.</p> <p>Only a process with the <code>PRIV_SYS_AUDIT</code> privilege asserted may successfully set its user identity. To get its identity successfully, a process must have <code>PRIV_SYS_AUDIT</code>, <code>PRIV_PROC_AUDIT_TCB</code>, or <code>PRIV_PROC_AUDIT_APPL</code> in its set of effective privileges.</p>				
RETURN VALUES	<p>Upon successful completion, the <code>getaudit()</code> function returns the audit user ID of the current process on success. Otherwise, it returns <code>-1</code> and sets <code>errno</code> to indicate the error.</p> <p>Upon successful completion the <code>setaudit()</code> function returns <code>0</code>. Otherwise, <code>-1</code> is returned and <code>errno</code> is set to indicate the error.</p>				
ERRORS	<p>The <code>getaudit()</code> and <code>setaudit()</code> functions will fail if:</p> <table> <tr> <td>EFAULT</td><td>The <i>auid</i> argument points to an invalid address.</td></tr> <tr> <td>EPERM</td><td>The process does not have the appropriate privileges.</td></tr> </table>	EFAULT	The <i>auid</i> argument points to an invalid address.	EPERM	The process does not have the appropriate privileges.
EFAULT	The <i>auid</i> argument points to an invalid address.				
EPERM	The process does not have the appropriate privileges.				
SUMMARY OF TRUSTED SOLARIS CHANGES	<p>This functionality is active only if auditing is enabled. By default, auditing is enabled in the Trusted Solaris environment. See <i>Trusted Solaris Audit Administration Manual</i> for more information.</p> <p>The privileges explained in DESCRIPTION are needed to run this command successfully.</p> <p>These system calls have been superseded by <code>getaudit()</code> and <code>setaudit()</code>.</p>				
Trusted Solaris 8 4/01 Reference Manual	audit(2), getaudit(2)				

getclearance(2)

NAME	getclearance – Get process clearance
SYNOPSIS	<pre>cc [flags...] file ... -ltsol [library...] #include <tsol/label.h> int getclearance(bclear_t *clearance_p) ;</pre>
DESCRIPTION	getclearance() obtains the clearance of the calling process. The clearance information is placed into the memory pointed to by <i>clearance_p</i> .
RETURN VALUES	getclearance() returns: 0 On success. -1 On failure, and sets errno to indicate the error.
ERRORS	getclearance() will fail (and <i>clearance_p</i> will not refer to a valid clearance) if this condition is true: EFAULT <i>clearance_p</i> points to an invalid address.
Trusted Solaris 8 4/01 Reference Manual	setclearance(2)

NAME	getcmwfsrange, fgetcmwfsrange – Get file system sensitivity label range														
SYNOPSIS	<pre>cc [flags...] file... -ltsol #include <tsol/label.h> int getcmwfsrange(char *path, brange_t *range_p); int fgetcmwfsrange(int fd, brange_t *range_p);</pre>														
DESCRIPTION	<p>getcmwfsrange() returns the sensitivity label range of a mounted file system. <i>path</i> is the path name of any file within the mounted filesystem. <i>range_p</i> is a pointer to a sensitivity label range structure defined as follows:</p> <pre>struct binary_level_range { blevel_t lower_bound; blevel_t upper_bound; }; typedef struct binary_level_range brange_t; /* Level Range */</pre> <p>fgetcmwfsrange() returns the same information about an open file referred to by descriptor <i>fd</i>.</p>														
RETURN VALUES	<p>getcmwfsrange() and fgetcmwfsrange() return:</p> <p>0 On success.</p> <p>-1 On failure, and set <i>errno</i> to indicate the error.</p>														
ERRORS	<p>getcmwfsrange() fails if one or more of the following are true:</p> <table> <tr> <td>EACCES</td><td>Search permission is denied for a component of the path prefix of <i>path</i>. To override this restriction, the calling process may assert the PRIV_FILE_DAC_SEARCH privilege and/or the PRIV_FILE_MAC_SEARCH privilege.</td></tr> <tr> <td>EFAULT</td><td><i>range_p</i> or <i>path</i> points to an invalid address.</td></tr> <tr> <td>EIO</td><td>An I/O error occurred while reading from or writing to the file system.</td></tr> <tr> <td>ELOOP</td><td>Too many symbolic links were encountered in translating <i>path</i>.</td></tr> <tr> <td>ENAMETOOLONG</td><td>The length of the path argument exceeds PATH_MAX. A pathname component is longer than NAME_MAX (see sysconf(3C)) while _POSIX_NO_TRUNC is in effect (see pathconf(2V)).</td></tr> <tr> <td>ENOENT</td><td>The file referred to by <i>path</i> does not exist.</td></tr> <tr> <td>ENOTDIR</td><td>A component of the path prefix of <i>path</i> is not a directory.</td></tr> </table> <p>fgetcmwfsrange() fails if one or more of the following are true:</p>	EACCES	Search permission is denied for a component of the path prefix of <i>path</i> . To override this restriction, the calling process may assert the PRIV_FILE_DAC_SEARCH privilege and/or the PRIV_FILE_MAC_SEARCH privilege.	EFAULT	<i>range_p</i> or <i>path</i> points to an invalid address.	EIO	An I/O error occurred while reading from or writing to the file system.	ELOOP	Too many symbolic links were encountered in translating <i>path</i> .	ENAMETOOLONG	The length of the path argument exceeds PATH_MAX. A pathname component is longer than NAME_MAX (see sysconf(3C)) while _POSIX_NO_TRUNC is in effect (see pathconf(2V)).	ENOENT	The file referred to by <i>path</i> does not exist.	ENOTDIR	A component of the path prefix of <i>path</i> is not a directory.
EACCES	Search permission is denied for a component of the path prefix of <i>path</i> . To override this restriction, the calling process may assert the PRIV_FILE_DAC_SEARCH privilege and/or the PRIV_FILE_MAC_SEARCH privilege.														
EFAULT	<i>range_p</i> or <i>path</i> points to an invalid address.														
EIO	An I/O error occurred while reading from or writing to the file system.														
ELOOP	Too many symbolic links were encountered in translating <i>path</i> .														
ENAMETOOLONG	The length of the path argument exceeds PATH_MAX. A pathname component is longer than NAME_MAX (see sysconf(3C)) while _POSIX_NO_TRUNC is in effect (see pathconf(2V)).														
ENOENT	The file referred to by <i>path</i> does not exist.														
ENOTDIR	A component of the path prefix of <i>path</i> is not a directory.														

getcmwfsrange(2)

EBADF	<i>fd</i> is not a valid open file descriptor.
EFAULT	<i>range_p</i> points to an invalid address.
EINVAL	<i>fd</i> refers to a socket, not a file.
EIO	An I/O error occurred while reading from the file system.

Trusted Solaris 8
4/01 Reference
Manual
SunOS 5.8
Reference Manual

pathconf(2)
sysconf(3C)

NAME	getcmwlabel, lgetcmwlabel, fgetcmwlabel – get file CMW label
SYNOPSIS	<pre>cc [flags...] file ... -ltsol [library...] #include <tsol/label.h> int getcmwlabel(char *path, bclabel_t *label_p); int lgetcmwlabel(char *path, bclabel_t *label_p); int fgetcmwlabel(int fd, bclabel_t *label_p);</pre>
DESCRIPTION	<p>getcmwlabel() obtains the CMW label of the file named by <i>path</i>. Mandatory read access to the final component of <i>path</i> is required or the calling process must have PRIV_FILE_MAC_READ in its set of effective privileges. Discretionary read, write or execute permission to the final component of <i>path</i> is not required, but all directories in the path prefix of <i>path</i> must be searchable.</p> <p>lgetcmwlabel() is like getcmwlabel() except in the case where the final component of <i>path</i> is a symbolic link, in which case lgetcmwlabel() returns the CMW label of the link, while getcmwlabel() returns the CMW label of the file to which the link refers.</p> <p>fgetcmwlabel() obtains the CMW label of an open file referred to by the argument descriptor, such as would be obtained by an open(2) call. If the descriptor is only open for writing, then mandatory read access to the object is required or the calling process must have PRIV_FILE_MAC_READ in its set of effective privileges.</p> <p><i>label_p</i> is a pointer to an opaque CMW label structure.</p> <p>An exception to the access rules applies in the case of pty pseudo-terminals (/dev/ptyp* and /dev/ttyp*). Normally mandatory read access is required or the calling process must have PRIV_FILE_MAC_READ in its set of effective privileges. If the specified file is a pty device file and the calling process does not have mandatory read access or PRIV_FILE_MAC_READ is not in its set of effective privileges, each function returns success and sets <i>label_p</i> to ADMIN_LOW.</p>
RETURN VALUES	<p>getcmwlabel(), lgetcmwlabel() and fgetcmwlabel() return:</p> <p>0 On success.</p> <p>-1 On failure, and set errno to indicate the error.</p>
ERRORS	<p>getcmwlabel() and lgetcmwlabel() fail if one or more of the following are true:</p> <p>EACCES Search permission is denied for a component of the path prefix of <i>path</i>. To override this restriction, the calling process may assert the PRIV_FILE_DAC_SEARCH privilege and/or the PRIV_FILE_MAC_SEARCH privilege.</p> <p> The calling process does not have mandatory read access to <i>path</i> because the sensitivity label of the calling process does not</p>

getcmwlabel(2)

	dominate the sensitivity label of the final component of <i>path</i> and the calling process does not have PRIV_FILE_MAC_READ in its set of effective privileges.
EFAULT	<i>label_p</i> or <i>path</i> points to an invalid address.
EIO	An I/O error occurred while reading from or writing to the file system.
ELOOP	Too many symbolic links were encountered in translating <i>path</i> .
ENAMETOOLONG	The length of the path argument exceeds PATH_MAX. A pathname component is longer than NAME_MAX while _POSIX_NO_TRUNC is in effect (see pathconf(2)).
ENOENT	The file referred to by <i>path</i> does not exist.
ENOTDIR	A component of the path prefix of <i>path</i> is not a directory.
EPERM	The calling process does not have mandatory read access to <i>path</i> because the sensitivity label of <i>path</i> is outside the calling process' clearance and the calling process does not have PRIV_FILE_MAC_READ in its set of effective privileges.
fgetcmwlabel() fails if one or more of the following are true:	
EACCES	The descriptor is only open for writing and the calling process does not have mandatory read access to the object referred to by the descriptor because the sensitivity label of the calling process does not dominate the sensitivity label of the object and the calling process does not have PRIV_FILE_MAC_READ in its set of effective privileges.
EBADF	<i>fd</i> is not a valid open file descriptor.
EFAULT	<i>label_p</i> points to an invalid address.
EIO	An I/O error occurred while reading from or writing to the file system.

pathconf(2), open(2), setcmwlabel(2)

NAME	getcmwplabel – Get process CMW label
SYNOPSIS	<pre>cc [flags...] file ... -ltsol [library...] #include <tsol/label.h> int getcmwplabel(bclabel_t *label_p);</pre>
DESCRIPTION	getcmwplabel() obtains the CMW label of the calling process. The label information is placed into the memory to which <i>label_p</i> points.
RETURN VALUES	<p>getcmwplabel() returns:</p> <p>0 On success.</p> <p>-1 On failure, and sets errno to indicate the error.</p>
ERRORS	<p>getcmwplabel() fails (and <i>label_p</i> does not refer to a valid CMW label) if this condition is true:</p> <p>EFAULT <i>label_p</i> points to an invalid address.</p>
SEE ALSO	setcmwplabel(2)

getdents(2)

NAME	getdents – Read directory entries and put in a file system independent format																				
SYNOPSIS	<pre>#include <sys/dirent.h> int getdents(int <i>fildes</i>, struct dirent *<i>buf</i>, size_t <i>nbyte</i>);</pre>																				
DESCRIPTION	<p>The <code>getdents()</code> function attempts to read <i>nbyte</i> bytes from the directory associated with the file descriptor <i>fildes</i> and to format them as file system independent directory entries in the buffer pointed to by <i>buf</i>. Since the file system independent directory entries are of variable lengths, in most cases the actual number of bytes returned will be less than <i>nbyte</i>. The file system independent directory entry is specified by the <code>dirent</code> structure. See <code>dirent(3HEAD)</code>.</p> <p>On devices capable of seeking, <code>getdents()</code> starts at a position in the file given by the file pointer associated with <i>fildes</i>. Upon return from <code>getdents()</code>, the file pointer is incremented to point to the next directory entry.</p>																				
RETURN VALUES	Upon successful completion, a non-negative integer is returned indicating the number of bytes actually read. A return value of 0 indicates the end of the directory has been reached. Otherwise, -1 is returned and <code>errno</code> is set to indicate the error.																				
ERRORS	<p>The <code>getdents()</code> function will fail if:</p> <table><tr><td>EACCESS</td><td>The calling process is not allowed to read the <i>procfs</i> file system. To override this restriction, the calling process may assert one or both of these privileges: <code>PRIV_FILE_DAC_READ</code> and <code>PRIV_FILE_MAC_READ</code>.</td></tr><tr><td></td><td>The system is configured to check mandatory read access to the directory entries. To override this restriction, the calling process may assert the <code>PRIV_FILE_MAC_READ</code> privilege.</td></tr><tr><td>EBADF</td><td>The <i>fildes</i> argument is not a valid file descriptor open for reading.</td></tr><tr><td>EFAULT</td><td>The <i>buf</i> argument points to an illegal address.</td></tr><tr><td>EINVAL</td><td>The <i>nbyte</i> argument is not large enough for one directory entry.</td></tr><tr><td>EIO</td><td>An I/O error occurred while accessing the file system.</td></tr><tr><td>ENOENT</td><td>The current file pointer for the directory is not located at a valid entry.</td></tr><tr><td>ENOLINK</td><td>The <i>fildes</i> argument points to a remote machine and the link to that machine is no longer active.</td></tr><tr><td>ENOTDIR</td><td>The <i>fildes</i> argument is not a directory.</td></tr><tr><td>EOVERFLOW</td><td>The value of the <code>dirent</code> structure member <code>d_ino</code> or <code>d_off</code> cannot be represented in an <code>ino_t</code> or <code>off_t</code>.</td></tr></table>	EACCESS	The calling process is not allowed to read the <i>procfs</i> file system. To override this restriction, the calling process may assert one or both of these privileges: <code>PRIV_FILE_DAC_READ</code> and <code>PRIV_FILE_MAC_READ</code> .		The system is configured to check mandatory read access to the directory entries. To override this restriction, the calling process may assert the <code>PRIV_FILE_MAC_READ</code> privilege.	EBADF	The <i>fildes</i> argument is not a valid file descriptor open for reading.	EFAULT	The <i>buf</i> argument points to an illegal address.	EINVAL	The <i>nbyte</i> argument is not large enough for one directory entry.	EIO	An I/O error occurred while accessing the file system.	ENOENT	The current file pointer for the directory is not located at a valid entry.	ENOLINK	The <i>fildes</i> argument points to a remote machine and the link to that machine is no longer active.	ENOTDIR	The <i>fildes</i> argument is not a directory.	EOVERFLOW	The value of the <code>dirent</code> structure member <code>d_ino</code> or <code>d_off</code> cannot be represented in an <code>ino_t</code> or <code>off_t</code> .
EACCESS	The calling process is not allowed to read the <i>procfs</i> file system. To override this restriction, the calling process may assert one or both of these privileges: <code>PRIV_FILE_DAC_READ</code> and <code>PRIV_FILE_MAC_READ</code> .																				
	The system is configured to check mandatory read access to the directory entries. To override this restriction, the calling process may assert the <code>PRIV_FILE_MAC_READ</code> privilege.																				
EBADF	The <i>fildes</i> argument is not a valid file descriptor open for reading.																				
EFAULT	The <i>buf</i> argument points to an illegal address.																				
EINVAL	The <i>nbyte</i> argument is not large enough for one directory entry.																				
EIO	An I/O error occurred while accessing the file system.																				
ENOENT	The current file pointer for the directory is not located at a valid entry.																				
ENOLINK	The <i>fildes</i> argument points to a remote machine and the link to that machine is no longer active.																				
ENOTDIR	The <i>fildes</i> argument is not a directory.																				
EOVERFLOW	The value of the <code>dirent</code> structure member <code>d_ino</code> or <code>d_off</code> cannot be represented in an <code>ino_t</code> or <code>off_t</code> .																				
USAGE	The <code>getdents()</code> function was developed to implement the <code>readdir(3C)</code> function and should not be used for other purposes.																				

getdents(2)

The `getdents()` function has a transitional interface for 64-bit file offsets. See `lf64(5)`.

Appropriate privilege is required to override access checks.

`readdir(3C)`, `dirent(3HEAD)`, `lf64(5)`

getfattrflag(2)

NAME	getfattrflag, fsetfattrflag, fgetfattrflag, setfattrflag, mldgetfattrflag, mldsetfattrflag – set/get the security attribute flags of a file
SYNOPSIS	<pre>cc [flags...] file... -ltsol #include <tsol/secflgs.h> int getfattrflag(const char *path, secflgs_t *flags); int setfattrflag(const char *path, secflgs_t which, secflgs_t flags); int fgetfattrflag(int fildes, secflgs_t *flags); int fsetfattrflag(int fildes, secflgs_t which, secflgs_t flags); int mldgetfattrflag(const char *path, secflgs_t *flags); int mldsetfattrflag(const char *path, secflgs_t which, secflgs_t flags);</pre>
DESCRIPTION	<p>setfattrflag(), fsetfattrflag(), and mldsetfattrflag() set the security flags of the file whose name is given by <i>path</i> or referred to by the open file descriptor <i>fildes</i>. The bit pattern contained in <i>which</i> is used to indicate which flags are being affected. The corresponding bits in <i>flags</i> are set to 1 or 0 to indicate whether the affected flags are being set or unset respectively.</p> <p>getfattrflag(), fgetfattrflag(), and mldgetfattrflag() get the security flags of the file whose name is given by <i>path</i> or referred to by the open file descriptor <i>fildes</i> and store it in the location pointed to by <i>flags</i>.</p> <p>Attribute bits are interpreted as follows:</p> <p>FAF_MLD Directory has MLD semantics.</p> <p>FAF_PUBLIC Filesystem object is a public object.</p> <p>FAF_SLD Directory is an SLD.</p> <p>Attribute flags are constructed by OR'ing the attribute flag bits.</p> <p>FAF_MLD is the only flag that may be modified without privilege if the directory is empty, the effective user ID of the process matches the directory owner, and the process has mandatory as well as discretionary write access. The FAF_MLD flag, once set, cannot be unset. Additionally, the FAF_MLD flag may only be set via the mldsetfattrflag interface. The FAF_PUBLIC flag can only be read or modified by a process possessing the PRIV_FILE_AUDIT privilege. A process attempting to read the FAF_PUBLIC flag without the PRIV_FILE_AUDIT privilege in effect will not fail. However the value of FAF_PUBLIC will be returned as unset. The FAF_SLD flag can never be set. The ability to read any flag is dependant upon the process having mandatory and discretionary read access to the file. The ability to set any flag is dependant upon the process having mandatory and discretionary write access to the file.</p>

If *path* is a symbolic link, the target's attribute flags are affected rather than the link's. If *path* is a multilevel directory, `getfattrflag()` and `setfattrflag()` will affect the underlying single-level directory beneath (unless *path* is adorned). `mldgetfattrflag()` and `mldsetfattrflag()` do not translate multi-level directories to underlying single-level directories. `fgetfattrflag()` and `fsetsattrflag()` affect only the file referred to by *filde*s.

RETURN VALUES

These functions return:

- 0 On success.
- 1 On failure, and set `errno` to indicate the error.

ERRORS

`getfattrflag()` and `mldgetfattrflag()` will fail if one or more of the following are true:

- EACCES** Search permission is denied on a component of the path prefix of *path*. To override this restriction, the calling process may assert the `PRIV_FILE_DAC_SEARCH` privilege and/or the `PRIV_FILE_MAC_SEARCH` privilege.
 - EACCES** Read permission is denied the final component of *path*. To override this restriction, the calling process may assert the `PRIV_FILE_MAC_READ` privilege.
 - EFAULT** *path* points to an illegal address.
 - EINTR** A signal was caught during execution of the function.
 - EIO** An I/O error occurred while reading from the file system.
 - ELOOP** Too many symbolic links were encountered in translating *path*.
 - EMULTIHOP** Components of *path* require hopping to multiple remote machines and file system type does not allow it.
 - ENAMETOOLONG** The length of the *path* argument exceeds `PATH_MAX`, or the length of a *path* component exceeds `NAME_MAX` while `_POSIX_NO_TRUNC` is in effect.
 - ENOENT** Either a component of the path prefix, or the file referred to by *path* does not exist or is a null pathname.
 - ENOLINK** *filde*s points to a remote machine and the link to that machine is no longer active.
 - ENOTDIR** A component of the prefix of *path* is not a directory.
- `fgetfattrflag()` fails and the file mode is unchanged if:
- EACCES** Read permission is denied on *filde*s. To override this restriction, the calling process may assert the `PRIV_FILE_MAC_READ` privilege.
 - EBADF** *filde*s is not an open file descriptor.

getfattrflag(2)

EIO	An I/O error occurred while reading from the file system.
EINTR	A signal was caught during execution of the <code>fgetfattrflag()</code> function.
<code>setfattrflag()</code> and <code>mltsetfattrflag()</code> will fail and the file mode is unchanged if one or more of the following are true:	
EACCES	Search permission is denied on a component of the path prefix of <i>path</i> . To override this restriction, the calling process may assert the <code>PRIV_FILE_DAC_SEARCH</code> privilege and/or the <code>PRIV_FILE_MAC_SEARCH</code> privilege.
EACCES	Write permission is denied <i>path</i> . To override this restriction, the calling process may assert the <code>PRIV_FILE_MAC_WRITE</code> privilege.
EACCES	The calling process does not own <i>files</i> . To override this restriction, the calling process may assert the <code>PRIV_FILE_OWNER</code> privilege.
EFAULT	<i>path</i> points to an illegal address.
EINTR	A signal was caught during execution of the function.
EINVAL	<i>path</i> is not a valid pathname. When setting <code>FAF_MLD</code> , <i>path</i> must refer to an empty directory.
EIO	An I/O error occurred while writing to the file system.
ELOOP	Too many symbolic links were encountered in translating <i>path</i> .
EMULTIHOP	Components of <i>path</i> require hopping to multiple remote machines and filesystem type does not allow it.
ENAMETOOLONG	The length of the <i>path</i> argument exceeds <code>PATH_MAX</code> , or the length of a <i>path</i> component exceeds <code>NAME_MAX</code> while <code>_POSIX_NO_TRUNC</code> is in effect.
ENOENT	Either a component of the path prefix, or the file referred to by <i>path</i> does not exist or is a null pathname.
ENOLINK	<i>path</i> points to a remote machine and the link to that machine is no longer active.
ENOTDIR	A component of the prefix of <i>path</i> is not a directory.
EPERM	The effective user ID does not match the owner of the file and the process does not possess the privilege <code>PRIV_FILE_OWNER</code> .
EPERM	The process does not possess the privilege <code>PRIV_FILE_AUDIT</code> and is attempting to set the <code>FAF_PUBLIC</code> flag.
EROFS	The file referred to by <i>path</i> resides on a read-only file system.
<code>fsetfattrflag()</code> fails and the file mode is unchanged if:	

EACCES	The calling process does not own <i>filde</i> s. To override this restriction, the calling process may assert the PRIV_FILE_OWNER privilege.
EACCES	Write access is denied on <i>filde</i> s. To override this restriction, the calling process may assert the PRIV_FILE_MAC_WRITE privilege.
EINVAL	<i>filde</i> s is not a valid pathname. When setting FAF_MLD, <i>filde</i> s must refer to an empty directory.
EBADF	<i>filde</i> s is not an open file descriptor.
EIO	An I/O error occurred while writing to the file system.
EINTR	A signal was caught during execution of the fsetfattrflag() function.
EPERM	The process does not possess the privilege PRIV_FILE_AUDIT and is attempting to set the FAF_PUBLIC flag.
EROFS	The file referred to by <i>filde</i> s resides on a read-only file system.

getfpriv(2)

NAME	getfpriv, fgetfpriv, setfpriv, fsetfpriv – return or set a privilege set associated with a file										
SYNOPSIS	<pre>cc [flags...] file... -ltsol int getfpriv(char *path, priv_ftype_t type, priv_set_t *priv_set); int setfpriv(char *path, priv_op_t op, priv_ftype_t type, priv_set_t *priv_set); int fgetfpriv(int fd, priv_ftype_t type, priv_set_t *priv_set); int fsetfpriv(int fd, priv_op_t op, priv_ftype_t type, priv_set_t *priv_set);</pre>										
DESCRIPTION	<p>Set or get privileges of the file that is named by <i>path</i> or referred to by <i>fd</i>. <i>fgetfpriv()</i> and <i>fsetfpriv()</i> function exactly like <i>getfpriv()</i> and <i>setfpriv()</i> respectively, except that they require an open reference to a file as their argument.</p> <p><i>getfpriv()</i> copies the privilege set indicated by <i>type</i> and associated with the named file into the address specified by <i>priv_set</i>. Values for <i>type</i> are:</p> <table> <tr> <td>PRIV_FORCED</td><td>The forced privilege set.</td></tr> <tr> <td>PRIV_ALLOWED</td><td>The allowed privilege set.</td></tr> </table> <p>MAC read permission is required for the named file unless the privilege PRIV_FILE_MAC_READ is effective.</p> <p><i>setfpriv()</i> sets/modifies the privilege set (the target set) indicated by <i>type</i> and associated with the named file. Modification occurs according to the value of <i>op</i> and the privilege set specified by <i>priv_set</i> (the specified set). Values for <i>op</i> are:</p> <table> <tr> <td>PRIV_ON</td><td>Each privilege asserted in the specified set is asserted in the target set.</td></tr> <tr> <td>PRIV_OFF</td><td>Each privilege asserted in the specified set is cleared in the target set.</td></tr> <tr> <td>PRIV_SET</td><td>The target set is set exactly equal to the specified set.</td></tr> </table> <p>Values for <i>type</i> are the same as those used for <i>getfpriv()</i>.</p> <p>In all cases, the privilege PRIV_FILE_SETPRIV must be effective. In addition, only the owner of a file may change its privilege sets, unless the privilege PRIV_FILE_OWNER is effective.</p> <p>The invoking process must have MAC write permission for the named file (unless the privilege PRIV_FILE_MAC_WRITE is effective). DAC write access is not required.</p> <p>It is an error to attempt to assert a forced privilege if the corresponding allowed privilege is not present. For this reason, it is recommended that the allowed privilege set be modified first whenever both privilege sets are to be modified.</p>	PRIV_FORCED	The forced privilege set.	PRIV_ALLOWED	The allowed privilege set.	PRIV_ON	Each privilege asserted in the specified set is asserted in the target set.	PRIV_OFF	Each privilege asserted in the specified set is cleared in the target set.	PRIV_SET	The target set is set exactly equal to the specified set.
PRIV_FORCED	The forced privilege set.										
PRIV_ALLOWED	The allowed privilege set.										
PRIV_ON	Each privilege asserted in the specified set is asserted in the target set.										
PRIV_OFF	Each privilege asserted in the specified set is cleared in the target set.										
PRIV_SET	The target set is set exactly equal to the specified set.										

If the target set is the allowed set, all privileges cleared from the target set are also automatically cleared from the forced set.

Normally MAC read permission is required or the privilege `PRIV_FILE_MAC_READ` must be effective for `getfpriv()` to complete its operation successfully unless the named file is a pty pseudo-terminal. If the named file is a pseudo-terminal (`/dev/ptyp*` or `/dev/ttyp*`) and the label of the process invoking `getfpriv()` does not dominate the label of the named file and the privilege `PRIV_FILE_MAC_READ` is not effective then `getfpriv()` returns success but sets the privilege fields of `priv_set` to zero.

RETURN VALUES

These routines return:

- 0 On success.
- 1 On failure, and set `errno` to indicate the error.

ERRORS

These routines fail and the target set is not modified if:

- `EINVAL` An illegal or undefined value is supplied for *size* or *type*.
- `EFAULT` *priv_set* refers to an invalid address.

Additionally, `getfpriv()` and `setfpriv()` fail if:

- `EACCES` Search permission is denied a component of *path*. To override this restriction, the calling process may assert the `PRIV_FILE_DAC_SEARCH` privilege and/or the `PRIV_FILE_MAC_SEARCH` privilege.

`getfpriv()` and `fgetfpriv()` fail if:

- `EACCES` MAC read permission is denied for the named file, and privilege `PRIV_FILE_MAC_READ` is not effective.
- `ENOENT` A component of the specified path does not exist.
- `ENOTDIR` A component of the specified path prefix is not a directory.
- `ENAMETOOLONG` The length of the path argument exceeds `PATH_MAX`, or a pathname component is longer than `NAME_MAX` while `_POSIX_NO_TRUNC` is in effect.

`setfpriv()` and `fsetfpriv()` fail and the target set is not modified if:

- `EACCES` MAC write permission is denied for the named file, privilege `PRIV_FILE_MAC_WRITE` is not effective, and the user's clearance dominates the sensitivity label of the file.
- `EINVAL` (1) The named file resides on a file system that does not support privileges (that is, a file system other than NFS, TMPFS) or (2) an illegal or undefined value is supplied for *op*. Also if privilege `PRIV_FILE_MAC_WRITE` is not effective.

getfpriv(2)

EPERM	MAC write permission is denied for the named file, and the user's clearance does not dominate the label of the named file, or (2) PRIV_FILE_SETPRIV is not effective, or (3) the effective uid does not match the owner of the named file and privilege PRIV_FILE_OWNER is not effective.
EROFS	The named file resides on a read-only file system.

**Trusted Solaris 8
4/01 Reference
Manual**

getppriv(2), setppriv(2), priv_macros(5)

NAME	getfsattr, fgetfsattr – Get filesystem security attributes																
SYNOPSIS	<pre>cc [flags...] file... -ltsol #include <tsol/fsattr.h> int getfsattr(char *path, u_long type, void *buf_p, int len); int fgetfsattr(int fd, u_long type, void *buf_p, int len);</pre>																
DESCRIPTION	<p>getfsattr() returns the file system security attributes of a mounted file system. <i>path</i> is the pathname of any file within the mounted file system. <i>type</i> is the type of attribute requested. Values for <i>type</i> are:</p> <table> <tr> <td>FSA_ACLCNT</td><td>The file system access ACL count.</td></tr> <tr> <td>FSA_ACL</td><td>The file system access ACL.</td></tr> <tr> <td>FSA_APRIV</td><td>The file system allowed privilege set.</td></tr> <tr> <td>FSA_FPRIV</td><td>The file system forced privilege set.</td></tr> <tr> <td>FSA_LABEL</td><td>The file system CMW label.</td></tr> <tr> <td>FSA_AFLAGS</td><td>The file system attribute flags.</td></tr> <tr> <td>FSA_LBLRNG</td><td>The file system label range.</td></tr> <tr> <td>FSA_MLDPFX</td><td>The file system MLD prefix string.</td></tr> </table> <p><i>buf_p</i> is a pointer to a buffer to hold the requested attribute, and <i>len</i> is the buffer length.</p> <p>fgetfsattr() returns the same information, but for an open file referred to by descriptor <i>fd</i>. <i>type</i>, <i>buf_p</i>, and <i>len</i> are the same as for getfsattr(). The information label of <i>path</i> or <i>fd</i> is unchanged.</p>	FSA_ACLCNT	The file system access ACL count.	FSA_ACL	The file system access ACL.	FSA_APRIV	The file system allowed privilege set.	FSA_FPRIV	The file system forced privilege set.	FSA_LABEL	The file system CMW label.	FSA_AFLAGS	The file system attribute flags.	FSA_LBLRNG	The file system label range.	FSA_MLDPFX	The file system MLD prefix string.
FSA_ACLCNT	The file system access ACL count.																
FSA_ACL	The file system access ACL.																
FSA_APRIV	The file system allowed privilege set.																
FSA_FPRIV	The file system forced privilege set.																
FSA_LABEL	The file system CMW label.																
FSA_AFLAGS	The file system attribute flags.																
FSA_LBLRNG	The file system label range.																
FSA_MLDPFX	The file system MLD prefix string.																
RETURN VALUES	<p>getfsattr() and fgetfsattr() return:</p> <table> <tr> <td>0</td><td>On success.</td></tr> <tr> <td>-1</td><td>On failure and set errno to indicate the error.</td></tr> </table>	0	On success.	-1	On failure and set errno to indicate the error.												
0	On success.																
-1	On failure and set errno to indicate the error.																
ERRORS	<p>getfsattr() fails if one or more of the following are true:</p> <table> <tr> <td>EACCES</td><td>Search permission is denied for a component of the path prefix of <i>path</i>. To override this restriction, the calling process may assert the PRIV_FILE_DAC_SEARCH privilege and/or the PRIV_FILE_MAC_SEARCH privilege.</td></tr> <tr> <td>EFAULT</td><td><i>buf_p</i> or <i>path</i> points to an invalid address.</td></tr> <tr> <td>EINVAL</td><td>The requested attributed is not set.</td></tr> <tr> <td>EIO</td><td>An I/O error occurred while reading from the file system.</td></tr> <tr> <td>ELOOP</td><td>Too many symbolic links were encountered in translating <i>path</i>.</td></tr> </table>	EACCES	Search permission is denied for a component of the path prefix of <i>path</i> . To override this restriction, the calling process may assert the PRIV_FILE_DAC_SEARCH privilege and/or the PRIV_FILE_MAC_SEARCH privilege.	EFAULT	<i>buf_p</i> or <i>path</i> points to an invalid address.	EINVAL	The requested attributed is not set.	EIO	An I/O error occurred while reading from the file system.	ELOOP	Too many symbolic links were encountered in translating <i>path</i> .						
EACCES	Search permission is denied for a component of the path prefix of <i>path</i> . To override this restriction, the calling process may assert the PRIV_FILE_DAC_SEARCH privilege and/or the PRIV_FILE_MAC_SEARCH privilege.																
EFAULT	<i>buf_p</i> or <i>path</i> points to an invalid address.																
EINVAL	The requested attributed is not set.																
EIO	An I/O error occurred while reading from the file system.																
ELOOP	Too many symbolic links were encountered in translating <i>path</i> .																

getfsattr(2)

ENAMETOOLONG	The length of the path argument exceeds PATH_MAX. A pathname component is longer than NAME_MAX (see sysconf(3C)) while _POSIX_NO_TRUNC is in effect (see pathconf(2)).
ENOENT	The file referred to by <i>path</i> does not exist.
ENOTDIR	A component of the path prefix of <i>path</i> is not a directory.
fgetfsattr() fails if one or more of the following are true:	
EBADF	<i>fd</i> is not a valid open file descriptor.
EFAULT	<i>buf_p</i> points to an invalid address.
EINVAL	<i>fd</i> refers to a socket, not a file; or the requested attribute is not set.
EIO	An I/O error occurred while reading from the file system.

NAME	getgroups, setgroups – Get or set supplementary group access list IDs								
SYNOPSIS	<pre>#include <unistd.h> int getgroups(int <i>gidsetsize</i>, gid_t *<i>grouplist</i>); int setgroups(int <i>ngroups</i>, const gid_t *<i>grouplist</i>);</pre>								
DESCRIPTION	<p>The <code>getgroups()</code> function gets the current supplemental group access list of the calling process and stores the result in the array of group IDs specified by <i>grouplist</i>. This array has <i>gidsetsize</i> entries and must be large enough to contain the entire list. This list cannot be larger than <code>NGROUPS_MAX</code>. If <i>gidsetsize</i> equals 0, <code>getgroups()</code> will return the number of groups to which the calling process belongs without modifying the array pointed to by <i>grouplist</i>.</p> <p>The <code>setgroups()</code> function sets the supplementary group access list of the calling process from the array of group IDs specified by <i>grouplist</i>. The number of entries is specified by <i>ngroups</i> and can not be greater than <code>NGROUPS_MAX</code>. The calling process must have <code>PRIV_PROC_SETID</code> in its set of effective privileges to set new groups. If <code>PRIV_PROC_SETID</code> is not in the effective privilege set, the operation fails and sets <code>errno</code> to <code>EPERM</code>.</p>								
RETURN VALUES	Upon successful completion, <code>getgroups()</code> returns the number of supplementary group IDs set for the calling process and <code>setgroups()</code> returns 0. Otherwise, <code>-1</code> is returned and <code>errno</code> is set to indicate the error.								
ERRORS	<p>The <code>getgroups()</code> and <code>setgroups()</code> functions will fail if:</p> <table> <tr> <td>EFAULT</td><td>A referenced part of the array pointed to by <i>grouplist</i> is an illegal address.</td></tr> </table> <p>The <code>getgroups()</code> function will fail if:</p> <table> <tr> <td>EINVAL</td><td>The value of <i>gidsetsize</i> is non-zero and less than the number of supplementary group IDs set for the calling process.</td></tr> </table> <p>The <code>setgroups()</code> function will fail if:</p> <table> <tr> <td>EINVAL</td><td>The value of <i>ngroups</i> is greater than <code>NGROUPS_MAX</code>.</td></tr> <tr> <td>EPERM</td><td>The calling process does not have the <code>PRIV_PROC_SETID</code> privilege.</td></tr> </table>	EFAULT	A referenced part of the array pointed to by <i>grouplist</i> is an illegal address.	EINVAL	The value of <i>gidsetsize</i> is non-zero and less than the number of supplementary group IDs set for the calling process.	EINVAL	The value of <i>ngroups</i> is greater than <code>NGROUPS_MAX</code> .	EPERM	The calling process does not have the <code>PRIV_PROC_SETID</code> privilege.
EFAULT	A referenced part of the array pointed to by <i>grouplist</i> is an illegal address.								
EINVAL	The value of <i>gidsetsize</i> is non-zero and less than the number of supplementary group IDs set for the calling process.								
EINVAL	The value of <i>ngroups</i> is greater than <code>NGROUPS_MAX</code> .								
EPERM	The calling process does not have the <code>PRIV_PROC_SETID</code> privilege.								
ATTRIBUTES	<p>See <code>attributes(5)</code> for descriptions of the following attributes:</p> <table> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> <tr> <td>MT-Level</td><td>Async-Signal-Safe</td></tr> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	Async-Signal-Safe				
ATTRIBUTE TYPE	ATTRIBUTE VALUE								
MT-Level	Async-Signal-Safe								
SUMMARY OF TRUSTED SOLARIS CHANGES	To set new groups, the calling process must have <code>PRIV_PROC_SETID</code> in its set of effective privileges.								

getgroups(2)

Trusted Solaris 8
4/01 Reference
Manual
Reference Manual

chown(2), setuid(2)

groups(1), getuid(2), getgrnam(3C), initgroups(3C), attributes(5)

NAME	getmldadorn, fgetmldadorn – Get file system multilevel directory adornment																
SYNOPSIS	<pre>cc [flags...] file... -ltsol #include <tsol/mld.h> int getmldadorn(char *path_name, char adorn_buf[MLD_ADORN_MAX]); int fgetmldadorn(intfd, char adorn_buf[MLD_ADORN_MAX]);</pre>																
DESCRIPTION	<p>getmldadorn() returns the MLD adornment of the file system on which <i>path_name</i> resides. <i>path_name</i> is the path name of any file within the mounted filesystem. <i>adorn_buf</i> is a pointer to a buffer of at least MLD_ADORN_MAX bytes in which the null-terminated MLD adornment is returned.</p> <p>fgetmldadorn() returns the same information about an open file referred to by descriptor <i>fd</i>.</p> <p>The information label of <i>path_name</i> or <i>fd</i> is unchanged. The information label of the calling process is also unchanged.</p>																
RETURN VALUES	<p>getmldadorn() and fgetmldadorn() return:</p> <p>0 On success.</p> <p>-1 On failure and set errno to indicate the error.</p>																
ERRORS	<p>getmldadorn() fails if one or more of the following are true:</p> <table> <tr> <td>EACCES</td><td>Search permission is denied for a component of the path prefix of <i>path_name</i>. To override this restriction, the calling process may assert the PRIV_FILE_DAC_SEARCH privilege and/or the PRIV_FILE_MAC_SEARCH privilege.</td></tr> <tr> <td>EFAULT</td><td><i>adorn_buf</i> or <i>path_name</i> points to an invalid address.</td></tr> <tr> <td>EIO</td><td>An I/O error occurred while reading from or writing to the file system.</td></tr> <tr> <td>ELOOP</td><td>Too many symbolic links were encountered in translating <i>path_name</i>.</td></tr> <tr> <td>ENAMETOOLONG</td><td>The length of the path argument exceeds PATH_MAX. A pathname component is longer than NAME_MAX (see sysconf(3C)) while _POSIX_NO_TRUNC is in effect (see pathconf(2)).</td></tr> <tr> <td>ENOENT</td><td>The file referred to by <i>path_name</i> does not exist.</td></tr> <tr> <td>ENOTDIR</td><td>A component of the path prefix of <i>path_name</i> is not a directory.</td></tr> </table> <p>fgetmldadorn() fails if one or more of the following are true:</p> <table> <tr> <td>EBADF</td><td><i>fd</i> is not a valid open file descriptor.</td></tr> </table>	EACCES	Search permission is denied for a component of the path prefix of <i>path_name</i> . To override this restriction, the calling process may assert the PRIV_FILE_DAC_SEARCH privilege and/or the PRIV_FILE_MAC_SEARCH privilege.	EFAULT	<i>adorn_buf</i> or <i>path_name</i> points to an invalid address.	EIO	An I/O error occurred while reading from or writing to the file system.	ELOOP	Too many symbolic links were encountered in translating <i>path_name</i> .	ENAMETOOLONG	The length of the path argument exceeds PATH_MAX. A pathname component is longer than NAME_MAX (see sysconf(3C)) while _POSIX_NO_TRUNC is in effect (see pathconf(2)).	ENOENT	The file referred to by <i>path_name</i> does not exist.	ENOTDIR	A component of the path prefix of <i>path_name</i> is not a directory.	EBADF	<i>fd</i> is not a valid open file descriptor.
EACCES	Search permission is denied for a component of the path prefix of <i>path_name</i> . To override this restriction, the calling process may assert the PRIV_FILE_DAC_SEARCH privilege and/or the PRIV_FILE_MAC_SEARCH privilege.																
EFAULT	<i>adorn_buf</i> or <i>path_name</i> points to an invalid address.																
EIO	An I/O error occurred while reading from or writing to the file system.																
ELOOP	Too many symbolic links were encountered in translating <i>path_name</i> .																
ENAMETOOLONG	The length of the path argument exceeds PATH_MAX. A pathname component is longer than NAME_MAX (see sysconf(3C)) while _POSIX_NO_TRUNC is in effect (see pathconf(2)).																
ENOENT	The file referred to by <i>path_name</i> does not exist.																
ENOTDIR	A component of the path prefix of <i>path_name</i> is not a directory.																
EBADF	<i>fd</i> is not a valid open file descriptor.																

getmldadorn(2)

EFAULT	<i>adorn_buf</i> points to an invalid address.
EINVAL	<i>fd</i> refers to a socket, not a file.
EIO	An I/O error occurred while reading from the file system.

WARNINGS	If the filesystem of the <i>fd</i> does not support MLDs and no <code>mld_prefix</code> attribute was specified at mount time, no error is returned, and a zero-length string is returned in the <i>adorn_buf</i> buffer.
-----------------	---

Trusted Solaris 8 4/01 Reference Manual	<code>fgetsldname(2)</code> , <code>getsldname(2)</code>
--	--

NAME	getmsgqcmwlabel, getshmcmwlabel, getsemcmwlabel – Get the CMW labels associated with System V IPC structures
SYNOPSIS	<pre>cc [flags...] file ... -ltsol [library...] #include <sys/tsol/ipcl.h> int getmsgqcmwlabel(int msgqid, bclabel_t *clabel); int getshmcmwlabel(int shmid, bclabel_t *clabel); int getsemcmwlabel(int semid, bclabel_t *clabel);</pre>
DESCRIPTION	<p>These functions return the value of the CMW labels associated with message queues, shared memory, and semaphore structures.</p> <p>getmsgqcmwlabel() returns the CMW label for the message queue identified by <i>msgqid</i> into the label buffer to which <i>clabel</i> points. The information label portion of the CMW label is undefined for message queues; therefore the sensitivity label portion may have to be extracted using getcsl(3TSOL) in order to be useful.</p> <p>getshmcmwlabel() returns the CMW label for the shared-memory segment identified by <i>shmid</i> into the label buffer to which <i>clabel</i> points.</p> <p>getsemcmwlabel() returns the CMW label for the semaphore array identified by <i>semid</i> into the label buffer to which <i>clabel</i> points.</p> <p>The calling process must have mandatory read access to the IPC or must have asserted the PRIV_IPC_MAC_READ privilege, and must have discretionary read access to the data structure or must have the PRIV_IPC_DAC_READ privilege in its set of effective privileges.</p>
RETURN VALUES	<p>getmsgqcmwlabel(), getshmcmwlabel(), and getsemcmwlabel() return:</p> <p>0 On success.</p> <p>–1 On failure, and sets errno to indicate the error.</p>
ERRORS	<p>These functions will fail if any of these conditions is true:</p> <p>EACCES Read access is denied to the calling process, which does not have one or both of these privileges in its set of effective privileges: PRIV_IPC_DAC_READ and PRIV_IPC_MAC_READ.</p> <p>EINVAL <i>msgqid</i>, <i>semid</i>, or <i>shmid</i> is not a valid IPC object identifier.</p> <p>EFAULT <i>clabel</i> points to an illegal address.</p>
Trusted Solaris 8 4/01 Reference Manual	msgget(2), semget(2), shmget(2)

getpattr(2)

NAME	getpattr, setpattr – get/set process attribute flags																
SYNOPSIS	<pre>cc [flags...] file... -ltsol #include <tsol/pattr.h> int getpattr(pattr_type_t type, pattr_flag_t *value); int setpattr(pattr_type_t type, pattr_flag_t value);</pre>																
DESCRIPTION	<p>Process attribute flags are a set of flags that describe additional attributes that the process has. Each flag in the set is separately addressable although all flags share the <code>getpattr()</code> and the <code>setpattr()</code> system call interfaces. Likewise, each flag in the set has its own protection policy although all flags use the same protection mechanism. In the set are seven types of flags, specified in <code><tsol/pattr.h></code>, and addressed by the <i>type</i> argument. These are the values for <i>type</i>:</p> <table><tr><td>PAF_TRUSTED_PATH</td><td>Trusted path flag</td></tr><tr><td>PAF_PRIV_DBG</td><td>Privilege debugging flag</td></tr><tr><td>PAF_TOKMAPPER</td><td>Network token mapping process flag</td></tr><tr><td>PAF_LABEL_VIEW</td><td>Label view flags</td></tr><tr><td>PAF_LABEL_XLATE</td><td>Label translation flags</td></tr><tr><td>PAF_DISKLESS_BOOT</td><td>Part of diskless boot flag</td></tr><tr><td>PAF_SELAGNT</td><td>Part of selection agent flag</td></tr><tr><td>PAF_PRINT_SYSTEM</td><td>Part of trusted printing system flag</td></tr></table> <p>A description of each type of process attribute flag follows.</p> <p>Trusted path flag This one-bit flag marks a trusted path process. This flag can be viewed and cleared, but cannot be set. In other words, the call to <code>setpattr(PAF_TRUSTED_PATH, PAF_TP_ON)</code> will always fail. A process inherits the trusted path flag from its parent process. The <code>init</code> process receives the trusted path flag from the system. A user session creator, such as <code>login</code>, clears this flag before starting a user session.</p> <pre>setpattr(PAF_TRUSTED_PATH, PAF_TP_OFF)</pre> <p>Privilege debugging flag This one-bit flag indicates that the process is in privilege-debugging mode—a process-operation mode in which privilege requirement is logged but not enforced. This flag can be viewed or cleared, but cannot be set except by a trusted path process.</p> <p>Network token mapping process flag This one-bit flag, when set, identifies the process as the network token mapping process. The network token mapping process is exempt from network token mapping. This flag can be viewed and cleared, but cannot be set except by a trusted path process.</p>	PAF_TRUSTED_PATH	Trusted path flag	PAF_PRIV_DBG	Privilege debugging flag	PAF_TOKMAPPER	Network token mapping process flag	PAF_LABEL_VIEW	Label view flags	PAF_LABEL_XLATE	Label translation flags	PAF_DISKLESS_BOOT	Part of diskless boot flag	PAF_SELAGNT	Part of selection agent flag	PAF_PRINT_SYSTEM	Part of trusted printing system flag
PAF_TRUSTED_PATH	Trusted path flag																
PAF_PRIV_DBG	Privilege debugging flag																
PAF_TOKMAPPER	Network token mapping process flag																
PAF_LABEL_VIEW	Label view flags																
PAF_LABEL_XLATE	Label translation flags																
PAF_DISKLESS_BOOT	Part of diskless boot flag																
PAF_SELAGNT	Part of selection agent flag																
PAF_PRINT_SYSTEM	Part of trusted printing system flag																

Label view flags	These two-bit flags support per-process label translation. These flags are viewable and modifiable without restriction.
Label translation flags	These fifteen-bit flags support the GFI <code>FLAGS=</code> option in the <code>label_encodings(4)</code> file. Only a trusted path process can view or modify these flags.
Part of diskless boot flag	This one-bit flag identifies the process as taking part in diskless booting. This flag can be viewed and cleared, but cannot be set except by a trusted path process.
Part of selection agent flag	This one-bit flag identifies the process as part of the “cut and paste” selection agent. This flag can be viewed and cleared, but cannot be set except by a trusted path process.
Part of trusted printing system flag	<p>This one-bit flag identifies the process as a member of the Trusted Printing System. This flag can be viewed and cleared, but cannot be set except by a trusted path process.</p> <p>In short, these flag-related protection policies apply. Any process may view or clear any process attribute flag except the label translation flags; viewing or clearing the label translation flags requires that a process have the trusted path attribute. Any process may set label view flags; setting other flags requires that the setting process have the trusted path attribute.</p> <p><code>getpattr()</code> copies the <i>type</i> process flag of the calling process into the <i>pattr_flag_t</i> variable addressed by <i>value</i>. Only the lower <i>n</i> bits are copied, where <i>n</i> is the width of the flag. The higher bits are cleared.</p> <p><code>setpattr()</code> copies the lower <i>n</i> bits of <i>value</i> to the <i>type</i> process flag of the calling process, where <i>n</i> is the width of the selected process flag.</p>
RETURN VALUES	<p><code>getpattr()</code> and <code>setpattr()</code> return:</p> <p>0 On success.</p> <p>-1 On failure, and sets <code>errno</code> to indicate the error.</p>
ERRORS	<p><code>getpattr()</code> may fail for one of these reasons:</p> <p><code>EFAULT</code> The <i>value</i> argument points to a bad address.</p> <p><code>EINVAL</code> The <i>type</i> argument is not one of the listed type constants.</p> <p><code>EACCES</code> The calling process is not a trusted path process as required to view the <i>type</i> flag.</p> <p><code>setpattr()</code> may fail for one of these reasons:</p> <p><code>EFAULT</code> The <i>value</i> argument points to a bad address.</p> <p><code>EINVAL</code> The <i>type</i> argument is not one of the listed type constants.</p> <p><code>EACCES</code> The calling process is not a trusted path process as required to modify the <i>type</i> flag.</p>

getpattr(2)

**Trusted Solaris 8
4/01 Reference
Manual**

pattr(1)

	getpgid(2)						
NAME	getpid, getpgrp, getppid, getpgid – Get process, process group, and parent process IDs						
SYNOPSIS	<pre>#include <unistd.h> pid_t getpid(void); pid_t getpgrp(void); pid_t getppid(void); pid_t getpgid(pid_t pid);</pre>						
DESCRIPTION	<p>The <code>getpid()</code> function returns the process ID of the calling process.</p> <p>The <code>getpgrp()</code> function returns the process group ID of the calling process.</p> <p>The <code>getppid()</code> function returns the parent process ID of the calling process.</p> <p>The <code>getpgid()</code> function returns the process group ID of the process whose process ID is equal to <i>pid</i>, or the process group ID of the calling process, if <i>pid</i> is equal to 0.</p>						
RETURN VALUES	Upon successful completion, these functions return the process group ID. Otherwise, <code>getpgid()</code> returns <code>(pid_t)-1</code> and sets <code>errno</code> to indicate the error.						
ERRORS	<p>The <code>getpgid()</code> function will fail if:</p> <table> <tr> <td>EPERM</td><td>The process whose process ID is equal to <i>pid</i> is not in the same session as the calling process, and the implementation does not allow access to the process group ID of that process from the calling process.</td></tr> <tr> <td>ESRCH</td><td>There is no process with a process ID equal to <i>pid</i>. Or, the calling process does not have MAC read access to the target process, and does not have <code>PRIV_PROC_MAC_READ</code> overriding privilege. Or, the calling process' real or effective user ID does not match the real or saved user ID of the target process, and does not have <code>PRIV_PROC_OWNER</code> overriding privilege.</td></tr> </table> <p>The <code>getpgid()</code> function may fail if:</p> <table> <tr> <td>EINVAL</td><td>The value of the <i>pid</i> argument is invalid.</td></tr> </table>	EPERM	The process whose process ID is equal to <i>pid</i> is not in the same session as the calling process, and the implementation does not allow access to the process group ID of that process from the calling process.	ESRCH	There is no process with a process ID equal to <i>pid</i> . Or, the calling process does not have MAC read access to the target process, and does not have <code>PRIV_PROC_MAC_READ</code> overriding privilege. Or, the calling process' real or effective user ID does not match the real or saved user ID of the target process, and does not have <code>PRIV_PROC_OWNER</code> overriding privilege.	EINVAL	The value of the <i>pid</i> argument is invalid.
EPERM	The process whose process ID is equal to <i>pid</i> is not in the same session as the calling process, and the implementation does not allow access to the process group ID of that process from the calling process.						
ESRCH	There is no process with a process ID equal to <i>pid</i> . Or, the calling process does not have MAC read access to the target process, and does not have <code>PRIV_PROC_MAC_READ</code> overriding privilege. Or, the calling process' real or effective user ID does not match the real or saved user ID of the target process, and does not have <code>PRIV_PROC_OWNER</code> overriding privilege.						
EINVAL	The value of the <i>pid</i> argument is invalid.						
ATTRIBUTES	<p>See <code>attributes(5)</code> for descriptions of the following attributes:</p> <table> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> <tr> <td>MT-Level</td><td>Async-Signal-Safe</td></tr> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	Async-Signal-Safe		
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
MT-Level	Async-Signal-Safe						
SUMMARY OF TRUSTED SOLARIS CHANGES	MAC and DAC policies are added to the <code>getpgid()</code> command. To avoid covert channel issues, the Trusted Solaris environment does not distinguish between failures due to policy and those due to nonexistence of the target process.						

getpgid(2)

Trusted Solaris 8
4/01 Reference
Manual

exec(2), fork(2), getsid(2), intro(3)

setpgid(2), setpgrp(2), setsid(2), signal(3C), attributes(5)

NAME	getpid, getpgrp, getppid, getpgid – Get process, process group, and parent process IDs						
SYNOPSIS	<pre>#include <unistd.h> pid_t getpid(void); pid_t getpgrp(void); pid_t getppid(void); pid_t getpgid(pid_t pid);</pre>						
DESCRIPTION	<p>The <code>getpid()</code> function returns the process ID of the calling process.</p> <p>The <code>getpgrp()</code> function returns the process group ID of the calling process.</p> <p>The <code>getppid()</code> function returns the parent process ID of the calling process.</p> <p>The <code>getpgid()</code> function returns the process group ID of the process whose process ID is equal to <i>pid</i>, or the process group ID of the calling process, if <i>pid</i> is equal to 0.</p>						
RETURN VALUES	Upon successful completion, these functions return the process group ID. Otherwise, <code>getpgid()</code> returns <code>(pid_t)-1</code> and sets <code>errno</code> to indicate the error.						
ERRORS	<p>The <code>getpgid()</code> function will fail if:</p> <table> <tr> <td>EPERM</td><td>The process whose process ID is equal to <i>pid</i> is not in the same session as the calling process, and the implementation does not allow access to the process group ID of that process from the calling process.</td></tr> <tr> <td>ESRCH</td><td>There is no process with a process ID equal to <i>pid</i>. Or, the calling process does not have MAC read access to the target process, and does not have <code>PRIV_PROC_MAC_READ</code> overriding privilege. Or, the calling process' real or effective user ID does not match the real or saved user ID of the target process, and does not have <code>PRIV_PROC_OWNER</code> overriding privilege.</td></tr> </table> <p>The <code>getpgid()</code> function may fail if:</p> <table> <tr> <td>EINVAL</td><td>The value of the <i>pid</i> argument is invalid.</td></tr> </table>	EPERM	The process whose process ID is equal to <i>pid</i> is not in the same session as the calling process, and the implementation does not allow access to the process group ID of that process from the calling process.	ESRCH	There is no process with a process ID equal to <i>pid</i> . Or, the calling process does not have MAC read access to the target process, and does not have <code>PRIV_PROC_MAC_READ</code> overriding privilege. Or, the calling process' real or effective user ID does not match the real or saved user ID of the target process, and does not have <code>PRIV_PROC_OWNER</code> overriding privilege.	EINVAL	The value of the <i>pid</i> argument is invalid.
EPERM	The process whose process ID is equal to <i>pid</i> is not in the same session as the calling process, and the implementation does not allow access to the process group ID of that process from the calling process.						
ESRCH	There is no process with a process ID equal to <i>pid</i> . Or, the calling process does not have MAC read access to the target process, and does not have <code>PRIV_PROC_MAC_READ</code> overriding privilege. Or, the calling process' real or effective user ID does not match the real or saved user ID of the target process, and does not have <code>PRIV_PROC_OWNER</code> overriding privilege.						
EINVAL	The value of the <i>pid</i> argument is invalid.						
ATTRIBUTES	<p>See <code>attributes(5)</code> for descriptions of the following attributes:</p> <table> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> <tr> <td>MT-Level</td><td>Async-Signal-Safe</td></tr> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	Async-Signal-Safe		
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
MT-Level	Async-Signal-Safe						
SUMMARY OF TRUSTED SOLARIS CHANGES	MAC and DAC policies are added to the <code>getpgid()</code> command. To avoid covert channel issues, the Trusted Solaris environment does not distinguish between failures due to policy and those due to nonexistence of the target process.						

getpgrp(2)

Trusted Solaris 8
4/01 Reference
Manual

exec(2), fork(2), getsid(2), intro(3)

setpgid(2), setpgrp(2), setsid(2), signal(3C), attributes(5)

NAME	getpid, getpgrp, getppid, getpgid – Get process, process group, and parent process IDs						
SYNOPSIS	<pre>#include <unistd.h> pid_t getpid(void); pid_t getpgrp(void); pid_t getppid(void); pid_t getpgid(pid_t pid);</pre>						
DESCRIPTION	<p>The <code>getpid()</code> function returns the process ID of the calling process.</p> <p>The <code>getpgrp()</code> function returns the process group ID of the calling process.</p> <p>The <code>getppid()</code> function returns the parent process ID of the calling process.</p> <p>The <code>getpgid()</code> function returns the process group ID of the process whose process ID is equal to <i>pid</i>, or the process group ID of the calling process, if <i>pid</i> is equal to 0.</p>						
RETURN VALUES	Upon successful completion, these functions return the process group ID. Otherwise, <code>getpgid()</code> returns <code>(pid_t)-1</code> and sets <code>errno</code> to indicate the error.						
ERRORS	<p>The <code>getpgid()</code> function will fail if:</p> <table> <tr> <td>EPERM</td><td>The process whose process ID is equal to <i>pid</i> is not in the same session as the calling process, and the implementation does not allow access to the process group ID of that process from the calling process.</td></tr> <tr> <td>ESRCH</td><td>There is no process with a process ID equal to <i>pid</i>. Or, the calling process does not have MAC read access to the target process, and does not have <code>PRIV_PROC_MAC_READ</code> overriding privilege. Or, the calling process' real or effective user ID does not match the real or saved user ID of the target process, and does not have <code>PRIV_PROC_OWNER</code> overriding privilege.</td></tr> </table> <p>The <code>getpgid()</code> function may fail if:</p> <table> <tr> <td>EINVAL</td><td>The value of the <i>pid</i> argument is invalid.</td></tr> </table>	EPERM	The process whose process ID is equal to <i>pid</i> is not in the same session as the calling process, and the implementation does not allow access to the process group ID of that process from the calling process.	ESRCH	There is no process with a process ID equal to <i>pid</i> . Or, the calling process does not have MAC read access to the target process, and does not have <code>PRIV_PROC_MAC_READ</code> overriding privilege. Or, the calling process' real or effective user ID does not match the real or saved user ID of the target process, and does not have <code>PRIV_PROC_OWNER</code> overriding privilege.	EINVAL	The value of the <i>pid</i> argument is invalid.
EPERM	The process whose process ID is equal to <i>pid</i> is not in the same session as the calling process, and the implementation does not allow access to the process group ID of that process from the calling process.						
ESRCH	There is no process with a process ID equal to <i>pid</i> . Or, the calling process does not have MAC read access to the target process, and does not have <code>PRIV_PROC_MAC_READ</code> overriding privilege. Or, the calling process' real or effective user ID does not match the real or saved user ID of the target process, and does not have <code>PRIV_PROC_OWNER</code> overriding privilege.						
EINVAL	The value of the <i>pid</i> argument is invalid.						
ATTRIBUTES	<p>See <code>attributes(5)</code> for descriptions of the following attributes:</p> <table> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> <tr> <td>MT-Level</td><td>Async-Signal-Safe</td></tr> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	Async-Signal-Safe		
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
MT-Level	Async-Signal-Safe						
SUMMARY OF TRUSTED SOLARIS CHANGES	MAC and DAC policies are added to the <code>getpgid()</code> command. To avoid covert channel issues, the Trusted Solaris environment does not distinguish between failures due to policy and those due to nonexistence of the target process.						

getpid(2)

Trusted Solaris 8
4/01 Reference
Manual
Reference Manual

exec(2), fork(2), getsid(2), intro(3)

setpgid(2), setpgrp(2), setsid(2), signal(3C), attributes(5)

	getppid(2)						
NAME	getpid, getpgrp, getppid, getpgid – Get process, process group, and parent process IDs						
SYNOPSIS	<pre>#include <unistd.h> pid_t getpid(void); pid_t getpgrp(void); pid_t getppid(void); pid_t getpgid(pid_t pid);</pre>						
DESCRIPTION	<p>The <code>getpid()</code> function returns the process ID of the calling process.</p> <p>The <code>getpgrp()</code> function returns the process group ID of the calling process.</p> <p>The <code>getppid()</code> function returns the parent process ID of the calling process.</p> <p>The <code>getpgid()</code> function returns the process group ID of the process whose process ID is equal to <i>pid</i>, or the process group ID of the calling process, if <i>pid</i> is equal to 0.</p>						
RETURN VALUES	Upon successful completion, these functions return the process group ID. Otherwise, <code>getpgid()</code> returns <code>(pid_t)-1</code> and sets <code>errno</code> to indicate the error.						
ERRORS	<p>The <code>getpgid()</code> function will fail if:</p> <table> <tr> <td>EPERM</td><td>The process whose process ID is equal to <i>pid</i> is not in the same session as the calling process, and the implementation does not allow access to the process group ID of that process from the calling process.</td></tr> <tr> <td>ESRCH</td><td>There is no process with a process ID equal to <i>pid</i>. Or, the calling process does not have MAC read access to the target process, and does not have <code>PRIV_PROC_MAC_READ</code> overriding privilege. Or, the calling process' real or effective user ID does not match the real or saved user ID of the target process, and does not have <code>PRIV_PROC_OWNER</code> overriding privilege.</td></tr> </table> <p>The <code>getpgid()</code> function may fail if:</p> <table> <tr> <td>EINVAL</td><td>The value of the <i>pid</i> argument is invalid.</td></tr> </table>	EPERM	The process whose process ID is equal to <i>pid</i> is not in the same session as the calling process, and the implementation does not allow access to the process group ID of that process from the calling process.	ESRCH	There is no process with a process ID equal to <i>pid</i> . Or, the calling process does not have MAC read access to the target process, and does not have <code>PRIV_PROC_MAC_READ</code> overriding privilege. Or, the calling process' real or effective user ID does not match the real or saved user ID of the target process, and does not have <code>PRIV_PROC_OWNER</code> overriding privilege.	EINVAL	The value of the <i>pid</i> argument is invalid.
EPERM	The process whose process ID is equal to <i>pid</i> is not in the same session as the calling process, and the implementation does not allow access to the process group ID of that process from the calling process.						
ESRCH	There is no process with a process ID equal to <i>pid</i> . Or, the calling process does not have MAC read access to the target process, and does not have <code>PRIV_PROC_MAC_READ</code> overriding privilege. Or, the calling process' real or effective user ID does not match the real or saved user ID of the target process, and does not have <code>PRIV_PROC_OWNER</code> overriding privilege.						
EINVAL	The value of the <i>pid</i> argument is invalid.						
ATTRIBUTES	<p>See <code>attributes(5)</code> for descriptions of the following attributes:</p> <table> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> <tr> <td>MT-Level</td><td>Async-Signal-Safe</td></tr> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	Async-Signal-Safe		
ATTRIBUTE TYPE	ATTRIBUTE VALUE						
MT-Level	Async-Signal-Safe						
SUMMARY OF TRUSTED SOLARIS CHANGES	MAC and DAC policies are added to the <code>getpgid()</code> command. To avoid covert channel issues, the Trusted Solaris environment does not distinguish between failures due to policy and those due to nonexistence of the target process.						

getppid(2)

Trusted Solaris 8
4/01 Reference
Manual
Reference Manual

exec(2), fork(2), getsid(2), intro(3)

setpgid(2), setpgrp(2), setsid(2), signal(3C), attributes(5)

NAME	getppriv, setppriv – Return or assign a privilege set associated with the invoking process														
SYNOPSIS	<pre>cc [flags...] file... -ltsol #include <tsol/priv.h> int getppriv(priv_ptype_ttype, priv_set_t*pset); int setppriv(priv_op_top, priv_ptype_ttype, priv_set_t*pset);</pre>														
DESCRIPTION	<p>getppriv() copies the <i>type</i> privilege set of the invoking process into the <i>pset</i> address. <i>type</i> may have one of four values, specified in <tsol/priv.h>:</p> <table> <tr> <td>PRIV_EFFECTIVE</td><td>The effective privilege set</td></tr> <tr> <td>PRIV_INHERITABLE</td><td>The inheritable privilege set</td></tr> <tr> <td>PRIV_PERMITTED</td><td>The permitted privilege set</td></tr> <tr> <td>PRIV_SAVED</td><td>The saved privilege set</td></tr> </table> <p>setppriv() assigns or modifies the <i>type</i> privilege set (the target set) of the invoking process. Modification occurs according to the values of <i>op</i> and of the <i>pset</i> privilege set (the source set). <i>op</i> values are specified in <tsol/priv.h>:</p> <table> <tr> <td>PRIV_ON</td><td>Each privilege asserted in the source set is asserted in the target set.</td></tr> <tr> <td>PRIV_OFF</td><td>Each privilege asserted in the source set is cleared in the target set.</td></tr> <tr> <td>PRIV_SET</td><td>The target set is made exactly equal to the source set.</td></tr> </table> <p>Values for <i>type</i> are the same as those for <i>type</i> in getppriv(), exclusive of PRIV_SAVED.</p> <p>If the target set is the permitted set, all privileges cleared from the target set are also cleared from the effective set. Any attempted assignment of a privilege cleared in the permitted set is always an error. Attempting to clear a privilege that is already cleared is not an error.</p>	PRIV_EFFECTIVE	The effective privilege set	PRIV_INHERITABLE	The inheritable privilege set	PRIV_PERMITTED	The permitted privilege set	PRIV_SAVED	The saved privilege set	PRIV_ON	Each privilege asserted in the source set is asserted in the target set.	PRIV_OFF	Each privilege asserted in the source set is cleared in the target set.	PRIV_SET	The target set is made exactly equal to the source set.
PRIV_EFFECTIVE	The effective privilege set														
PRIV_INHERITABLE	The inheritable privilege set														
PRIV_PERMITTED	The permitted privilege set														
PRIV_SAVED	The saved privilege set														
PRIV_ON	Each privilege asserted in the source set is asserted in the target set.														
PRIV_OFF	Each privilege asserted in the source set is cleared in the target set.														
PRIV_SET	The target set is made exactly equal to the source set.														
RETURN VALUES	<p>getppriv() and setppriv() return:</p> <table> <tr> <td>0</td><td>On success.</td></tr> <tr> <td>-1</td><td>On failure, and set errno to indicate the error.</td></tr> </table>	0	On success.	-1	On failure, and set errno to indicate the error.										
0	On success.														
-1	On failure, and set errno to indicate the error.														
ERRORS	<p>getppriv() fails if either of these conditions prevails:</p> <table> <tr> <td>EINVAL</td><td>An illegal or undefined value was supplied for <i>type</i>.</td></tr> <tr> <td>EFAULT</td><td><i>pset</i> refers to an invalid address.</td></tr> </table> <p>setppriv() fails and the target set is not modified if any of these conditions prevails:</p> <table> <tr> <td>EINVAL</td><td>An illegal or undefined value is supplied for <i>type</i> or <i>op</i>.</td></tr> </table>	EINVAL	An illegal or undefined value was supplied for <i>type</i> .	EFAULT	<i>pset</i> refers to an invalid address.	EINVAL	An illegal or undefined value is supplied for <i>type</i> or <i>op</i> .								
EINVAL	An illegal or undefined value was supplied for <i>type</i> .														
EFAULT	<i>pset</i> refers to an invalid address.														
EINVAL	An illegal or undefined value is supplied for <i>type</i> or <i>op</i> .														

getppriv(2)

EFAULT	set refers to an invalid address.
EINVAL	In a process privilege set, an attempt is made to assert a privilege that is cleared in the permitted set of the process.

**Trusted Solaris 8
4/01 Reference
Manual**

getfpriv(2), setfpriv(2), priv_to_str(3TSOL), priv_set_to_str(3TSOL),
str_to_priv(3TSOL), str_to_priv_set(3TSOL), priv_macros(5)

NAME	getrlimit, setrlimit – Control maximum system resource consumption								
SYNOPSIS	<pre>#include <sys/resource.h> int getrlimit(int <i>resource</i>, struct rlimit <i>*rlp</i>); int setrlimit(int <i>resource</i>, const struct rlimit <i>*rlp</i>);</pre>								
DESCRIPTION	<p>Limits on the consumption of a variety of system resources by a process and each process it creates may be obtained with the <code>getrlimit()</code> and set with <code>setrlimit()</code> functions.</p> <p>Each call to either <code>getrlimit()</code> or <code>setrlimit()</code> identifies a specific resource to be operated upon as well as a resource limit. A resource limit is a pair of values: one specifying the current (soft) limit, the other a maximum (hard) limit. Soft limits may be changed by a process to any value that is less than or equal to the hard limit. A process may (irreversibly) lower its hard limit to any value that is greater than or equal to the soft limit. Only a process that has the <code>PRIV_SYS_CONFIG</code> privilege can raise a hard limit. Both hard and soft limits can be changed in a single call to <code>setrlimit()</code> subject to the constraints described above. Limits may have an “infinite” value of <code>RLIM_INFINITY</code>. The <i>rlp</i> argument is a pointer to <code>struct rlimit</code> that includes the following members:</p> <pre>rlim_t rlim_cur; /* current (soft) limit */ rlim_t rlim_max; /* hard limit */</pre> <p>The type <code>rlim_t</code> is an arithmetic data type to which objects of type <code>int</code>, <code>size_t</code>, and <code>off_t</code> can be cast without loss of information.</p> <p>The possible resources, their descriptions, and the actions taken when the current limit is exceeded are summarized as follows:</p> <table> <tr> <td>RLIMIT_CORE</td><td>The maximum size of a core file in bytes that may be created by a process. A limit of 0 will prevent the creation of a core file. The writing of a core file will terminate at this size.</td></tr> <tr> <td>RLIMIT_CPU</td><td>The maximum amount of CPU time in seconds used by a process. This is a soft limit only. The <code>SIGXCPU</code> signal is sent to the process. If the process is holding or ignoring <code>SIGXCPU</code>, the behavior is scheduling class defined.</td></tr> <tr> <td>RLIMIT_DATA</td><td>The maximum size of a process’s heap in bytes. The <code>brk(2)</code> function will fail with <code>errno</code> set to <code>ENOMEM</code>.</td></tr> <tr> <td>RLIMIT_FSIZE</td><td>The maximum size of a file in bytes that may be created by a process. A limit of 0 will prevent the creation of a file. The <code>SIGXFSZ</code> signal is sent to the process. If the process is holding or ignoring <code>SIGXFSZ</code>, continued attempts to increase the size of a file beyond the limit will fail with <code>errno</code> set to <code>EFBIG</code>.</td></tr> </table>	RLIMIT_CORE	The maximum size of a core file in bytes that may be created by a process. A limit of 0 will prevent the creation of a core file. The writing of a core file will terminate at this size.	RLIMIT_CPU	The maximum amount of CPU time in seconds used by a process. This is a soft limit only. The <code>SIGXCPU</code> signal is sent to the process. If the process is holding or ignoring <code>SIGXCPU</code> , the behavior is scheduling class defined.	RLIMIT_DATA	The maximum size of a process’s heap in bytes. The <code>brk(2)</code> function will fail with <code>errno</code> set to <code>ENOMEM</code> .	RLIMIT_FSIZE	The maximum size of a file in bytes that may be created by a process. A limit of 0 will prevent the creation of a file. The <code>SIGXFSZ</code> signal is sent to the process. If the process is holding or ignoring <code>SIGXFSZ</code> , continued attempts to increase the size of a file beyond the limit will fail with <code>errno</code> set to <code>EFBIG</code> .
RLIMIT_CORE	The maximum size of a core file in bytes that may be created by a process. A limit of 0 will prevent the creation of a core file. The writing of a core file will terminate at this size.								
RLIMIT_CPU	The maximum amount of CPU time in seconds used by a process. This is a soft limit only. The <code>SIGXCPU</code> signal is sent to the process. If the process is holding or ignoring <code>SIGXCPU</code> , the behavior is scheduling class defined.								
RLIMIT_DATA	The maximum size of a process’s heap in bytes. The <code>brk(2)</code> function will fail with <code>errno</code> set to <code>ENOMEM</code> .								
RLIMIT_FSIZE	The maximum size of a file in bytes that may be created by a process. A limit of 0 will prevent the creation of a file. The <code>SIGXFSZ</code> signal is sent to the process. If the process is holding or ignoring <code>SIGXFSZ</code> , continued attempts to increase the size of a file beyond the limit will fail with <code>errno</code> set to <code>EFBIG</code> .								

getrlimit(2)

RLIMIT_NOFILE	One more than the maximum value that the system may assign to a newly created descriptor. This limit constrains the number of file descriptors that a process may create.
RLIMIT_STACK	<p>The maximum size of a process's stack in bytes. The system will not automatically grow the stack beyond this limit.</p> <p>Within a process, <code>setrlimit()</code> will increase the limit on the size of your stack, but will not move current memory segments to allow for that growth. To guarantee that the process stack can grow to the limit, the limit must be altered prior to the execution of the process in which the new stack size is to be used.</p> <p>Within a multithreaded process, <code>setrlimit()</code> has no impact on the stack size limit for the calling thread if the calling thread is not the main thread. A call to <code>setrlimit()</code> for <code>RLIMIT_STACK</code> impacts only the main thread's stack, and should be made only from the main thread, if at all.</p> <p>The <code>SIGSEGV</code> signal is sent to the process. If the process is holding or ignoring <code>SIGSEGV</code>, or is catching <code>SIGSEGV</code> and has not made arrangements to use an alternate stack (see <code>sigaltstack(2)</code>), the disposition of <code>SIGSEGV</code> will be set to <code>SIG_DFL</code> before it is sent.</p>
RLIMIT_VMEM	The maximum size of a process's mapped address space in bytes. If this limit is exceeded, the <code>brk(2)</code> and <code>mmap(2)</code> functions will fail with <code>errno</code> set to <code>ENOMEM</code> . In addition, the automatic stack growth will fail with the effects outlined above.
RLIMIT_AS	This is the maximum size of a process's total available memory, in bytes. If this limit is exceeded, the <code>brk(2)</code> , <code>malloc(3C)</code> , <code>mmap(2)</code> and <code>sbrk(2)</code> functions will fail with <code>errno</code> set to <code>ENOMEM</code> . In addition, the automatic stack growth will fail with the effects outlined above.

Because limit information is stored in the per-process information, the shell builtin `ulimit` command must directly execute this system call if it is to affect all future processes created by the shell.

The value of the current limit of the following resources affect these implementation defined parameters:

Limit	Implementation Defined Constant
RLIMIT_FSIZE	FCHR_MAX
RLIMIT_NOFILE	OPEN_MAX

When using the `getrlimit()` function, if a resource limit can be represented correctly in an object of type `rlim_t`, then its representation is returned; otherwise, if the value of the resource limit is equal to that of the corresponding saved hard limit, the value returned is `RLIM_SAVED_MAX`; otherwise the value returned is `RLIM_SAVED_CUR`.

When using the `setrlimit()` function, if the requested new limit is `RLIM_INFINITY`, the new limit will be "no limit"; otherwise if the requested new limit is `RLIM_SAVED_MAX`, the new limit will be the corresponding saved hard limit; otherwise, if the requested new limit is `RLIM_SAVED_CUR`, the new limit will be the corresponding saved soft limit; otherwise, the new limit will be the requested value. In addition, if the corresponding saved limit can be represented correctly in an object of type `rlim_t`, then it will be overwritten with the new limit.

The result of setting a limit to `RLIM_SAVED_MAX` or `RLIM_SAVED_CUR` is unspecified unless a previous call to `getrlimit()` returned that value as the soft or hard limit for the corresponding resource limit.

A limit whose value is greater than `RLIM_INFINITY` is permitted.

The `exec` family of functions also cause resource limits to be saved. See `exec(2)`.

RETURN VALUES

`getrlimit()` and `setrlimit()` return:

- 0 On success.
- 1 On failure, and set `errno` to indicate the error.

ERRORS

The `getrlimit()` and `setrlimit()` functions will fail if:

- `EFAULT` The *rlp* argument points to an illegal address.
- `EINVAL` An invalid *resource* was specified; or in a `setrlimit()` call, the new `rlim_cur` exceeds the new `rlim_max`.
- `EPERM` The limit specified to `setrlimit()` would have raised the maximum limit value, and the calling process does not have the `PRIV_SYS_CONFIG` privilege.

The `setrlimit()` function may fail if:

- `EINVAL` The limit specified cannot be lowered because current usage is already higher than the limit.

USAGE

The `getrlimit()` and `setrlimit()` functions have transitional interfaces for 64-bit file offsets. See `lf64(5)`.

SUMMARY OF TRUSTED SOLARIS CHANGES Trusted Solaris 8 4/01 Reference Manual

The calling process must have the `PRIV_SYS_CONFIG` privilege in order to increase a hard resource limit.

`open(2)`

getrlimit(2)

**SunOS 5.8
Reference Manual**

brk(2), sigaltstack(2), malloc(3C), signal(3C), signal(5)

NAME	getmsgqcmwlabel, getshmcmwlabel, getsemcmwlabel – Get the CMW labels associated with System V IPC structures
SYNOPSIS	<pre>cc [flags...] file ... -ltsol [library...] #include <sys/tsol/ipcl.h> int getmsgqcmwlabel(int msgqid, bclabel_t *clabel); int getshmcmwlabel(int shmid, bclabel_t *clabel); int getsemcmwlabel(int semid, bclabel_t *clabel);</pre>
DESCRIPTION	<p>These functions return the value of the CMW labels associated with message queues, shared memory, and semaphore structures.</p> <p>getmsgqcmwlabel() returns the CMW label for the message queue identified by <i>msgqid</i> into the label buffer to which <i>clabel</i> points. The information label portion of the CMW label is undefined for message queues; therefore the sensitivity label portion may have to be extracted using getcsl(3TSOL) in order to be useful.</p> <p>getshmcmwlabel() returns the CMW label for the shared-memory segment identified by <i>shmid</i> into the label buffer to which <i>clabel</i> points.</p> <p>getsemcmwlabel() returns the CMW label for the semaphore array identified by <i>semid</i> into the label buffer to which <i>clabel</i> points.</p> <p>The calling process must have mandatory read access to the IPC or must have asserted the PRIV_IPC_MAC_READ privilege, and must have discretionary read access to the data structure or must have the PRIV_IPC_DAC_READ privilege in its set of effective privileges.</p>
RETURN VALUES	<p>getmsgqcmwlabel(), getshmcmwlabel(), and getsemcmwlabel() return:</p> <p>0 On success.</p> <p>-1 On failure, and sets errno to indicate the error.</p>
ERRORS	<p>These functions will fail if any of these conditions is true:</p> <p>EACCES Read access is denied to the calling process, which does not have one or both of these privileges in its set of effective privileges: PRIV_IPC_DAC_READ and PRIV_IPC_MAC_READ.</p> <p>EINVAL <i>msgqid</i>, <i>semid</i>, or <i>shmid</i> is not a valid IPC object identifier.</p> <p>EFAULT <i>clabel</i> points to an illegal address.</p>
Trusted Solaris 8 4/01 Reference Manual	msgget(2), semget(2), shmget(2)

getshmcmwlabel(2)

NAME	getmsgqcmwlabel, getshmcmwlabel, getsemcmwlabel – Get the CMW labels associated with System V IPC structures						
SYNOPSIS	<pre>cc [flags...] file ... -ltsol [library...] #include <sys/tsol/ipcl.h> int getmsgqcmwlabel(int msgqid, bclabel_t *clabel); int getshmcmwlabel(int shmid, bclabel_t *clabel); int getsemcmwlabel(int semid, bclabel_t *clabel);</pre>						
DESCRIPTION	<p>These functions return the value of the CMW labels associated with message queues, shared memory, and semaphore structures.</p> <p>getmsgqcmwlabel() returns the CMW label for the message queue identified by <i>msgqid</i> into the label buffer to which <i>clabel</i> points. The information label portion of the CMW label is undefined for message queues; therefore the sensitivity label portion may have to be extracted using getcsl(3TSOL) in order to be useful.</p> <p>getshmcmwlabel() returns the CMW label for the shared-memory segment identified by <i>shmid</i> into the label buffer to which <i>clabel</i> points.</p> <p>getsemcmwlabel() returns the CMW label for the semaphore array identified by <i>semid</i> into the label buffer to which <i>clabel</i> points.</p> <p>The calling process must have mandatory read access to the IPC or must have asserted the PRIV_IPC_MAC_READ privilege, and must have discretionary read access to the data structure or must have the PRIV_IPC_DAC_READ privilege in its set of effective privileges.</p>						
RETURN VALUES	<p>getmsgqcmwlabel(), getshmcmwlabel(), and getsemcmwlabel() return:</p> <p>0 On success.</p> <p>-1 On failure, and sets errno to indicate the error.</p>						
ERRORS	<p>These functions will fail if any of these conditions is true:</p> <table> <tr> <td>EACCES</td><td>Read access is denied to the calling process, which does not have one or both of these privileges in its set of effective privileges: PRIV_IPC_DAC_READ and PRIV_IPC_MAC_READ.</td></tr> <tr> <td>EINVAL</td><td><i>msgqid</i>, <i>semid</i>, or <i>shmid</i> is not a valid IPC object identifier.</td></tr> <tr> <td>EFAULT</td><td><i>clabel</i> points to an illegal address.</td></tr> </table>	EACCES	Read access is denied to the calling process, which does not have one or both of these privileges in its set of effective privileges: PRIV_IPC_DAC_READ and PRIV_IPC_MAC_READ.	EINVAL	<i>msgqid</i> , <i>semid</i> , or <i>shmid</i> is not a valid IPC object identifier.	EFAULT	<i>clabel</i> points to an illegal address.
EACCES	Read access is denied to the calling process, which does not have one or both of these privileges in its set of effective privileges: PRIV_IPC_DAC_READ and PRIV_IPC_MAC_READ.						
EINVAL	<i>msgqid</i> , <i>semid</i> , or <i>shmid</i> is not a valid IPC object identifier.						
EFAULT	<i>clabel</i> points to an illegal address.						
Trusted Solaris 8 4/01 Reference Manual	msgget(2), semget(2), shmget(2)						

NAME	getsid – Get process group ID of session leader				
SYNOPSIS	<pre>#include <unistd.h> pid_t getsid(pid_t pid) ;</pre>				
DESCRIPTION	<p>The function <code>getsid()</code> returns the session ID of the process whose process ID is equal to <i>pid</i>. If <i>pid</i> is equal to <code>(pid_t) 0</code>, <code>getsid()</code> returns the session ID of the calling process. The calling process must have MAC read access to the target process. The calling process' real or effective user ID must match the real or saved user ID of the target process.</p> <p>If the calling process is not already a process group leader, <code>setsid()</code> sets the process group ID and session ID of the calling process to the process ID of the calling process, and releases the process's controlling terminal.</p> <p>See <code>intro(2)</code> for more information on process groups and controlling terminals.</p>				
RETURN VALUES	Upon successful completion, <code>getsid()</code> returns the process group ID of the session leader of the specified process. Otherwise, it returns <code>(pid_t) -1</code> and sets <code>errno</code> to indicate the error.				
ERRORS	<p>The <code>getsid()</code> function will fail if:</p> <table> <tr> <td>EPERM</td><td>The process specified by <i>pid</i> is not in the same session as the calling process, and the implementation does not allow access to the process group ID of the session leader of that process from the calling process.</td></tr> <tr> <td>ESRCH</td><td>There is no process with a process ID equal to <i>pid</i>. Or, the calling process does not have MAC read access to the target process, and does not have <code>PRIV_PROC_MAC_READ</code> overriding privilege. Or, the calling process' real or effective user ID does not match the real or saved user ID of the target process, and does not have <code>PRIV_PROC_OWNER</code> overriding privilege.</td></tr> </table>	EPERM	The process specified by <i>pid</i> is not in the same session as the calling process, and the implementation does not allow access to the process group ID of the session leader of that process from the calling process.	ESRCH	There is no process with a process ID equal to <i>pid</i> . Or, the calling process does not have MAC read access to the target process, and does not have <code>PRIV_PROC_MAC_READ</code> overriding privilege. Or, the calling process' real or effective user ID does not match the real or saved user ID of the target process, and does not have <code>PRIV_PROC_OWNER</code> overriding privilege.
EPERM	The process specified by <i>pid</i> is not in the same session as the calling process, and the implementation does not allow access to the process group ID of the session leader of that process from the calling process.				
ESRCH	There is no process with a process ID equal to <i>pid</i> . Or, the calling process does not have MAC read access to the target process, and does not have <code>PRIV_PROC_MAC_READ</code> overriding privilege. Or, the calling process' real or effective user ID does not match the real or saved user ID of the target process, and does not have <code>PRIV_PROC_OWNER</code> overriding privilege.				
SUMMARY OF TRUSTED SQUARES CHANGES	<p>MAC and DAC policies are added to the <code>getsid()</code> system call.</p> <p><code>intro(2)</code>, <code>exec(2)</code>, <code>fork(2)</code>, <code>getpid(2)</code>, <code>setpgid(2)</code></p>				

Trusted Squares
4/01 Reference
Manual

getslldname(2)

NAME	getslldname, fgetslldname – Get file system single-level directory name				
SYNOPSIS	<pre>cc [flags...] file... -ltsol #include <tsol/label.h> int getslldname(char *path_name, bslabel_t *slabel_p, char *name_buf, const int length); int fgetslldname(int fd, const bslabel_t *slabel_p, char *name_buf, const int length);</pre>				
DESCRIPTION	<p>getslldname() returns the SLD name associated with the sensitivity label to which <i>slabel_p</i> refers within the context of the file system on which <i>path_name</i> resides. <i>path_name</i> is the path name of any multilevel directory within the mounted filesystem. <i>name_buf</i> is a pointer to a buffer of at least SLD_NAME_MAX bytes.</p> <p>fgetslldname() returns the SLD name associated with the sensitivity label to which <i>slabel_p</i> refers if the MLD to which descriptor <i>fd</i> refers was opened by the directory name (not by the fully adorned, multilevel directory name.) If the MLD to which descriptor <i>fd</i> refers was opened using the fully adorned, multilevel directory name, fgetslldname() returns the MLD and the SLD name associated with the sensitivity label to which <i>slabel_p</i> refers.</p> <p>If it does not exist, the single-level directory that corresponds to <i>slabel_p</i> is created with the attributes of the parent multilevel directory, the specified sensitivity label, and an ADMIN_LOW information label. If the sensitivity label of the calling process is equal to <i>slabel_p</i>, no additional privileges are needed. If the sensitivity label of the calling process is strictly dominated by <i>slabel_p</i>, the calling process may assert the PRIV_FILE_UPGRADE_SL privilege to create the directory. Otherwise, the calling process may assert the PRIV_FILE_DOWNGRADE_SL privilege to create the directory.</p>				
ATTRIBUTES	<p>See for descriptions of the following attributes:</p> <table border="1"> <thead> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> </thead> <tbody> <tr> <td>Availability</td><td>SUNWtsu</td></tr> </tbody> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
Availability	SUNWtsu				
RETURN VALUES	<p>getslldname() and fgetslldname() return:</p> <p>0 On success.</p> <p>-1 On failure and set errno to indicate the error.</p>				
ERRORS	<p>getslldname() fails if any of these conditions is true:</p> <p>EACCES Search permission is denied for a component of the path prefix of <i>path_name</i>. To override this restriction, the calling process may assert one or both of these privileges: PRIV_FILE_DAC_SEARCH and PRIV_FILE_MAC_SEARCH.</p>				

	The single-level directory specified does not exist, the system is configured to require write access to create a single-level directory, and the calling process does not have discretionary write access to <i>path_name</i> . To override this restriction, the calling process may assert the PRIV_FILE_DAC_WRITE privilege.
EFAULT	<i>name_buf</i> , <i>path_name</i> , or <i>slabel_p</i> points to an invalid address.
EIO	An I/O error occurred while reading from or writing to the file system
ELOOP	Too many symbolic links were encountered in translating <i>path_name</i> .
ENAMETOOLONG	The length of the path argument exceeds PATH_MAX. A pathname component is longer than NAME_MAX [see sysconf(3C)] while _POSIX_NO_TRUNC is in effect. [See pathconf(2).]
ENOENT	The file to which <i>path_name</i> refers does not exist.
ENOTDIR	A component of the path prefix of <i>path_name</i> is not a directory.
EPERM	The SLD that corresponds to <i>slabel_p</i> does not exist and one of these conditions is true: the sensitivity label of the calling process is strictly dominated by <i>slabel_p</i> and the calling process has not asserted the PRIV_FILE_DOWNGRADE privilege; the sensitivity label of the calling process is not dominated by <i>slabel_p</i> and the calling process has not asserted the PRIV_FILE_DOWNGRADE_SL privilege.
fgetsldname() fails if any of these conditions is true:	
EBADF	<i>fd</i> is not a valid open file descriptor.
EFAULT	<i>name_buf</i> or <i>slabel_p</i> points to an invalid address.
EINVAL	<i>fd</i> does not refer to a multilevel directory.
EIO	An I/O error occurred while reading from the file system.
EPERM	The SLD that corresponds to <i>slabel_p</i> does not exist and one of these conditions is true: the sensitivity label of the calling process is strictly dominated by <i>slabel_p</i> and the calling process has not asserted the PRIV_FILE_UPGRADE_SL privilege; the sensitivity label of the calling process is not dominated by <i>slabel_p</i> and the calling process has not asserted the PRIV_FILE_DOWNGRADE_SL privilege.

WARNINGS If the file system that contains *path_name* or the object referred to by *fd* does not support MLDs, no error is returned and the first SLD_NAME_MAX bytes in the *name_buf* are cleared.

getsldname(2)

Trusted Solaris 8
4/01 Reference
SunOS 5.8
Reference Manual

fgetmldadorn(2), getmldadorn(2)

sysconf(3C)

NAME	kill – Send a signal to a process or a group of processes
SYNOPSIS	<pre>#include <sys/types.h> #include <signal.h> int kill(pid_t pid, int sig);</pre>
DESCRIPTION	<p>kill() sends a signal to a process or a group of processes specified by <i>pid</i>. The signal that is to be sent, specified by <i>sig</i>, is either one from the list given in signal() (see signal(3HEAD)), or 0. If <i>sig</i> is 0 (the null signal), error checking is performed but no signal is actually sent. This method can be used to check the validity of <i>pid</i>.</p> <p>The sending process must have MAC write access to the receiving processes. The real or effective user ID of the sending process must match the real or saved [from exec(2)] user ID of the receiving process unless the sending process has the PRIV_PROC_OWNER effective privilege, or <i>sig</i> is SIGCONT and the sending process has the same session ID as the receiving process.</p> <p>If <i>pid</i> is greater than 0, <i>sig</i> will be sent to the process whose process ID is equal to <i>pid</i>.</p> <p>If <i>pid</i> is negative but not (pid_t)-1, <i>sig</i> will be sent to all processes whose process group ID is equal to the absolute value of <i>pid</i> and for which the process has permission to send a signal.</p> <p>If <i>pid</i> is 0, <i>sig</i> will be sent to all processes excluding special processes (see intro(2)) whose process group ID is equal to the process group ID of the sender.</p> <p>If <i>pid</i> is (pid_t)-1 and the sender does not have PRIV_PROC_OWNER in its effective privilege set, <i>sig</i> will be sent to all processes excluding special processes whose real user ID is equal to the effective user ID of the sender.</p>
RETURN VALUES	<p>kill() returns:</p> <ul style="list-style-type: none"> 0 On success. -1 On failure, does not send a signal, and sets <code>errno</code> to indicate the error.
ERRORS	<p>The kill() function will fail if:</p> <ul style="list-style-type: none"> EINVAL The <i>sig</i> argument is not a valid signal number. EPERM The calling process failed in MAC write access to the receiving process and does not have PRIV_PROC_MAC_WRITE overriding privilege. <p><i>sig</i> is SIGKILL and <i>pid</i> is (pid_t) 1. (That is, the calling process does not have permission to send the signal to any of the processes specified by <i>pid</i>).</p> <p>The effective user of the calling process does not match the real or saved user and the sending process does not have</p>

kill(2)

SUMMARY OF
TRUSTED
SQUARIS
Trusted Squares
4/01 Reference
Manual

	PRIV_PROC_OWNER privilege, and the calling process is not sending SIGCONT to a process that shares the same session ID.				
ESRCH	No process or process group can be found corresponding to that specified by <i>pid</i> .				
USAGE	The sigsend(2) function provides a more versatile way to send signals to processes.				
ATTRIBUTES	See attributes(5) for descriptions of the following attributes: <table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>MT-Level</td><td>Async-Signal-Safe</td></tr></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	Async-Signal-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	Async-Signal-Safe				
SUMMARY OF TRUSTED SQUARIS	Process MAC write policy and the process owner policy is checked.				
Trusted Squares 4/01 Reference Manual	exec(2), getpid(2), getsid(2), sigsend(2), intro(3)				
	kill(1), setpgrp(2), sigaction(2), signal(3C), signal(3HEAD), attributes(5)				

NAME	chown, lchown, fchown – change owner and group of a file
SYNOPSIS	<pre>#include <unistd.h> #include <sys/types.h> int chown(const char *path, uid_t owner, gid_t group); int lchown(const char *path, uid_t owner, gid_t group); int fchown(int fildes, uid_t owner, gid_t group);</pre>
DESCRIPTION	<p>The <code>chown()</code> function sets the owner ID and group ID of the file specified by <i>path</i> or referenced by the open file descriptor <i>fildes</i> to <i>owner</i> and <i>group</i> respectively. If <i>owner</i> or <i>group</i> is specified as <code>-1</code>, <code>chown()</code> does not change the corresponding ID of the file.</p> <p>The <code>lchown()</code> function sets the owner ID and group ID of the named file in the same manner as <code>chown()</code>, unless the named file is a symbolic link. In this case, <code>lchown()</code> changes the ownership of the symbolic link file itself, while <code>chown()</code> changes the ownership of the file or directory to which the symbolic link refers.</p> <p>If <code>chown()</code>, <code>lchown()</code>, or <code>fchown()</code> is invoked, the set-user-ID and set-group-ID bits of the file mode, <code>chmod(2)</code>, respectively, are cleared. See <code>chmod(2)</code>. To bypass this restriction, the process may assert the <code>PRIV_FILE_SETID</code> privilege.</p> <p>The operating system has a configuration option, <code>_POSIX_CHOWN_RESTRICTED</code>, to restrict ownership changes for the <code>chown()</code>, <code>lchown()</code>, and <code>fchown()</code> functions. When <code>_POSIX_CHOWN_RESTRICTED</code> is not in effect, the effective user ID of the process must match the owner of the file. To override this restriction, the calling process must assert the <code>PRIV_FILE_CHOWN</code> privilege. When <code>_POSIX_CHOWN_RESTRICTED</code> is not in effect, the effective user ID of the process must match the owner of the file or the process must be the super-user to change the ownership of a file. When <code>_POSIX_CHOWN_RESTRICTED</code> is in effect, the <code>chown()</code>, <code>lchown()</code>, and <code>fchown()</code> functions require that the calling process assert the <code>PRIV_FILE_CHOWN</code> privilege to change the user ID of a file. To change the group ID of a file, the process must be the owner of the file and the new group ID must be the group of the process ID or must be in the supplementary group list of the process. To override this restriction, the calling process may assert the <code>PRIV_FILE_CHOWN</code> privilege.</p> <p>set rstchown = 1</p> <p>To disable this option, include the following line in <code>/etc/system</code>:</p> <p>set rstchown = 0</p> <p>See <code>system(4)</code> and <code>fpathconf(2)</code>.</p> <p>Upon successful completion, <code>chown()</code>, <code>fchown()</code> and <code>lchown()</code> mark for update the <code>st_ctime</code> field of the file.</p>

lchown(2)

RETURN VALUES

chown() returns:

- 0 On success.
- 1 On failure, and sets errno to indicate the error.

ERRORS

The chown() and lchown() functions will fail if:

- | | |
|--------------|---|
| EACCES | Search permission is denied on a component of the path prefix of <i>path</i> . To override this restriction, the calling process may assert one or both of these privileges: PRIV_FILE_DAC_SEARCH and PRIV_FILE_MAC_SEARCH.

Write permission is denied on <i>path</i> or <i>filde</i> s. To override this restriction, the calling process may assert the PRIV_FILE_MAC_WRITE privilege. |
| EFAULT | The <i>path</i> argument points to an illegal address. |
| EINTR | A signal was caught during the execution of the chown() or lchown() function. |
| EINVAL | The <i>group</i> or <i>owner</i> argument is out of range. |
| EIO | An I/O error occurred while reading from or writing to the file system. |
| ELOOP | Too many symbolic links were encountered in translating <i>path</i> . |
| ENAMETOOLONG | The length of the <i>path</i> argument exceeds PATH_MAX, or the length of a <i>path</i> component exceeds NAME_MAX while _POSIX_NO_TRUNC is in effect. |
| ENOLINK | The <i>path</i> argument points to a remote machine and the link to that machine is no longer active. |
| ENOENT | Either a component of the path prefix or the file referred to by <i>path</i> does not exist or is a null pathname. |
| ENOTDIR | A component of the path prefix of <i>path</i> is not a directory. |
| EPERM | The effective user ID does not match the owner of the file. If _POSIX_CHOWN_RESTRICTED is set, the calling process must assert the PRIV_FILE_CHOWN privilege. If _POSIX_CHOWN_RESTRICTED is not set, the calling process may assert the PRIV_FILE_CHOWN privilege. |
| EROFS | The named file resides on a read-only file system. |
- The fchown() function will fail if:
- | | |
|-------|---|
| EBADF | The <i>filde</i> s argument is not an open file descriptor. |
|-------|---|

lchown(2)

EIO	An I/O error occurred while reading from or writing to the file system.
EINTR	A signal was caught during execution of the function.
ENOLINK	The <i>fildev</i> argument points to a remote machine and the link to that machine is no longer active.
EINVAL	The <i>group</i> or <i>owner</i> argument is out of range.
EPERM	The effective user ID does not match the owner of the file, or the process is not the super-user and <code>_POSIX_CHOWN_RESTRICTED</code> indicates that such privilege is required.
EROFS	The named file referred to by <i>fildev</i> resides on a read-only file system.

ATTRIBUTES See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	chown () is Async-Signal-Safe

**SUMMARY OF
TRUSTED
SOLARIS
CHANGES**

Appropriate privilege is required to override access checks.

When the ownership of *path* and *fildev* is changed, the set-user-ID and set-group-ID bits are cleared. The calling process may assert the `PRIV_FILE_SETID` privilege to bypass this restriction.

To change the user ID of the file when the calling process does not own the file and `_POSIX_CHOWN_RESTRICTED` is not in effect, the calling process may assert the `PRIV_FILE_CHOWN` privilege.

To change the group ID of the file when the calling process does not own the file, and the new group ID is not in the group ID of the process or in the supplementary group list of the process, and `_POSIX_CHOWN_RESTRICTED` is not in effect, the calling process may assert the `PRIV_FILE_CHOWN` privilege.

**Trusted Solaris 8
4/01 Reference
Manual**

chgrp(1), chown(1), chmod(2)

attributes(5)

lgetcmwlabel(2)

NAME	getcmwlabel, lgetcmwlabel, fgetcmwlabel – get file CMW label
SYNOPSIS	<pre>cc [flags...] file ... -ltsol [library...] #include <tsol/label.h> int getcmwlabel(char *path, bclabel_t *label_p); int lgetcmwlabel(char *path, bclabel_t *label_p); int fgetcmwlabel(int fd, bclabel_t *label_p);</pre>
DESCRIPTION	<p>getcmwlabel() obtains the CMW label of the file named by <i>path</i>. Mandatory read access to the final component of <i>path</i> is required or the calling process must have PRIV_FILE_MAC_READ in its set of effective privileges. Discretionary read, write or execute permission to the final component of <i>path</i> is not required, but all directories in the path prefix of <i>path</i> must be searchable.</p> <p>lgetcmwlabel() is like getcmwlabel() except in the case where the final component of <i>path</i> is a symbolic link, in which case lgetcmwlabel() returns the CMW label of the link, while getcmwlabel() returns the CMW label of the file to which the link refers.</p> <p>fgetcmwlabel() obtains the CMW label of an open file referred to by the argument descriptor, such as would be obtained by an open(2) call. If the descriptor is only open for writing, then mandatory read access to the object is required or the calling process must have PRIV_FILE_MAC_READ in its set of effective privileges.</p> <p><i>label_p</i> is a pointer to an opaque CMW label structure.</p> <p>An exception to the access rules applies in the case of pty pseudo-terminals (/dev/ptyp* and /dev/ttyp*). Normally mandatory read access is required or the calling process must have PRIV_FILE_MAC_READ in its set of effective privileges. If the specified file is a pty device file and the calling process does not have mandatory read access or PRIV_FILE_MAC_READ is not in its set of effective privileges, each function returns success and sets <i>label_p</i> to ADMIN_LOW.</p>
RETURN VALUES	<p>getcmwlabel(), lgetcmwlabel() and fgetcmwlabel() return:</p> <p>0 On success.</p> <p>-1 On failure, and set errno to indicate the error.</p>
ERRORS	<p>getcmwlabel() and lgetcmwlabel() fail if one or more of the following are true:</p> <p>EACCES Search permission is denied for a component of the path prefix of <i>path</i>. To override this restriction, the calling process may assert the PRIV_FILE_DAC_SEARCH privilege and/or the PRIV_FILE_MAC_SEARCH privilege.</p> <p> The calling process does not have mandatory read access to <i>path</i> because the sensitivity label of the calling process does not</p>

lgetcmwlabel(2)

dominate the sensitivity label of the final component of *path* and the calling process does not have `PRIV_FILE_MAC_READ` in its set of effective privileges.

EFAULT	<i>label_p</i> or <i>path</i> points to an invalid address.
EIO	An I/O error occurred while reading from or writing to the file system.
ELOOP	Too many symbolic links were encountered in translating <i>path</i> .
ENAMETOOLONG	The length of the path argument exceeds <code>PATH_MAX</code> . A pathname component is longer than <code>NAME_MAX</code> while <code>_POSIX_NO_TRUNC</code> is in effect (see <code>pathconf(2)</code>).
ENOENT	The file referred to by <i>path</i> does not exist.
ENOTDIR	A component of the path prefix of <i>path</i> is not a directory.
EPERM	The calling process does not have mandatory read access to <i>path</i> because the sensitivity label of <i>path</i> is outside the calling process' clearance and the calling process does not have <code>PRIV_FILE_MAC_READ</code> in its set of effective privileges.
<code>fgetcmwlabel()</code> fails if one or more of the following are true:	
EACCES	The descriptor is only open for writing and the calling process does not have mandatory read access to the object referred to by the descriptor because the sensitivity label of the calling process does not dominate the sensitivity label of the object and the calling process does not have <code>PRIV_FILE_MAC_READ</code> in its set of effective privileges.
EBADF	<i>fd</i> is not a valid open file descriptor.
EFAULT	<i>label_p</i> points to an invalid address.
EIO	An I/O error occurred while reading from or writing to the file system.

`pathconf(2)`, `open(2)`, `setcmwlabel(2)`

link(2)

NAME	link – Link to a file														
SYNOPSIS	<pre>#include <unistd.h> int link(const char *existing, const char *new);</pre>														
DESCRIPTION	<p>The <code>link()</code> function creates a new link (directory entry) for the existing file and increments its link count by one. The <i>existing</i> argument points to a path name naming an existing file. The <i>new</i> argument points to a pathname naming the new directory entry to be created.</p> <p>For creation of hard links, both files must be on the same file system. Both the old and the new link share equal access and rights to the underlying object. A calling process that has asserted the <code>PRIV_SYS_CONFIG</code> privilege may make multiple links to a directory.</p> <p>Upon successful completion, <code>link()</code> marks for update the <code>st_ctime</code> field of the file. Also, the <code>st_ctime</code> and <code>st_mtime</code> fields of the directory that contains the new entry are marked for update.</p>														
RETURN VALUES	Upon successful completion, 0 is returned. Otherwise, -1 is returned, no link is created, and <code>errno</code> is set to indicate the error.														
ERRORS	<p>The <code>link()</code> function will fail if:</p> <table> <tr> <td>EACCES</td><td> <p>A component of either path prefix denies search permission. To override this restriction, the calling process may assert one or both of these privileges: <code>PRIV_FILE_MAC_SEARCH</code> and <code>PRIV_FILE_DAC_SEARCH</code>.</p> <p>The requested link requires writing in a directory with a mode that denies write permission. To override this restriction, the calling process may assert one or both of these privileges: <code>PRIV_FILE_MAC_WRITE</code> and <code>PRIV_FILE_DAC_WRITE</code>.</p> <p>The calling process needs both mandatory read and write access to <i>existing</i> and does not have that combination. To override this restriction, the calling process may assert one or both of these privileges: <code>PRIV_FILE_MAC_READ</code> and <code>PRIV_FILE_MAC_WRITE</code>.</p> </td></tr> <tr> <td>EDQUOT</td><td>The directory where the entry for the new link is being placed cannot be extended because the user's quota of disk blocks on that file system has been exhausted.</td></tr> <tr> <td>EEXIST</td><td>The link named by <i>new</i> exists.</td></tr> <tr> <td>EFAULT</td><td>The <i>existing</i> or <i>new</i> argument points to an illegal address.</td></tr> <tr> <td>EINTR</td><td>A signal was caught during the execution of the <code>link()</code> function.</td></tr> <tr> <td>ELOOP</td><td>Too many symbolic links were encountered in translating <i>path</i>.</td></tr> <tr> <td>EMLINK</td><td>The maximum number of links to a file would be exceeded.</td></tr> </table>	EACCES	<p>A component of either path prefix denies search permission. To override this restriction, the calling process may assert one or both of these privileges: <code>PRIV_FILE_MAC_SEARCH</code> and <code>PRIV_FILE_DAC_SEARCH</code>.</p> <p>The requested link requires writing in a directory with a mode that denies write permission. To override this restriction, the calling process may assert one or both of these privileges: <code>PRIV_FILE_MAC_WRITE</code> and <code>PRIV_FILE_DAC_WRITE</code>.</p> <p>The calling process needs both mandatory read and write access to <i>existing</i> and does not have that combination. To override this restriction, the calling process may assert one or both of these privileges: <code>PRIV_FILE_MAC_READ</code> and <code>PRIV_FILE_MAC_WRITE</code>.</p>	EDQUOT	The directory where the entry for the new link is being placed cannot be extended because the user's quota of disk blocks on that file system has been exhausted.	EEXIST	The link named by <i>new</i> exists.	EFAULT	The <i>existing</i> or <i>new</i> argument points to an illegal address.	EINTR	A signal was caught during the execution of the <code>link()</code> function.	ELOOP	Too many symbolic links were encountered in translating <i>path</i> .	EMLINK	The maximum number of links to a file would be exceeded.
EACCES	<p>A component of either path prefix denies search permission. To override this restriction, the calling process may assert one or both of these privileges: <code>PRIV_FILE_MAC_SEARCH</code> and <code>PRIV_FILE_DAC_SEARCH</code>.</p> <p>The requested link requires writing in a directory with a mode that denies write permission. To override this restriction, the calling process may assert one or both of these privileges: <code>PRIV_FILE_MAC_WRITE</code> and <code>PRIV_FILE_DAC_WRITE</code>.</p> <p>The calling process needs both mandatory read and write access to <i>existing</i> and does not have that combination. To override this restriction, the calling process may assert one or both of these privileges: <code>PRIV_FILE_MAC_READ</code> and <code>PRIV_FILE_MAC_WRITE</code>.</p>														
EDQUOT	The directory where the entry for the new link is being placed cannot be extended because the user's quota of disk blocks on that file system has been exhausted.														
EEXIST	The link named by <i>new</i> exists.														
EFAULT	The <i>existing</i> or <i>new</i> argument points to an illegal address.														
EINTR	A signal was caught during the execution of the <code>link()</code> function.														
ELOOP	Too many symbolic links were encountered in translating <i>path</i> .														
EMLINK	The maximum number of links to a file would be exceeded.														

link(2)

ENAMETOOLONG	The length of the <i>existing</i> or <i>new</i> argument exceeds <code>PATH_MAX</code> , or the length of a <i>existing</i> or <i>new</i> component exceeds <code>NAME_MAX</code> while <code>_POSIX_NO_TRUNC</code> is in effect.
ENOENT	The <i>existing</i> or <i>new</i> argument is a null pathname; a component of either path prefix does not exist; or the file named by <i>existing</i> does not exist.
ENOLINK	The <i>existing</i> or <i>new</i> argument points to a remote machine and the link to that machine is no longer active.
ENOSPC	The directory that would contain the link cannot be extended.
ENOTDIR	A component of either path prefix is not a directory.
EPERM	The file named by <i>existing</i> is a directory and the calling process has not asserted the <code>PRIV_SYS_CONFIG</code> privilege.
EROFS	The requested link requires writing in a directory on a read-only file system.
EXDEV	The link named by <i>new</i> and the file named by <i>existing</i> are on different logical devices (file systems).

ATTRIBUTES See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Async-Signal-Safe

**SUMMARY OF
TRUSTED
SOLARIS
CHANGES**

Appropriate privilege is required to override access checks.

If *existing* is a directory, the calling process must assert the `PRIV_SYS_CONFIG` privilege.

**Trusted Solaris 8
4/01 Reference
Manual**

`symlink(2)`, `unlink(2)`

llseek(2)

NAME	llseek – move extended read/write file pointer						
SYNOPSIS	<pre>#include <sys/types.h> #include <unistd.h> offset_t llseek(int <i>fildev</i>, offset_t <i>offset</i>, int <i>whence</i>);</pre>						
DESCRIPTION	<p>The <code>llseek()</code> function sets the 64-bit extended file pointer associated with the open file descriptor specified by <i>fildev</i> as follows:</p> <ul style="list-style-type: none">■ If <i>whence</i> is <code>SEEK_SET</code>, the pointer is set to <i>offset</i> bytes.■ If <i>whence</i> is <code>SEEK_CUR</code>, the pointer is set to its current location plus <i>offset</i>.■ If <i>whence</i> is <code>SEEK_END</code>, the pointer is set to the size of the file plus <i>offset</i>. <p>On success, <code>llseek()</code> returns the resulting pointer location, measured in bytes from the beginning of the file.</p> <p>Discretionary access checks have already been performed when <i>fildev</i> was opened.</p> <p>Most mandatory access checks have already been performed when <i>fildev</i> was opened. If <i>fildev</i> is open for writing, a check is made that the calling process has mandatory read access in case <i>fildev</i> is open for a write-up. The calling process may assert the <code>PRIV_FILE_MAC_READ</code> privilege to bypass this check. If mandatory read access is not granted, this system call succeeds, but offset data is not returned.</p>						
RETURN VALUES	<p>Upon successful completion, <code>llseek()</code> returns the resulting pointer location as measured in bytes from the beginning of the file. Remote file descriptors are the only ones that allow negative file pointers. Otherwise, <code>-1</code> is returned, the file pointer remains unchanged, and <code>errno</code> is set to indicate the error.</p>						
ERRORS	<p>The <code>llseek()</code> function will fail if:</p> <table><tr><td><code>EBADF</code></td><td>The <i>fildev</i> argument is not an open file descriptor.</td></tr><tr><td><code>EINVAL</code></td><td>The <i>whence</i> argument is not <code>SEEK_SET</code>, <code>SEEK_CUR</code>, or <code>SEEK_END</code>; the <i>offset</i> argument is not a valid offset for this file system type; or the <i>fildev</i> argument is not a remote file descriptor, and the resulting file pointer would be negative.</td></tr><tr><td><code>ESPIPE</code></td><td>The <i>fildev</i> argument is associated with a pipe or FIFO.</td></tr></table>	<code>EBADF</code>	The <i>fildev</i> argument is not an open file descriptor.	<code>EINVAL</code>	The <i>whence</i> argument is not <code>SEEK_SET</code> , <code>SEEK_CUR</code> , or <code>SEEK_END</code> ; the <i>offset</i> argument is not a valid offset for this file system type; or the <i>fildev</i> argument is not a remote file descriptor, and the resulting file pointer would be negative.	<code>ESPIPE</code>	The <i>fildev</i> argument is associated with a pipe or FIFO.
<code>EBADF</code>	The <i>fildev</i> argument is not an open file descriptor.						
<code>EINVAL</code>	The <i>whence</i> argument is not <code>SEEK_SET</code> , <code>SEEK_CUR</code> , or <code>SEEK_END</code> ; the <i>offset</i> argument is not a valid offset for this file system type; or the <i>fildev</i> argument is not a remote file descriptor, and the resulting file pointer would be negative.						
<code>ESPIPE</code>	The <i>fildev</i> argument is associated with a pipe or FIFO.						
SUMMARY OF TRUSTED SOLARIS CHANGES	<p>Appropriate privilege is required to override access checks.</p> <p>Discretionary access checks have already been performed when <i>fildev</i> was opened.</p> <p>Most mandatory access checks have already been performed when <i>fildev</i> was opened. The calling process may assert the <code>PRIV_FILE_MAC_READ</code> privilege to perform a write-up.</p>						
Trusted Solaris 8 4/01 Reference Manual	<p><code>creat(2)</code>, <code>fcntl(2)</code>, <code>lseek(2)</code>, <code>open(2)</code></p> <p><code>dup(2)</code></p>						

NAME	lseek – move read/write file pointer						
SYNOPSIS	<pre>#include <sys/types.h> #include <unistd.h> off_t lseek(int <i>fildes</i>, off_t <i>offset</i>, int <i>whence</i>);</pre>						
DESCRIPTION	<p>The <code>lseek()</code> function sets the file pointer associated with the open file descriptor specified by <i>fildes</i> as follows:</p> <ul style="list-style-type: none"> ■ If <i>whence</i> is <code>SEEK_SET</code>, the pointer is set to <i>offset</i> bytes. ■ If <i>whence</i> is <code>SEEK_CUR</code>, the pointer is set to its current location plus <i>offset</i>. ■ If <i>whence</i> is <code>SEEK_END</code>, the pointer is set to the size of the file plus <i>offset</i>. <p>The symbolic constants <code>SEEK_SET</code>, <code>SEEK_CUR</code>, and <code>SEEK_END</code> are defined in the header <code><unistd.h></code>.</p> <p>Some devices are incapable of seeking. The value of the file pointer associated with such a device is undefined.</p> <p>The <code>lseek()</code> function allows the file pointer to be set beyond the existing data in the file. If data are later written at this point, subsequent reads in the gap between the previous end of data and the newly written data will return bytes of value 0 until data are written into the gap.</p> <p>If <i>fildes</i> is a remote file descriptor and <i>offset</i> is negative, <code>lseek()</code> returns the file pointer even if it is negative. The <code>lseek()</code> function will not, by itself, extend the size of a file.</p> <p>Discretionary access checks have already been performed when <i>fildes</i> was opened.</p> <p>Most mandatory access checks have already been performed when <i>fildes</i> was opened. If <i>fildes</i> is open for writing, a check is made that the calling process has mandatory read access in case <i>fildes</i> is open for a write-up. The calling process may assert the <code>PRIV_FILE_MAC_READ</code> privilege to bypass this check. If mandatory read access is not granted, this system call succeeds; but offset data is not returned.</p>						
RETURN VALUES	Upon successful completion, the resulting offset, as measured in bytes from the beginning of the file, is returned. Otherwise, <code>(off_t)-1</code> is returned, the file offset remains unchanged, and <code>errno</code> is set to indicate the error.						
ERRORS	<p>The <code>lseek()</code> function will fail if:</p> <table> <tr> <td>EBADF</td><td>The <i>fildes</i> argument is not an open file descriptor.</td></tr> <tr> <td>EINVAL</td><td>The <i>whence</i> argument is not <code>SEEK_SET</code>, <code>SEEK_CUR</code>, or <code>SEEK_END</code>; or the <i>fildes</i> argument is not a remote file descriptor and the resulting file pointer would be negative.</td></tr> <tr> <td>EOVERFLOW</td><td>The resulting file offset would be a value which cannot be represented correctly in an object of type <code>off_t</code> for regular files.</td></tr> </table>	EBADF	The <i>fildes</i> argument is not an open file descriptor.	EINVAL	The <i>whence</i> argument is not <code>SEEK_SET</code> , <code>SEEK_CUR</code> , or <code>SEEK_END</code> ; or the <i>fildes</i> argument is not a remote file descriptor and the resulting file pointer would be negative.	EOVERFLOW	The resulting file offset would be a value which cannot be represented correctly in an object of type <code>off_t</code> for regular files.
EBADF	The <i>fildes</i> argument is not an open file descriptor.						
EINVAL	The <i>whence</i> argument is not <code>SEEK_SET</code> , <code>SEEK_CUR</code> , or <code>SEEK_END</code> ; or the <i>fildes</i> argument is not a remote file descriptor and the resulting file pointer would be negative.						
EOVERFLOW	The resulting file offset would be a value which cannot be represented correctly in an object of type <code>off_t</code> for regular files.						

lseek(2)

USAGE ESPIPE The *fildev* argument is associated with a pipe, a FIFO, or a socket.

The `lseek()` function has a transitional interface for 64-bit file offsets. See `lseek64(5)`.

In multithreaded applications, using `lseek()` in conjunction with a `read(2)` or `write(2)` call on a file descriptor shared by more than one thread is not an atomic operation. To ensure atomicity, use `pread()` or `pwrite()`.

ATTRIBUTES See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Async-Signal-Safe

SUMMARY OF TRUSTED SOLARIS CHANGES

Appropriate privilege is required to override access checks.

Discretionary access checks have already been performed when *fildev* was opened.

Most mandatory access checks have already been performed when *fildev* was opened. The calling process may assert the `PRIV_FILE_MAC_READ` privilege to perform a write-up.

**Trusted Solaris 8
4/01 Reference
Manual
SunOS 5.9
Reference Manual**

`creat(2)`, `fcntl(2)`, `lseek(2)`, `open(2)`

`dup(2)`, `attributes(5)`

NAME	setcmwlabel, fsetcmwlabel, lsetcmwlabel – Set CMW label of a file
SYNOPSIS	<pre>cc [flags...] file ... -ltsol [library...] #include <tsol/label.h> int setcmwlabel(const char *path, const bclabel_t *label_p, const setting_flag_t flag); int fsetcmwlabel(int fd, const bclabel_t *label_p, const setting_flag_t flag); int lsetcmwlabel(const char *path, const bclabel_t *label_p, const setting_flag_t flag);</pre>
DESCRIPTION	<p>The file that is named by <i>path</i> or referred to by <i>fd</i> has its CMW label changed as specified provided the file resides on a file system that supports the setting of labels on individual objects.</p> <p>If <i>flag</i> equals SETCL_ALL, then both parts of the file's CMW label are to be set and the following checks must be made:</p> <ul style="list-style-type: none"> ■ The sensitivity label of <i>label_p</i> must be in the sensitivity label range of the containing file system. ■ If the sensitivity label of <i>label_p</i> equals the existing sensitivity label, then neither PRIV_FILE_UPGRADE_SL nor PRIV_FILE_DOWNGRADE_SL is required. ■ If the sensitivity label of <i>label_p</i> dominates but does not equal the existing sensitivity label (an upgrade), then the calling process must have PRIV_FILE_UPGRADE_SL in its set of effective privileges. ■ If the sensitivity label of <i>label_p</i> does not dominate the existing sensitivity label (a downgrade), then the calling process must have PRIV_FILE_DOWNGRADE_SL in its set of effective privileges. ■ If the sensitivity label operation is a downgrade and the calling process is not the owner of the file, then the calling process must have PRIV_FILE_OWNER in its set of effective privileges. <p>If <i>flag</i> equals SETCL_SL, then the sensitivity label of the file's CMW label is to be set and the following checks must be made:</p> <ul style="list-style-type: none"> ■ The sensitivity label of <i>label_p</i> must be in the sensitivity label range of the containing file system. ■ If the sensitivity label of <i>label_p</i> equals the existing sensitivity label, then neither PRIV_FILE_UPGRADE_SL nor PRIV_FILE_DOWNGRADE_SL is required. ■ If the sensitivity label of <i>label_p</i> dominates but does not equal the existing sensitivity label (an upgrade), then the calling process must have PRIV_FILE_UPGRADE_SL in its set of effective privileges. ■ If the sensitivity label of <i>label_p</i> does not dominate the existing sensitivity label (a downgrade), then the calling process must have PRIV_FILE_DOWNGRADE_SL in

lsetcmwlabel(2)

	<p>its set of effective privileges.</p> <ul style="list-style-type: none"> ■ If the operation is a sensitivity label downgrade and the calling process is not the owner of the file, then the calling process must have <code>PRIV_FILE_OWNER</code> in its set of effective privileges. <p>There are several checks that are applicable if the sensitivity label is being changed:</p> <ul style="list-style-type: none"> ■ The calling process must have discretionary write access to the file. ■ If there is an open file descriptor reference to the file, then the calling process must have <code>PRIV_PROC_TRANQUIL</code> in its set of effective privileges. <p><code>setcmwlabel()</code> and <code>lsetcmwlabel()</code> function identically except when the final component is a symbolic link. If the final component is a symbolic link, <code>lsetcmwlabel()</code> sets the CMW label of the symbolic link, but <code>setcmwlabel()</code> sets the CMW label of the object referred to by the symbolic link.</p>								
NOTES	<p>If the sensitivity label is being set, then the calling process is responsible for verifying that sensitivity label is within the accreditation range of the system.</p>								
RETURN VALUES	<p><code>setcmwlabel()</code>, <code>fsetcmwlabel()</code>, and <code>lsetcmwlabel()</code> return:</p> <p>0 On success.</p> <p>-1 On failure, and set <code>errno</code> to indicate the error.</p>								
ERRORS	<p><code>setcmwlabel()</code> and <code>lsetcmwlabel()</code> fail and the file is unchanged if any of these conditions prevails:</p> <table> <tr> <td>EACCES</td><td> <p>Search permission is denied for a component of the path prefix of <i>path</i>.</p> <p>The calling process does not have mandatory write access to the final component of <i>path</i> because the sensitivity label of the final component of <i>path</i> does not dominate the sensitivity label of the calling process and the calling process does not have <code>PRIV_FILE_MAC_WRITE</code> in its set of effective privileges.</p> <p>The calling process does not have discretionary write access to the final component of <i>path</i>.</p> </td></tr> <tr> <td>EBUSY</td><td> <p>There is an open file descriptor reference to the final component of <i>path</i> and the calling process does not have <code>PRIV_PROC_TRANQUIL</code> in its set of effective privileges.</p> </td></tr> <tr> <td>EFAULT</td><td> <p><i>path</i> or <i>label_p</i> points outside the allocated address space of the process.</p> </td></tr> <tr> <td>EINVAL</td><td> <p><i>path</i> does not reside on a file system that supports the setting of labels on individual objects.</p> </td></tr> </table>	EACCES	<p>Search permission is denied for a component of the path prefix of <i>path</i>.</p> <p>The calling process does not have mandatory write access to the final component of <i>path</i> because the sensitivity label of the final component of <i>path</i> does not dominate the sensitivity label of the calling process and the calling process does not have <code>PRIV_FILE_MAC_WRITE</code> in its set of effective privileges.</p> <p>The calling process does not have discretionary write access to the final component of <i>path</i>.</p>	EBUSY	<p>There is an open file descriptor reference to the final component of <i>path</i> and the calling process does not have <code>PRIV_PROC_TRANQUIL</code> in its set of effective privileges.</p>	EFAULT	<p><i>path</i> or <i>label_p</i> points outside the allocated address space of the process.</p>	EINVAL	<p><i>path</i> does not reside on a file system that supports the setting of labels on individual objects.</p>
EACCES	<p>Search permission is denied for a component of the path prefix of <i>path</i>.</p> <p>The calling process does not have mandatory write access to the final component of <i>path</i> because the sensitivity label of the final component of <i>path</i> does not dominate the sensitivity label of the calling process and the calling process does not have <code>PRIV_FILE_MAC_WRITE</code> in its set of effective privileges.</p> <p>The calling process does not have discretionary write access to the final component of <i>path</i>.</p>								
EBUSY	<p>There is an open file descriptor reference to the final component of <i>path</i> and the calling process does not have <code>PRIV_PROC_TRANQUIL</code> in its set of effective privileges.</p>								
EFAULT	<p><i>path</i> or <i>label_p</i> points outside the allocated address space of the process.</p>								
EINVAL	<p><i>path</i> does not reside on a file system that supports the setting of labels on individual objects.</p>								

	The sensitivity label of <i>label_p</i> is not in the sensitivity label range of the containing file system.
EIO	An I/O error occurred while reading from or writing to the file system.
ELOOP	Too many symbolic links were encountered in translating <i>path</i> .
ENAMETOOLONG	The length of the path argument exceeds <code>PATH_MAX</code> . A pathname component is longer than <code>NAME_MAX</code> [see <code>sysconf(3C)</code>] while <code>_POSIX_NO_TRUNC</code> is in effect. See <code>pathconf(2)</code> .
ENOENT	The file referred to by <i>path</i> does not exist.
ENOTDIR	A component of the path prefix of <i>path</i> is not a directory.
EPERM	The calling process does not have mandatory write access to the final component of <i>path</i> because the sensitivity label of the final component of <i>path</i> is outside the clearance of the calling process and the calling process does not have <code>PRIV_FILE_MAC_WRITE</code> in its set of effective privileges. A calling process that is not the owner of the file attempted to downgrade the sensitivity label associated with the final component of <i>path</i> but did not have <code>PRIV_FILE_OWNER</code> in its set of effective privileges. The calling process attempted to upgrade the sensitivity label associated with the final component of <i>path</i> but did not have <code>PRIV_FILE_UPGRADE_SL</code> in its set of effective privileges. The calling process attempted to downgrade the sensitivity label associated with the final component of <i>path</i> but did not have <code>PRIV_FILE_DOWNGRADE_SL</code> in its set of effective privileges.
EROFS	The file referred to by <i>path</i> resides on a read-only file system.
<code>fsetcmwlabel()</code>	fails if any of these conditions prevails:
EBADF	<i>fd</i> does not refer to a valid descriptor.
EBUSY	There is an open file descriptor reference to the object referred to by the descriptor and the calling process does not have <code>PRIV_PROC_TRANQUIL</code> in its set of effective privileges.
EFAULT	<i>label_p</i> points outside the allocated address space of the process.
EINVAL	<i>fd</i> refers to a socket, not a file. <i>fd</i> does not refer to a file on a file system that supports the setting of labels on individual objects.

lsetcmwlabel(2)

	<p>The sensitivity label of <i>label_p</i> is not in the sensitivity label range of the containing file system.</p>
EIO	<p>An I/O error occurred while reading from or writing to the file system.</p> <p>The calling process is not the owner of the file, attempted to downgrade the sensitivity label associated with the file, but did not have <code>PRIV_FILE_OWNER</code> in its set of effective privileges.</p> <p>The calling process attempted to upgrade the sensitivity label associated with the file but did not have <code>PRIV_FILE_UPGRADE_SL</code> in its set of effective privileges.</p> <p>The calling process attempted to downgrade the sensitivity label associated with the file but did not have <code>PRIV_FILE_DOWNGRADE_SL</code> in its set of effective privileges.</p>
EPERM	<p>The calling process does not have mandatory write access to the object referred to by <i>fd</i> because the sensitivity label of the object referred to by <i>fd</i> is outside the clearance of the calling process and the calling process does not have <code>PRIV_FILE_MAC_WRITE</code> in its set of effective privileges.</p> <p>A calling process that is not the owner of the file attempted to downgrade the sensitivity label associated with the object referred to by <i>fd</i> but did not have <code>PRIV_FILE_OWNER</code> in its set of effective privileges.</p> <p>The calling process attempted to upgrade the sensitivity label associated with the object referred to by <i>fd</i> but did not have <code>PRIV_FILE_UPGRADE_SL</code> in its set of effective privileges.</p> <p>The calling process attempted to downgrade the sensitivity label associated with the object referred to by <i>fd</i> but did not have <code>PRIV_FILE_DOWNGRADE_SL</code> in its set of effective privileges.</p>
EROFS	<p>The file referred to by <i>fd</i> resides on a read-only file system.</p>
getcmwfsrange(2), getcmwlabel(2)	

Trusted Solaris 8
4/01 Reference
Manual

NAME	stat, lstat, fstat – get file status				
SYNOPSIS	<pre>#include <sys/types.h> #include <sys/stat.h> int stat(const char *path, struct stat *buf); int lstat(const char *path, struct stat *buf); int fstat(int fd, struct stat *buf);</pre>				
DESCRIPTION	<p>The <code>stat()</code> function obtains information about the file pointed to by <i>path</i>. Read, write, or execute permission of the named file is not required, but all directories listed in the path name leading to the file must be searchable.</p> <p>The <code>lstat()</code> function obtains file attributes similar to <code>stat()</code>, except when the named file is a symbolic link; in that case <code>lstat()</code> returns information about the link, while <code>stat()</code> returns information about the file the link references.</p> <p>The <code>fstat()</code> function obtains information about an open file known by the file descriptor <i>fd</i>, obtained from a successful <code>open(2)</code>, <code>creat(2)</code>, <code>dup(2)</code>, <code>fcntl(2)</code>, or <code>pipe(2)</code> function.</p> <p>The <i>buf</i> argument is a pointer to a <code>stat</code> structure into which information is placed concerning the file. A <code>stat</code> structure includes the following members:</p> <pre>mode_t st_mode; /* File mode (see mknod(2)) */ ino_t st_ino; /* Inode number */ dev_t st_dev; /* ID of device containing */ dev_t st_rdev; /* a directory entry for this file */ dev_t st_rdev; /* ID of device */ /* This entry is defined only for */ /* char special or block special files */ nlink_t st_nlink; /* Number of links */ uid_t st_uid; /* User ID of the file's owner */ gid_t st_gid; /* Group ID of the file's group */ off_t st_size; /* File size in bytes */ time_t st_atime; /* Time of last access */ time_t st_mtime; /* Time of last data modification */ time_t st_ctime; /* Time of last file status change */ /* Times measured in seconds since */ /* 00:00:00 UTC, Jan. 1, 1970 */ long st_blksize; /* Preferred I/O block size */ blkcnt_t st_blocks; /* Number of 512 byte blocks allocated*/</pre> <p>Descriptions of structure members are as follows:</p> <table> <tr> <td>st_mode</td><td>The mode of the file as described in <code>mknod(2)</code>. In addition to the modes described in <code>mknod()</code>, the mode of a file may also be <code>S_IFLNK</code> if the file is a symbolic link. <code>S_IFLNK</code> may only be returned by <code>lstat()</code>.</td></tr> <tr> <td>st_ino</td><td>This field uniquely identifies the file in a given file system. The pair <code>st_ino</code> and <code>st_dev</code> uniquely identifies regular files.</td></tr> </table>	st_mode	The mode of the file as described in <code>mknod(2)</code> . In addition to the modes described in <code>mknod()</code> , the mode of a file may also be <code>S_IFLNK</code> if the file is a symbolic link. <code>S_IFLNK</code> may only be returned by <code>lstat()</code> .	st_ino	This field uniquely identifies the file in a given file system. The pair <code>st_ino</code> and <code>st_dev</code> uniquely identifies regular files.
st_mode	The mode of the file as described in <code>mknod(2)</code> . In addition to the modes described in <code>mknod()</code> , the mode of a file may also be <code>S_IFLNK</code> if the file is a symbolic link. <code>S_IFLNK</code> may only be returned by <code>lstat()</code> .				
st_ino	This field uniquely identifies the file in a given file system. The pair <code>st_ino</code> and <code>st_dev</code> uniquely identifies regular files.				

lstat(2)

	<code>st_dev</code>	This field uniquely identifies the file system that contains the file. Its value may be used as input to the <code>ustat()</code> function to determine more information about this file system. No other meaning is associated with this value.
	<code>st_rdev</code>	This field should be used only by administrative commands. It is valid only for block special or character special files and only has meaning on the system where the file was configured.
	<code>st_nlink</code>	This field should be used only by administrative commands.
	<code>st_uid</code>	The user ID of the file's owner.
	<code>st_gid</code>	The group ID of the file's group.
	<code>st_size</code>	For regular files, this is the address of the end of the file. For block special or character special, this is not defined. See also <code>pipe(2)</code> .
	<code>st_atime</code>	Time when file data was last accessed. Changed by the following functions: <code>creat()</code> , <code>mknod()</code> , <code>pipe()</code> , <code>utime(2)</code> , and <code>read(2)</code> .
	<code>st_mtime</code>	Time when data was last modified. Changed by the following functions: <code>creat()</code> , <code>mknod()</code> , <code>pipe()</code> , <code>utime()</code> , and <code>write(2)</code> .
	<code>st_ctime</code>	Time when file status was last changed. Changed by the following functions: <code>chmod()</code> , <code>chown()</code> , <code>creat()</code> , <code>link(2)</code> , <code>mknod()</code> , <code>pipe()</code> , <code>unlink(2)</code> , <code>utime()</code> , and <code>write()</code> .
	<code>st_blksize</code>	A hint as to the "best" unit size for I/O operations. This field is not defined for block special or character special files.
	<code>st_blocks</code>	The total number of physical blocks of size 512 bytes actually allocated on disk. This field is not defined for block special or character special files.
	<p><code>stat()</code>, <code>lstat()</code>, and <code>fstat()</code> require mandatory read access to the final component of <i>path</i>. If the file descriptor is open only for writing, <code>fstat()</code> requires mandatory read access to the object to which the descriptor refers. To override these restrictions, the calling process may assert the <code>PRIV_FILE_MAC_READ</code> privilege in its set of effective privileges.</p> <p>If the calling process does not have mandatory read access, <code>stat()</code>, <code>lstat()</code>, and <code>fstat()</code> return fixed values for some elements of the <code>stat</code> structure.</p>	
RETURN VALUES	Upon successful completion, 0 is returned. Otherwise, -1 is returned and <code>errno</code> is set to indicate the error.	
ERRORS	The <code>stat()</code> , <code>fstat()</code> , and <code>lstat()</code> functions will fail if:	
	<code>EOVERFLOW</code>	The file size in bytes or the number of blocks allocated to the file or the file serial number cannot be represented correctly in the structure pointed to by <i>buf</i> .

The `stat()` and `lstat()` functions will fail if:

EACCES	Search permission is denied for a component of the path prefix. To override this restriction, the calling process may assert one or both of these privileges: <code>PRIV_FILE_DAC_SEARCH</code> and <code>PRIV_FILE_MAC_SEARCH</code> .
EFAULT	The <i>buf</i> or <i>path</i> argument points to an illegal address.
EINTR	A signal was caught during the execution of the <code>stat()</code> or <code>lstat()</code> function.
ELOOP	Too many symbolic links were encountered in translating <i>path</i> .
ENAMETOOLONG	The length of the <i>path</i> argument exceeds <code>PATH_MAX</code> , or the length of a <i>path</i> component exceeds <code>NAME_MAX</code> while <code>_POSIX_NO_TRUNC</code> is in effect.
ENOENT	The named file does not exist or is the null pathname.
ENOLINK	The <i>path</i> argument points to a remote machine and the link to that machine is no longer active.
ENOTDIR	A component of the path prefix is not a directory.
EOVERFLOW	A component is too large to store in the structure pointed to by <i>buf</i> .

The `fstat()` function will fail if:

EBADF	The <i>fildev</i> argument is not a valid open file descriptor.
EFAULT	The <i>buf</i> argument points to an illegal address.
EINTR	A signal was caught during the execution of the <code>fstat()</code> function.
ENOLINK	The <i>fildev</i> argument points to a remote machine and the link to that machine is no longer active.
EOVERFLOW	A component is too large to store in the structure pointed to by <i>buf</i> .

USAGE The `stat()`, `fstat()`, and `lstat()` functions have transitional interfaces for 64-bit file offsets. See `lf64(5)`.

ATTRIBUTES See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	<code>stat()</code> and <code>fstat()</code> are Async-Signal-Safe

SUMMARY OF TRUSTED SOLARIS CHANGES

`stat()`, `lstat()`, and `fstat()` require mandatory read access to the final component of *path*. If the file descriptor is open only for writing, `fstat()` requires

lstat(2)

mandatory read access to the object to which the descriptor refers. To override these restrictions, the calling process may assert the `PRIV_FILE_MAC_READ` privilege in its set of effective privileges.

To override access restrictions, the calling process of `stat()` or `lstat()` may also assert one or both of these privileges: `PRIV_FILE_DAC_SEARCH` and `PRIV_FILE_MAC_SEARCH`.

Certain uses of this interface may present a covert channel. If a covert channel is exploited, the execution of the process may be delayed. To bypass this delay, the process may assert the `PRIV_PROC_NODELAY` privilege.

**Trusted Solaris 8
4/01 Reference
Manual**

`chmod(2)`, `chown(2)`, `creat(2)`, `fcntl(2)`, `link(2)`, `mknod(2)`, `open(2)`, `read(2)`, `unlink(2)`, `write(2)`

**SunOS 5.8
Reference Manual
NOTES**

`dup(2)`, `pipe(2)`, `time(2)`, `utime(2)`, `fattach(3C)`, `stat(3HEAD)`, `attributes(5)`

If you use `chmod(2)` to change the file group owner permissions on a file with ACL entries, both the file group owner permissions and the ACL mask are changed to the new permissions. Be aware that the new ACL mask permissions may change the effective permissions for additional users and groups who have ACL entries on the file.

NAME	mkdir – make a directory
SYNOPSIS	<pre>#include <sys/types.h> #include <sys/stat.h> int mkdir(const char *path, mode_t mode);</pre>
DESCRIPTION	<p>The <code>mkdir()</code> function creates a new directory named by the path name pointed to by <i>path</i>. The mode of the new directory is initialized from <i>mode</i> (see <code>chmod(2)</code> for values of mode). The protection part of the <i>mode</i> argument is modified by the process' file creation mask (see <code>umask(2)</code>).</p> <p>The directory's owner ID is set to the process's effective user ID. The directory's group ID is set to the process's effective group ID, or if the <code>S_ISGID</code> bit is set in the parent directory, then the group ID of the directory is inherited from the parent. The <code>S_ISGID</code> bit of the new directory is inherited from the parent directory.</p> <p>If <i>path</i> is a symbolic link, it is not followed.</p> <p>The newly created directory is empty with the exception of entries for itself (<code>.</code>) and its parent directory (<code>..</code>).</p> <p>Upon successful completion, <code>mkdir()</code> marks for update the <code>st_atime</code>, <code>st_ctime</code>, and <code>st_mtime</code> fields of the directory. Also, the <code>st_ctime</code> and <code>st_mtime</code> fields of the directory that contains the new entry are marked for update. This system call will not create a directory in a multilevel directory. Single-level directories are automatically created as needed during path-name lookup and the <code>getslcname(2)</code> system call.</p> <p>The Trusted Solaris operating environment distinguishes directories with sensitivity labels from unlabeled directories through special prefixes called adornments that are appended to the beginning of the directory's name. See <code>setfsattr(1M)</code>. A multilevel directory has the default adornment <code>".MLD."</code>; a single-level directory has the adornment <code>".SLD.n/"</code>, where <i>n</i> is a number. If the directory name includes the multilevel adornment, the directory will be created as a multilevel directory, provided all other conditions for success are met. Use the <code>mlpwd</code> command within a multilevel directory to see the adorned names of the multilevel directory and the single-level directories. For example, executing the <code>mlpwd</code> command within the <i>user_name</i> home directory shows this output: <code>/export/home/.MLD.user_name/.SLD.2</code></p> <p>Use the <code>mlcrealpath</code> command to see the adorned name for a file or directory within a multilevel directory. For example, <code>mlcrealpath file.c</code> shows this output: <code>/export/home/.MLD.user_name/.SLD.2/file.c</code></p> <p>The new directory is created with its sensitivity label set to the sensitivity label of the calling process.</p> <p>If the new directory's containing directory has a default access control list (ACL), the default and access ACLs of the new directory are set to the default ACL of the containing directory.</p>

mkdir(2)

RETURN VALUES

mkdir() returns:

- 0 On success.
- 1 On failure, and sets errno to indicate the error.

ERRORS

The mkdir() function will fail if:

EACCES	Either a component of the path prefix denies search permission or write permission is denied on the parent directory of the directory to be created. To override these restrictions, the calling process may assert one or more of these privileges: PRIV_FILE_DAC_SEARCH, PRIV_FILE_MAC_SEARCH, PRIV_FILE_DAC_WRITE, and PRIV_FILE_MAC_WRITE.
EINVAL	An attempt was made to create a directory at a sensitivity label outside the range of the file system.
EDQUOT	The directory where the new file entry is being placed cannot be extended because the user's quota of disk blocks on that file system has been exhausted; the new directory cannot be created because the user's quota of disk blocks on that file system has been exhausted; or the user's quota of inodes on the file system where the file is being created has been exhausted.
EEXIST	The named file already exists.
EFAULT	The <i>path</i> argument points to an illegal address.
EIO	An I/O error has occurred while accessing the file system.
ELOOP	Too many symbolic links were encountered in translating <i>path</i> .
EMLINK	The maximum number of links to the parent directory would be exceeded.
ENAMETOOLONG	The length of the <i>path</i> argument exceeds PATH_MAX, or the length of a <i>path</i> component exceeds NAME_MAX while _POSIX_NO_TRUNC is in effect.
ENOENT	A component of the path prefix does not exist or is a null pathname.
ENOLINK	The <i>path</i> argument points to a remote machine and the link to that machine is no longer active.
ENOSPC	No free space is available on the device containing the directory.
ENOTDIR	A component of the path prefix is not a directory.

mkdir(2)

EROFS

The path prefix resides on a read-only file system.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Async-Signal-Safe

SUMMARY OF TRUSTED SOLARIS CHANGES

Appropriate privilege is required to override access checks as described under File Access in Intro(2). The Trusted Solaris environment distinguishes directories with sensitivity labels from unlabeled directories through special prefixes called adornments.

Trusted Solaris 8
4/01 Reference
Manual
SunOS 5.8
Reference Manual

adornfc(1M), chmod(2), mknod(2)

umask(2), attributes(5), stat(5)

mknod(2)

NAME	mknod – make a directory, or a special or ordinary file																																														
SYNOPSIS	<pre>#include <sys/stat.h> int mknod(const char *path, mode_t mode, dev_t dev);</pre>																																														
DESCRIPTION	<p>The <code>mknod()</code> function creates a new file named by the path name pointed to by <i>path</i>. The file type and permissions of the new file are initialized from <i>mode</i>. This system call will not create an object in a multilevel directory. Single-level directories are automatically created during path-name lookup and <code>getslename(2)</code>.</p> <p>The new object is created with its sensitivity label set to the sensitivity label of the calling process. If the containing directory has a default access control list (ACL), the ACL is copied to the new object as its access ACL.</p> <p>The file type is specified in <i>mode</i> by the <code>S_IFMT</code> bits, which must be set to one of the following values:</p> <table><tr><td><code>S_IFIFO</code></td><td>fifo special</td></tr><tr><td><code>S_IFCHR</code></td><td>character special</td></tr><tr><td><code>S_IFDIR</code></td><td>directory</td></tr><tr><td><code>S_IFBLK</code></td><td>block special</td></tr><tr><td><code>S_IFREG</code></td><td>ordinary file</td></tr></table> <p>The file access permissions are specified in <i>mode</i> by the 0007777 bits, and may be constructed by a bitwise OR operation of the following values:</p> <table><tr><td><code>S_ISUID</code></td><td>04000</td><td>Set user ID on execution.</td></tr><tr><td><code>S_ISGID</code></td><td>020#0</td><td>Set group ID on execution if # is 7, 5, 3, or 1. Enable mandatory file/record locking if # is 6, 4, 2, or 0</td></tr><tr><td><code>S_ISVTX</code></td><td>01000</td><td>Save text image after execution.</td></tr><tr><td><code>S_IRWXU</code></td><td>00700</td><td>Read, write, execute by owner.</td></tr><tr><td><code>S_IRUSR</code></td><td>00400</td><td>Read by owner.</td></tr><tr><td><code>S_IWUSR</code></td><td>00200</td><td>Write by owner.</td></tr><tr><td><code>S_IXUSR</code></td><td>00100</td><td>Execute (search if a directory) by owner.</td></tr><tr><td><code>S_IRWXG</code></td><td>00070</td><td>Read, write, execute by group.</td></tr><tr><td><code>S_IRGRP</code></td><td>00040</td><td>Read by group.</td></tr><tr><td><code>S_IWGRP</code></td><td>00020</td><td>Write by group.</td></tr><tr><td><code>S_IXGRP</code></td><td>00010</td><td>Execute by group.</td></tr><tr><td><code>S_IRWXO</code></td><td>00007</td><td>Read, write, execute (search) by others.</td></tr></table>	<code>S_IFIFO</code>	fifo special	<code>S_IFCHR</code>	character special	<code>S_IFDIR</code>	directory	<code>S_IFBLK</code>	block special	<code>S_IFREG</code>	ordinary file	<code>S_ISUID</code>	04000	Set user ID on execution.	<code>S_ISGID</code>	020#0	Set group ID on execution if # is 7, 5, 3, or 1. Enable mandatory file/record locking if # is 6, 4, 2, or 0	<code>S_ISVTX</code>	01000	Save text image after execution.	<code>S_IRWXU</code>	00700	Read, write, execute by owner.	<code>S_IRUSR</code>	00400	Read by owner.	<code>S_IWUSR</code>	00200	Write by owner.	<code>S_IXUSR</code>	00100	Execute (search if a directory) by owner.	<code>S_IRWXG</code>	00070	Read, write, execute by group.	<code>S_IRGRP</code>	00040	Read by group.	<code>S_IWGRP</code>	00020	Write by group.	<code>S_IXGRP</code>	00010	Execute by group.	<code>S_IRWXO</code>	00007	Read, write, execute (search) by others.
<code>S_IFIFO</code>	fifo special																																														
<code>S_IFCHR</code>	character special																																														
<code>S_IFDIR</code>	directory																																														
<code>S_IFBLK</code>	block special																																														
<code>S_IFREG</code>	ordinary file																																														
<code>S_ISUID</code>	04000	Set user ID on execution.																																													
<code>S_ISGID</code>	020#0	Set group ID on execution if # is 7, 5, 3, or 1. Enable mandatory file/record locking if # is 6, 4, 2, or 0																																													
<code>S_ISVTX</code>	01000	Save text image after execution.																																													
<code>S_IRWXU</code>	00700	Read, write, execute by owner.																																													
<code>S_IRUSR</code>	00400	Read by owner.																																													
<code>S_IWUSR</code>	00200	Write by owner.																																													
<code>S_IXUSR</code>	00100	Execute (search if a directory) by owner.																																													
<code>S_IRWXG</code>	00070	Read, write, execute by group.																																													
<code>S_IRGRP</code>	00040	Read by group.																																													
<code>S_IWGRP</code>	00020	Write by group.																																													
<code>S_IXGRP</code>	00010	Execute by group.																																													
<code>S_IRWXO</code>	00007	Read, write, execute (search) by others.																																													

S_IROTH	00004	Read by others.
S_IWOTH	00002	Write by others
S_IXOTH	00001	Execute by others.
S_ISVTX		On directories, restricted deletion flag.

The owner ID of the file is set to the effective user ID of the process. The group ID of the file is set to the effective group ID of the process. However, if the `S_ISGID` bit is set in the parent directory, then the group ID of the file is inherited from the parent. If the group ID of the new file does not match the effective group ID or one of the supplementary group IDs, the `S_ISGID` bit is cleared. To override this restriction, the calling process may assert the `PRIV_FILE_SETID` privilege.

If the file is not a directory, mode bit 01000 (save text image on execution) is cleared. The calling process may assert the `PRIV_SYS_CONFIG` privilege to override this restriction.

The access permission bits of *mode* are modified by the process' file mode creation mask: all bits set in the process' file mode creation mask are cleared (see `umask(2)`). If *mode* indicates a block or character special file, *dev* is a configuration-dependent specification of a character or block I/O device. If *mode* does not indicate a block special or character special device, *dev* is ignored. See `makedev(3C)`.

If *path* is a symbolic link, it is not followed.

RETURN VALUES

`mknod()` returns:

- 0 On success.
- 1 On failure, and sets `errno` to indicate the error.

ERRORS

The `mknod()` function will fail if:

- EACCES The calling process does not have search access to all directories in the object's path. To override this restriction, the calling process may assert one or both of these privileges: `PRIV_FILE_DAC_SEARCH` and `PRIV_FILE_MAC_SEARCH`.

The calling process does not have write access to the object's containing directory. To override this restriction, the calling process may assert one or both of these privileges: `PRIV_FILE_DAC_WRITE` and `PRIV_FILE_MAC_WRITE`.
- EDQUOT The directory where the new file entry is being placed cannot be extended because the user's quota of disk blocks on that file system has been exhausted, or the

mknod(2)

		user's quota of inodes on the file system where the file is being created has been exhausted.
	EEXIST	The named file exists.
	EFAULT	The <i>path</i> argument points to an illegal address.
	EINTR	A signal was caught during the execution of the <code>mknod()</code> function.
	EINVAL	An invalid argument exists.
	EIO	An I/O error occurred while accessing the file system.
	ELOOP	Too many symbolic links were encountered in translating <i>path</i> .
	ENAMETOOLONG	The length of the <i>path</i> argument exceeds <code>PATH_MAX</code> , or the length of a <i>path</i> component exceeds <code>NAME_MAX</code> while <code>_POSIX_NO_TRUNC</code> is in effect.
	ENOENT	A component of the path prefix specified by <i>path</i> does not name an existing directory or <i>path</i> is an empty string.
	ENOLINK	The <i>path</i> argument points to a remote machine and the link to that machine is no longer active.
	ENOSPC	The directory that would contain the new file cannot be extended or the file system is out of file allocation resources.
	ENOTDIR	A component of the path prefix is not a directory.
	EPERM	The value in <i>mode</i> is not a FIFO and the calling process has not asserted the <code>PRIV_SYS_DEVICES</code> privilege.
	EROFS	The directory in which the file is to be created is located on a read-only file system.
	The <code>mknod()</code> function may fail if:	
	ENAMETOOLONG	Pathname resolution of a symbolic link produced an intermediate result whose length exceeds <code>PATH_MAX</code> .
USAGE	Normally, applications should use the <code>mkdir(2)</code> routine to make a directory, since the function <code>mknod()</code> may not establish directory entries for the directory itself (<code>.</code>) and the parent directory (<code>..</code>), and appropriate permissions are not required. Similarly, <code>mkfifo(3C)</code> should be used in place of <code>mknod()</code> in order to create FIFOs.	
SUMMARY OF TRUSTED SOLARIS CHANGES	Appropriate privilege is required to override access checks.	

mknod(2)

The new object is created with its sensitivity label set to the sensitivity label of the calling process. If the containing directory has a default access control list (ACL), the ACL is copied to the new object as its access ACL.

chmod(2), exec(2), mkdir(2)
umask(2), makedev(3C), mkfifo(3C), stat(5)

mldgetfattrflag(2)

NAME	getfattrflag, fsetfattrflag, fgetfattrflag, setfattrflag, mldgetfattrflag, mldsetfattrflag – set/get the security attribute flags of a file
SYNOPSIS	<pre>cc [flags...] file... -ltso1 #include <tsol/secflgs.h> int getfattrflag(const char *path, secflgs_t *flags); int setfattrflag(const char *path, secflgs_t which, secflgs_t flags); int fgetfattrflag(int fildes, secflgs_t *flags); int fsetfattrflag(int fildes, secflgs_t which, secflgs_t flags); int mldgetfattrflag(const char *path, secflgs_t *flags); int mldsetfattrflag(const char *path, secflgs_t which, secflgs_t flags);</pre>
DESCRIPTION	<p>setfattrflag(), fsetfattrflag(), and mldsetfattrflag() set the security flags of the file whose name is given by <i>path</i> or referred to by the open file descriptor <i>fildes</i>. The bit pattern contained in <i>which</i> is used to indicate which flags are being affected. The corresponding bits in <i>flags</i> are set to 1 or 0 to indicate whether the affected flags are being set or unset respectively.</p> <p>getfattrflag(), fgetfattrflag(), and mldgetfattrflag() get the security flags of the file whose name is given by <i>path</i> or referred to by the open file descriptor <i>fildes</i> and store it in the location pointed to by <i>flags</i>.</p> <p>Attribute bits are interpreted as follows:</p> <p>FAF_MLD Directory has MLD semantics.</p> <p>FAF_PUBLIC Filesystem object is a public object.</p> <p>FAF_SLD Directory is an SLD.</p> <p>Attribute flags are constructed by OR'ing the attribute flag bits.</p> <p>FAF_MLD is the only flag that may be modified without privilege if the directory is empty, the effective user ID of the process matches the directory owner, and the process has mandatory as well as discretionary write access. The FAF_MLD flag, once set, cannot be unset. Additionally, the FAF_MLD flag may only be set via the mldsetfattrflag interface. The FAF_PUBLIC flag can only be read or modified by a process possessing the PRIV_FILE_AUDIT privilege. A process attempting to read the FAF_PUBLIC flag without the PRIV_FILE_AUDIT privilege in effect will not fail. However the value of FAF_PUBLIC will be returned as unset. The FAF_SLD flag can never be set. The ability to read any flag is dependant upon the process having mandatory and discretionary read access to the file. The ability to set any flag is dependant upon the process having mandatory and discretionary write access to the file.</p>

If *path* is a symbolic link, the target's attribute flags are affected rather than the link's. If *path* is a multilevel directory, `getfattrflag()` and `setfattrflag()` will affect the underlying single-level directory beneath (unless *path* is adorned). `mldgetfattrflag()` and `mldsetfattrflag()` do not translate multi-level directories to underlying single-level directories. `fgetfattrflag()` and `fsetsattrflag()` affect only the file referred to by *filde*s.

RETURN VALUES

These functions return:

- 0 On success.
- 1 On failure, and set `errno` to indicate the error.

ERRORS

`getfattrflag()` and `mldgetfattrflag()` will fail if one or more of the following are true:

- EACCES** Search permission is denied on a component of the path prefix of *path*. To override this restriction, the calling process may assert the `PRIV_FILE_DAC_SEARCH` privilege and/or the `PRIV_FILE_MAC_SEARCH` privilege.
 - EACCES** Read permission is denied the final component of *path*. To override this restriction, the calling process may assert the `PRIV_FILE_MAC_READ` privilege.
 - EFAULT** *path* points to an illegal address.
 - EINTR** A signal was caught during execution of the function.
 - EIO** An I/O error occurred while reading from the file system.
 - ELOOP** Too many symbolic links were encountered in translating *path*.
 - EMULTIHOP** Components of *path* require hopping to multiple remote machines and file system type does not allow it.
 - ENAMETOOLONG** The length of the *path* argument exceeds `PATH_MAX`, or the length of a *path* component exceeds `NAME_MAX` while `_POSIX_NO_TRUNC` is in effect.
 - ENOENT** Either a component of the path prefix, or the file referred to by *path* does not exist or is a null pathname.
 - ENOLINK** *filde*s points to a remote machine and the link to that machine is no longer active.
 - ENOTDIR** A component of the prefix of *path* is not a directory.
- `fgetfattrflag()` fails and the file mode is unchanged if:
- EACCES** Read permission is denied on *filde*s. To override this restriction, the calling process may assert the `PRIV_FILE_MAC_READ` privilege.
 - EBADF** *filde*s is not an open file descriptor.

mldgetfattrflag(2)

EIO	An I/O error occurred while reading from the file system.
EINTR	A signal was caught during execution of the <code>fgetfattrflag()</code> function.
<code>setfattrflag()</code> and <code>mldsetfattrflag()</code> will fail and the file mode is unchanged if one or more of the following are true:	
EACCES	Search permission is denied on a component of the path prefix of <i>path</i> . To override this restriction, the calling process may assert the <code>PRIV_FILE_DAC_SEARCH</code> privilege and/or the <code>PRIV_FILE_MAC_SEARCH</code> privilege.
EACCES	Write permission is denied <i>path</i> . To override this restriction, the calling process may assert the <code>PRIV_FILE_MAC_WRITE</code> privilege.
EACCES	The calling process does not own <i>files</i> . To override this restriction, the calling process may assert the <code>PRIV_FILE_OWNER</code> privilege.
EFAULT	<i>path</i> points to an illegal address.
EINTR	A signal was caught during execution of the function.
EINVAL	<i>path</i> is not a valid pathname. When setting <code>FAF_MLD</code> , <i>path</i> must refer to an empty directory.
EIO	An I/O error occurred while writing to the file system.
ELOOP	Too many symbolic links were encountered in translating <i>path</i> .
EMULTIHOP	Components of <i>path</i> require hopping to multiple remote machines and filesystem type does not allow it.
ENAMETOOLONG	The length of the <i>path</i> argument exceeds <code>PATH_MAX</code> , or the length of a <i>path</i> component exceeds <code>NAME_MAX</code> while <code>_POSIX_NO_TRUNC</code> is in effect.
ENOENT	Either a component of the path prefix, or the file referred to by <i>path</i> does not exist or is a null pathname.
ENOLINK	<i>path</i> points to a remote machine and the link to that machine is no longer active.
ENOTDIR	A component of the prefix of <i>path</i> is not a directory.
EPERM	The effective user ID does not match the owner of the file and the process does not possess the privilege <code>PRIV_FILE_OWNER</code> .
EPERM	The process does not possess the privilege <code>PRIV_FILE_AUDIT</code> and is attempting to set the <code>FAF_PUBLIC</code> flag.
EROFS	The file referred to by <i>path</i> resides on a read-only file system.
<code>fsetfattrflag()</code> fails and the file mode is unchanged if:	

mldgetfattrflag(2)

EACCES	The calling process does not own <i>filde</i> s. To override this restriction, the calling process may assert the PRIV_FILE_OWNER privilege.
EACCES	Write access is denied on <i>filde</i> s. To override this restriction, the calling process may assert the PRIV_FILE_MAC_WRITE privilege.
EINVAL	<i>filde</i> s is not a valid pathname. When setting FAF_MLD, <i>filde</i> s must refer to an empty directory.
EBADF	<i>filde</i> s is not an open file descriptor.
EIO	An I/O error occurred while writing to the file system.
EINTR	A signal was caught during execution of the fsetfattrflag() function.
EPERM	The process does not possess the privilege PRIV_FILE_AUDIT and is attempting to set the FAF_PUBLIC flag.
EROFS	The file referred to by <i>filde</i> s resides on a read-only file system.

setfattrflag(1), getfattrflag(1)

Trusted Solaris Developer's Guide

mldsetfattrflag(2)

NAME	getfattrflag, fsetfattrflag, fgetfattrflag, setfattrflag, mldgetfattrflag, mldsetfattrflag – set/get the security attribute flags of a file
SYNOPSIS	<pre>cc [flags...] file... -ltso1 #include <tsol/secflgs.h> int getfattrflag(const char *path, secflgs_t *flags); int setfattrflag(const char *path, secflgs_t which, secflgs_t flags); int fgetfattrflag(int fildes, secflgs_t *flags); int fsetfattrflag(int fildes, secflgs_t which, secflgs_t flags); int mldgetfattrflag(const char *path, secflgs_t *flags); int mldsetfattrflag(const char *path, secflgs_t which, secflgs_t flags);</pre>
DESCRIPTION	<p>setfattrflag(), fsetfattrflag(), and mldsetfattrflag() set the security flags of the file whose name is given by <i>path</i> or referred to by the open file descriptor <i>fildes</i>. The bit pattern contained in <i>which</i> is used to indicate which flags are being affected. The corresponding bits in <i>flags</i> are set to 1 or 0 to indicate whether the affected flags are being set or unset respectively.</p> <p>getfattrflag(), fgetfattrflag(), and mldgetfattrflag() get the security flags of the file whose name is given by <i>path</i> or referred to by the open file descriptor <i>fildes</i> and store it in the location pointed to by <i>flags</i>.</p> <p>Attribute bits are interpreted as follows:</p> <p>FAF_MLD Directory has MLD semantics.</p> <p>FAF_PUBLIC Filesystem object is a public object.</p> <p>FAF_SLD Directory is an SLD.</p> <p>Attribute flags are constructed by OR'ing the attribute flag bits.</p> <p>FAF_MLD is the only flag that may be modified without privilege if the directory is empty, the effective user ID of the process matches the directory owner, and the process has mandatory as well as discretionary write access. The FAF_MLD flag, once set, cannot be unset. Additionally, the FAF_MLD flag may only be set via the mldsetfattrflag interface. The FAF_PUBLIC flag can only be read or modified by a process possessing the PRIV_FILE_AUDIT privilege. A process attempting to read the FAF_PUBLIC flag without the PRIV_FILE_AUDIT privilege in effect will not fail. However the value of FAF_PUBLIC will be returned as unset. The FAF_SLD flag can never be set. The ability to read any flag is dependant upon the process having mandatory and discretionary read access to the file. The ability to set any flag is dependant upon the process having mandatory and discretionary write access to the file.</p>

If *path* is a symbolic link, the target's attribute flags are affected rather than the link's. If *path* is a multilevel directory, `getfattrflag()` and `setfattrflag()` will affect the underlying single-level directory beneath (unless *path* is adorned). `mldgetfattrflag()` and `mldsetfattrflag()` do not translate multi-level directories to underlying single-level directories. `fgetfattrflag()` and `fsetsattrflag()` affect only the file referred to by *filde*s.

RETURN VALUES

These functions return:

- 0 On success.
- 1 On failure, and set `errno` to indicate the error.

ERRORS

`getfattrflag()` and `mldgetfattrflag()` will fail if one or more of the following are true:

- EACCES** Search permission is denied on a component of the path prefix of *path*. To override this restriction, the calling process may assert the `PRIV_FILE_DAC_SEARCH` privilege and/or the `PRIV_FILE_MAC_SEARCH` privilege.
 - EACCES** Read permission is denied the final component of *path*. To override this restriction, the calling process may assert the `PRIV_FILE_MAC_READ` privilege.
 - EFAULT** *path* points to an illegal address.
 - EINTR** A signal was caught during execution of the function.
 - EIO** An I/O error occurred while reading from the file system.
 - ELOOP** Too many symbolic links were encountered in translating *path*.
 - EMULTIHOP** Components of *path* require hopping to multiple remote machines and file system type does not allow it.
 - ENAMETOOLONG** The length of the *path* argument exceeds `PATH_MAX`, or the length of a *path* component exceeds `NAME_MAX` while `_POSIX_NO_TRUNC` is in effect.
 - ENOENT** Either a component of the path prefix, or the file referred to by *path* does not exist or is a null pathname.
 - ENOLINK** *filde*s points to a remote machine and the link to that machine is no longer active.
 - ENOTDIR** A component of the prefix of *path* is not a directory.
- `fgetfattrflag()` fails and the file mode is unchanged if:
- EACCES** Read permission is denied on *filde*s. To override this restriction, the calling process may assert the `PRIV_FILE_MAC_READ` privilege.
 - EBADF** *filde*s is not an open file descriptor.

mldsetfattrflag(2)

EIO	An I/O error occurred while reading from the file system.
EINTR	A signal was caught during execution of the <code>fgetfattrflag()</code> function.
<code>setfattrflag()</code> and <code>mldsetfattrflag()</code> will fail and the file mode is unchanged if one or more of the following are true:	
EACCES	Search permission is denied on a component of the path prefix of <i>path</i> . To override this restriction, the calling process may assert the <code>PRIV_FILE_DAC_SEARCH</code> privilege and/or the <code>PRIV_FILE_MAC_SEARCH</code> privilege.
EACCES	Write permission is denied <i>path</i> . To override this restriction, the calling process may assert the <code>PRIV_FILE_MAC_WRITE</code> privilege.
EACCES	The calling process does not own <i>files</i> . To override this restriction, the calling process may assert the <code>PRIV_FILE_OWNER</code> privilege.
EFAULT	<i>path</i> points to an illegal address.
EINTR	A signal was caught during execution of the function.
EINVAL	<i>path</i> is not a valid pathname. When setting <code>FAF_MLD</code> , <i>path</i> must refer to an empty directory.
EIO	An I/O error occurred while writing to the file system.
ELOOP	Too many symbolic links were encountered in translating <i>path</i> .
EMULTIHOP	Components of <i>path</i> require hopping to multiple remote machines and filesystem type does not allow it.
ENAMETOOLONG	The length of the <i>path</i> argument exceeds <code>PATH_MAX</code> , or the length of a <i>path</i> component exceeds <code>NAME_MAX</code> while <code>_POSIX_NO_TRUNC</code> is in effect.
ENOENT	Either a component of the path prefix, or the file referred to by <i>path</i> does not exist or is a null pathname.
ENOLINK	<i>path</i> points to a remote machine and the link to that machine is no longer active.
ENOTDIR	A component of the prefix of <i>path</i> is not a directory.
EPERM	The effective user ID does not match the owner of the file and the process does not possess the privilege <code>PRIV_FILE_OWNER</code> .
EPERM	The process does not possess the privilege <code>PRIV_FILE_AUDIT</code> and is attempting to set the <code>FAF_PUBLIC</code> flag.
EROFS	The file referred to by <i>path</i> resides on a read-only file system.
<code>fsetfattrflag()</code> fails and the file mode is unchanged if:	

mldsetattrflag(2)

EACCES	The calling process does not own <i>filde</i> s. To override this restriction, the calling process may assert the PRIV_FILE_OWNER privilege.
EACCES	Write access is denied on <i>filde</i> s. To override this restriction, the calling process may assert the PRIV_FILE_MAC_WRITE privilege.
EINVAL	<i>filde</i> s is not a valid pathname. When setting FAF_MLD, <i>filde</i> s must refer to an empty directory.
EBADF	<i>filde</i> s is not an open file descriptor.
EIO	An I/O error occurred while writing to the file system.
EINTR	A signal was caught during execution of the fsetattrflag() function.
EPERM	The process does not possess the privilege PRIV_FILE_AUDIT and is attempting to set the FAF_PUBLIC flag.
EROFS	The file referred to by <i>filde</i> s resides on a read-only file system.

Trusted Solaris 8
4/01 Reference
Manual

setattrflag(1), getattrflag(1)

Trusted Solaris Developer's Guide

mount(2)

NAME	mount – Mount a file system				
SYNOPSIS	<pre>#include <sys/types.h> #include <sys/mount.h> #include <sys/mntent.h> int mount(const char *spec, const char *dir, int mflag, char *fstype, char *dataptr, int datalen, char *optptr, int optlen);</pre>				
DESCRIPTION	<p>The <code>mount()</code> function requests that a removable file system contained on the block special file identified by <i>spec</i> be mounted on the directory identified by <i>dir</i>. The <i>spec</i> and <i>dir</i> arguments are pointers to path names. After a successful call to <code>mount()</code>, all references to the file <i>dir</i> refer to the root directory on the mounted file system. The mounted file system is inserted into the kernel list of all mounted file systems. This list can be examined through the mounted file system table (see <code>mnttab(4)</code>). The <i>fstype</i> argument is the file system type name. Standard file system names are defined with the prefix <code>MNTTYPE_</code> in <code><sys/mntent.h></code>. The <i>dataptr</i> argument is 0 if no file system-specific data is to be passed; otherwise it points to an area of size <i>datalen</i> that contains the file system-specific data for this mount and the <code>MS_DATA</code> flag should be set. If the <code>MS_OPTIONSTR</code> flag is set, then <i>optptr</i> points to a buffer containing the list of options to be used for this mount. The <i>optlen</i> argument specifies the length of the buffer. On completion of the <code>mount()</code> call, the options in effect for the mounted file system are returned in this buffer. If <code>MS_OPTIONSTR</code> is not specified, then the options for this mount will not appear in the mounted file systems table. The <i>mflag</i> argument is constructed by a bitwise-inclusive-OR of flags from the following list, defined in <code><sys/mount.h></code>.</p> <table> <tr> <td><code>MS_DATA</code></td><td>If this flag is set, the <i>dataptr</i> and <i>datalen</i> arguments describe a block of file system-specific binary data at address <i>dataptr</i> of length <i>datalen</i>. This is interpreted by file system-specific code within the operating system and its format depends on the file system type. If a particular file system type does not require this data, <i>dataptr</i> and <i>datalen</i> should both be 0.</td></tr> <tr> <td><code>MS_OPTIONSTR</code></td><td>If this flag is set, the <i>optptr</i> and <i>optlen</i> arguments describe a character buffer at address <i>optptr</i> of size <i>optlen</i>. When calling <code>mount()</code>, the character buffer should contain a null-terminated string of options to be passed to the file system-specific code within the operating system. On a successful return, the file system-specific code will return the list of options recognized. Unrecognized options are ignored. The format of the string is a list of option names separated by commas. Options that have values (rather than binary options such as <code>suid</code> or <code>nosuid</code>), are separated by "=" such as <code>dev=2c4046c</code>. Standard option names are defined in <code><sys/mntent.h></code>. The <i>slabel</i>, <i>low_range</i>, and <i>hi_range</i> values must be hexadecimal strings starting with 0x followed by exactly 68 hex digits. The allowed and forced values must be hexadecimal strings starting with 0x followed by exactly 32 hex digits. Only strings defined in the "C" locale are supported.</td></tr> </table>	<code>MS_DATA</code>	If this flag is set, the <i>dataptr</i> and <i>datalen</i> arguments describe a block of file system-specific binary data at address <i>dataptr</i> of length <i>datalen</i> . This is interpreted by file system-specific code within the operating system and its format depends on the file system type. If a particular file system type does not require this data, <i>dataptr</i> and <i>datalen</i> should both be 0.	<code>MS_OPTIONSTR</code>	If this flag is set, the <i>optptr</i> and <i>optlen</i> arguments describe a character buffer at address <i>optptr</i> of size <i>optlen</i> . When calling <code>mount()</code> , the character buffer should contain a null-terminated string of options to be passed to the file system-specific code within the operating system. On a successful return, the file system-specific code will return the list of options recognized. Unrecognized options are ignored. The format of the string is a list of option names separated by commas. Options that have values (rather than binary options such as <code>suid</code> or <code>nosuid</code>), are separated by "=" such as <code>dev=2c4046c</code> . Standard option names are defined in <code><sys/mntent.h></code> . The <i>slabel</i> , <i>low_range</i> , and <i>hi_range</i> values must be hexadecimal strings starting with 0x followed by exactly 68 hex digits. The allowed and forced values must be hexadecimal strings starting with 0x followed by exactly 32 hex digits. Only strings defined in the "C" locale are supported.
<code>MS_DATA</code>	If this flag is set, the <i>dataptr</i> and <i>datalen</i> arguments describe a block of file system-specific binary data at address <i>dataptr</i> of length <i>datalen</i> . This is interpreted by file system-specific code within the operating system and its format depends on the file system type. If a particular file system type does not require this data, <i>dataptr</i> and <i>datalen</i> should both be 0.				
<code>MS_OPTIONSTR</code>	If this flag is set, the <i>optptr</i> and <i>optlen</i> arguments describe a character buffer at address <i>optptr</i> of size <i>optlen</i> . When calling <code>mount()</code> , the character buffer should contain a null-terminated string of options to be passed to the file system-specific code within the operating system. On a successful return, the file system-specific code will return the list of options recognized. Unrecognized options are ignored. The format of the string is a list of option names separated by commas. Options that have values (rather than binary options such as <code>suid</code> or <code>nosuid</code>), are separated by "=" such as <code>dev=2c4046c</code> . Standard option names are defined in <code><sys/mntent.h></code> . The <i>slabel</i> , <i>low_range</i> , and <i>hi_range</i> values must be hexadecimal strings starting with 0x followed by exactly 68 hex digits. The allowed and forced values must be hexadecimal strings starting with 0x followed by exactly 32 hex digits. Only strings defined in the "C" locale are supported.				

mount(2)

The maximum length option string that can be passed to or returned from a `mount ()` call is defined by the `MAX_MNTOPT_STR` constant. The buffer should be long enough to contain more options than were passed in, as the state of any default options that were not passed in the input option string may also be returned in the recognized options list that is returned.

`MS_RDONLY` The file system should be mounted for reading only. This flag should also be specified for file systems that are incapable of writing (for example, CDROM). Without this flag, writing is permitted according to individual file accessibility.

`MS_NOSUID` This option prevents programs that are marked set-user-ID or set-group-ID from executing (see `chmod(1)`). It also causes `open(2)` to return `ENXIO` when attempting to open block or character special files.

`MS_REMOUNT` Remounts a read-only file system as read-write.

`MS_OVERLAY` Allow the file system to be mounted over an existing file system mounted on *dir*, making the underlying file system inaccessible. If a mount is attempted on a pre-existing mount point without setting this flag, the mount will fail.

`MS_GLOBAL` Mount a file system globally if the system is configured and booted as part of a cluster (see `clinfo(1M)`).

The `mount ()` system call may be invoked for all file system types except *namefs* by a calling process with the `PRIV_SYS_MOUNT` privilege. For the *namefs* file system, the calling process must either be the owner of *dir* or assert the `PRIV_FILE_OWNER` privilege. When mounting a UFS file system, the calling process should assert the `PRIV_SYS_FS_CONFIG` privilege. Otherwise, the mount succeeds, but logging is not enabled/disabled, `errno` is set to `EPERM`, and the user sees an error message.

RETURN VALUES

`mount ()` returns:

0 On success.

-1 On failure, and sets `errno` to indicate the error.

ERRORS

The `mount ()` function will fail if:

`EACCES` Search permission is denied on a component of *spec* or *dir*. To override this restriction, the calling process may assert one or both of these privileges:
`PRIV_FILE_DAC_SEARCH` and
`PRIV_FILE_MAC_SEARCH`.

Write permission is denied to the *namefs* file system specified in *dir*. To override this restriction, the calling process may assert one or both of these privileges:

mount(2)

	PRIV_FILE_DAC_WRITE and PRIV_FILE_MAC_WRITE.
EBUSY	The <i>dir</i> argument is currently mounted on, is someone's current working directory, or is otherwise busy; the device associated with <i>spec</i> is currently mounted; or there are no more mount table entries.
EFAULT	The <i>spec</i> , <i>dir</i> , <i>fstype</i> , or <i>dataptr</i> argument points outside the allocated address space of the process.
EINVAL	The super block has an invalid magic number or the <i>fstype</i> is invalid.
ELOOP	Too many symbolic links were encountered in translating <i>spec</i> or <i>dir</i> .
ENAMETOOLONG	The length of the <i>path</i> argument exceeds PATH_MAX, or the length of a <i>path</i> component exceeds NAME_MAX while _POSIX_NO_TRUNC is in effect.
ENOENT	None of the named files exists or is a null pathname.
ENOLINK	The <i>path</i> argument points to a remote machine and the link to that machine is no longer active.
ENOSPC	The file system state in the super block is not FSOKAY and <i>mflag</i> requests write permission.
ENOTBLK	The <i>spec</i> argument is not a block special device.
ENOTDIR	The <i>dir</i> argument is not a directory, or a component of a path prefix is not a directory.
ENOTSUP	A global mount is attempted (the MS_GLOBAL flag is set in <i>mflag</i>) on a machine which is not booted as a cluster or a local mount is attempted and <i>dir</i> is within a globally mounted file system.
ENXIO	The device associated with <i>spec</i> does not exist.
EOVERFLOW	The length of the option string to be returned in the <i>dataptr</i> argument exceeds the size of the buffer specified by <i>datalen</i> .
EPERM	The calling process does not own <i>dir</i> and <i>dir</i> is type <i>namefs</i> . To override this restriction, the calling process may assert the PRIV_FILE_OWNER privilege. <i>dir</i> is not a file system of type <i>namefs</i> and the calling process has not asserted the PRIV_SYS_MOUNT privilege.
EREMOTE	The <i>spec</i> argument is remote and cannot be mounted.

		mount(2)
	EROFS	The <i>spec</i> argument is write protected and <i>mflag</i> requests write permission.
		The mount () function will succeed, but logging will not be enabled/disabled if:
	EPERM	<i>dir</i> is a UFS file system and the calling process has not asserted the PRIV_SYS_FS_CONFIG privilege.
SUMMARY OF TRUSTED SOLARIS CHANGES		Appropriate privilege is required to override access, logging, or ownership checks.
		The mount () system call may be invoked for all file system types except <i>namefs</i> by a calling process with the PRIV_SYS_MOUNT privilege. For the <i>namefs</i> file system, the calling process must either be the owner of <i>dir</i> or assert the PRIV_FILE_OWNER privilege. When mounting a UFS file system, the calling process should assert the PRIV_SYS_FS_CONFIG privilege. Otherwise, the mount succeeds, but logging is not enabled/disabled and <i>errno</i> is set to EPERM.
Trusted Solaris 8 4/01 Reference Notes		mount(1M), umount(2), mnttab(4)
		MS_OPTIONSTR-type option strings should be used.
		Some flag bits set file system options that can also be passed in an option string. Options are first set from the option string with the last setting of an option in the string determining the value to be set by the option string. Any options controlled by flags are then applied, overriding any value set by the option string.

msgctl(2)

NAME	msgctl – Message control operations
SYNOPSIS	<pre>#include <sys/msg.h> int msgctl(int <i>msqid</i>, int <i>cmd</i>, struct msqid_ds *<i>buf</i>);</pre>
DESCRIPTION	<p>The <code>msgctl()</code> function provides a variety of message control operations as specified by <i>cmd</i>. The following <i>cmds</i> are available:</p> <p>IPC_STAT Place the current value of each member of the data structure associated with <i>msqid</i> into the structure pointed to by <i>buf</i>. The contents of this structure are defined in <code>intro(2)</code>.</p> <p>If it does not have discretionary read access to the data structure, the calling process must have <code>PRIV_IPC_DAC_READ</code> in its set of effective privileges. If it does not have mandatory read access to the data structure, the calling process must have <code>PRIV_IPC_MAC_READ</code> in its set of effective privileges.</p> <p>IPC_SET Set the value of the following members of the data structure associated with <i>msqid</i> to the corresponding value found in the structure pointed to by <i>buf</i>:</p> <pre>msg_perm.uid msg_perm.gid msg_perm.mode /* access permission bits only */ msg_qbytes</pre> <p>A process whose effective user ID does not match the value of <code>msg_perm.cuid</code> or <code>msg_perm.uid</code> must have the <code>PRIV_IPC_OWNER</code> privilege in its set of effective privileges. A process must have mandatory write access to the data structure or must have asserted the <code>PRIV_IPC_MAC_WRITE</code> privilege. Only a process with <code>PRIV_SYS_IPC_CONFIG</code> asserted can raise the value of <code>msg_qbytes</code>.</p> <p>IPC_RMID Remove from the system the message-queue identifier specified by <i>msqid</i> and destroy the message queue and data structure associated with it. This <i>cmd</i> can be executed only by a process that has an effective user ID equal to that of <code>msg_perm.cuid</code> or <code>msg_perm.uid</code> in the data structure associated with <i>msqid</i>, or has the <code>PRIV_IPC_OWNER</code> privilege asserted. A process must also have mandatory write access to the data structure or must have asserted the <code>PRIV_IPC_MAC_WRITE</code> privilege. <i>buf</i> is ignored.</p>
RETURN VALUES	<p><code>msgctl()</code> returns:</p> <p>0 On success.</p> <p>-1 On failure, and sets <code>errno</code> to indicate the error.</p>
ERRORS	The <code>msgctl()</code> function will fail if:

msgctl(2)

EACCES	<i>cmd</i> is IPC_STAT, operation permission is denied to the calling process (see intro(2)), and the calling process does not have the appropriate privilege(s) in its set of effective privileges.
EFAULT	The <i>buf</i> argument points to an illegal address.
EINVAL	The <i>msqid</i> argument is not a valid message queue identifier; or the <i>cmd</i> argument is not a valid command or is IPC_SET and <i>msg_perm.uid</i> or <i>msg_perm.gid</i> is not valid.
EPERM	<i>cmd</i> is IPC_RMID or IPC_SET, the discretionary and/or mandatory access checks failed, and the process did not have the appropriate override privilege asserted. <i>cmd</i> is IPC_SET, an attempt is being made to increase to the value of <i>msg_qbytes</i> , and the process did not have the appropriate override privilege asserted.
EOVERFLOW	The <i>cmd</i> argument is IPC_STAT and <i>uid</i> or <i>gid</i> is too large to be stored in the structure pointed to by <i>buf</i> .
	Appropriate privilege is required to override access checks.
	intro(2), msgget(2)

msgget(2)

NAME	msgget, msggetl – Get message queue
SYNOPSIS	<pre>#include <sys/msg.h> int msgget(key_t <i>key</i>, int <i>msgflg</i>); cc [<i>flags...</i>] <i>file...</i> -ltsol [<i>library...</i>] #include <sys/tso1/ipcl.h> int msggetl(key_t <i>key</i>, int <i>msgflg</i>, const bslabel_t <i>*slabel</i>);</pre>
DESCRIPTION	<p>A message queue is identified by a unique combination of key and sensitivity label. This qualification of keys by sensitivity labels allows applications that use message queues to be run at multiple process sensitivity labels without inadvertently sharing data.</p> <p><code>msgget()</code> returns the message-queue identifier associated with the union of <i>key</i> and the sensitivity label of the calling process.</p> <p><code>msggetl()</code> returns the message-queue identifier associated with the union of <i>key</i> and <i>slabel</i>. If the value of <i>slabel</i> does not match the sensitivity label of the calling process, then the effective privilege set of the process must contain <code>PRIV_IPC_MAC_READ</code> or <code>PRIV_IPC_MAC_WRITE</code>.</p> <p>If discretionary read/write access as specified by the low-order 9 bits of <i>msgflg</i> is denied to the calling process, <code>msgget()</code> and <code>msggetl()</code> require one or both of these privileges: <code>PRIV_IPC_DAC_READ</code> and <code>PRIV_IPC_DAC_WRITE</code>.</p> <p>A message-queue identifier and associated message queue and data structure (see <code>intro(2)</code>) are created for <i>key</i> if one of the following is true:</p> <ul style="list-style-type: none">■ <i>key</i> is <code>IPC_PRIVATE</code>.■ <i>key</i> does not already have a message queue identifier associated with it, and (<i>msgflg</i> & <code>IPC_CREAT</code>) is true. <p>On creation, the data structure associated with the new message queue identifier is initialized as follows:</p> <ul style="list-style-type: none">■ <code>msg_perm.cuid</code>, <code>msg_perm.uid</code>, <code>msg_perm.cgid</code>, and <code>msg_perm.gid</code> are set to the effective user ID and effective group ID, respectively, of the calling process.■ The low-order 9 bits of <code>msg_perm.mode</code> are set to the low-order 9 bits of <i>msgflg</i>.■ <code>msg_qnum</code>, <code>msg_lspid</code>, <code>msg_lrpid</code>, <code>msg_stime</code>, and <code>msg_rtime</code> are set to 0.■ <code>msg_ctime</code> is set to the current time.■ <code>msg_qbytes</code> is set to the system limit. <p>The sensitivity label on the message-queue internal is set either to the sensitivity label of the process or to <i>slabel</i>, depending on which interface was used.</p>

		msgget(2)
RETURN VALUES	Upon successful completion, a non-negative integer representing a message queue identifier is returned. Otherwise, -1 is returned and <code>errno</code> is set to indicate the error.	
ERRORS	The <code>msgget ()</code> function will fail if:	
	EACCES	A semaphore-structure identifier exists for the union of key and sensitivity label, but operation permission [see <code>intro(2)</code>] as specified by the low-order 9 bits of <code>semflg</code> would not be granted; or the sensitivity label check did not pass, and the calling process does not have the appropriate privilege override(s) in its set of effective privileges.
	EEXIST	A message queue identifier exists for <i>key</i> but (<code>msgflg&IPC_CREAT</code>) and (<code>msgflg&IPC_EXCL</code>) are both true.
	EFAULT	<i>slabel</i> points to an illegal address.
	EINVAL	The label to which <i>slabel</i> points is not a valid sensitivity label.
	ENOENT	A message queue identifier does not exist for the union of <i>key</i> and sensitivity label; and (<code>msgflg&IPC_CREAT</code>) is false.
	ENOSPC	A message queue identifier is to be created but the system-imposed limit on the maximum number of allowed message queue identifiers system wide would be exceeded.
SUMMARY OF TRUSTED SOLARIS CHANGES	Appropriate privilege is required to override access checks.	
Trusted Solaris 8 4/01 Reference Manual	Sensitivity labels are used together with <i>key</i> to determine message-queue identifiers.	
	<code>intro(2)</code> , <code>msgctl(2)</code>	
	<code>stdio(3C)</code>	

msggetl(2)

NAME	msgget, msggetl – Get message queue
SYNOPSIS	<pre>#include <sys/msg.h> int msgget(key_t <i>key</i>, int <i>msgflg</i>); cc [<i>flags...</i>] <i>file...</i> -ltsol [<i>library...</i>] #include <sys/tsol/ipcl.h> int msggetl(key_t <i>key</i>, int <i>msgflg</i>, const bslabel_t <i>*slabel</i>);</pre>
DESCRIPTION	<p>A message queue is identified by a unique combination of key and sensitivity label. This qualification of keys by sensitivity labels allows applications that use message queues to be run at multiple process sensitivity labels without inadvertently sharing data.</p> <p><code>msgget()</code> returns the message-queue identifier associated with the union of <i>key</i> and the sensitivity label of the calling process.</p> <p><code>msggetl()</code> returns the message-queue identifier associated with the union of <i>key</i> and <i>slabel</i>. If the value of <i>slabel</i> does not match the sensitivity label of the calling process, then the effective privilege set of the process must contain <code>PRIV_IPC_MAC_READ</code> or <code>PRIV_IPC_MAC_WRITE</code>.</p> <p>If discretionary read/write access as specified by the low-order 9 bits of <i>msgflg</i> is denied to the calling process, <code>msgget()</code> and <code>msggetl()</code> require one or both of these privileges: <code>PRIV_IPC_DAC_READ</code> and <code>PRIV_IPC_DAC_WRITE</code>.</p> <p>A message-queue identifier and associated message queue and data structure (see <code>intro(2)</code>) are created for <i>key</i> if one of the following is true:</p> <ul style="list-style-type: none">■ <i>key</i> is <code>IPC_PRIVATE</code>.■ <i>key</i> does not already have a message queue identifier associated with it, and (<i>msgflg</i> & <code>IPC_CREAT</code>) is true. <p>On creation, the data structure associated with the new message queue identifier is initialized as follows:</p> <ul style="list-style-type: none">■ <code>msg_perm.cuid</code>, <code>msg_perm.uid</code>, <code>msg_perm.cgid</code>, and <code>msg_perm.gid</code> are set to the effective user ID and effective group ID, respectively, of the calling process.■ The low-order 9 bits of <code>msg_perm.mode</code> are set to the low-order 9 bits of <i>msgflg</i>.■ <code>msg_qnum</code>, <code>msg_lspid</code>, <code>msg_lrpipd</code>, <code>msg_stime</code>, and <code>msg_rtime</code> are set to 0.■ <code>msg_ctime</code> is set to the current time.■ <code>msg_qbytes</code> is set to the system limit. <p>The sensitivity label on the message-queue internal is set either to the sensitivity label of the process or to <i>slabel</i>, depending on which interface was used.</p>

		msggetl(2)
RETURN VALUES	Upon successful completion, a non-negative integer representing a message queue identifier is returned. Otherwise, -1 is returned and <code>errno</code> is set to indicate the error.	
ERRORS	The <code>msgget ()</code> function will fail if:	
	EACCES	A semaphore-structure identifier exists for the union of key and sensitivity label, but operation permission [see <code>intro(2)</code>] as specified by the low-order 9 bits of <code>semflg</code> would not be granted; or the sensitivity label check did not pass, and the calling process does not have the appropriate privilege override(s) in its set of effective privileges.
	EEXIST	A message queue identifier exists for <i>key</i> but (<code>msgflg&IPC_CREAT</code>) and (<code>msgflg&IPC_EXCL</code>) are both true.
	EFAULT	<i>slabel</i> points to an illegal address.
	EINVAL	The label to which <i>slabel</i> points is not a valid sensitivity label.
	ENOENT	A message queue identifier does not exist for the union of <i>key</i> and sensitivity label; and (<code>msgflg&IPC_CREAT</code>) is false.
	ENOSPC	A message queue identifier is to be created but the system-imposed limit on the maximum number of allowed message queue identifiers system wide would be exceeded.
SUMMARY OF TRUSTED SOLARIS CHANGES	Appropriate privilege is required to override access checks.	
Trusted Solaris 8 4/01 Reference Manual	Sensitivity labels are used together with <i>key</i> to determine message-queue identifiers.	
	<code>intro(2)</code> , <code>msgctl(2)</code>	
	<code>stdio(3C)</code>	

msgrcv(2)

NAME	msgrcv – message receive operation
SYNOPSIS	<pre>#include <sys/msg.h> int msgrcv(int <i>msqid</i>, void *<i>msgp</i>, size_t <i>msgsz</i>, long <i>msgtyp</i>, int <i>msgflg</i>);</pre>
DESCRIPTION	<p>The <code>msgrcv()</code> function reads a message from the queue associated with the message queue identifier specified by <i>msqid</i> and places it in the user-defined buffer pointed to by <i>msgp</i>.</p> <p>The <i>msgp</i> argument points to a user-defined buffer that must contain first a field of type <code>long int</code> that will specify the type of the message, and then a data portion that will hold the data bytes of the message.</p> <p>The structure below is an example of what this user-defined buffer might look like:</p> <pre>struct mymsg { long mtype; /* message type */ char mtext[1]; /* message text */ }</pre> <p>The <i>mtype</i> member is the received message's type as specified by the sending process.</p> <p>The <i>mtext</i> member is the text of the message.</p> <p>The <i>msgsz</i> argument specifies the size in bytes of <i>mtext</i>. The received message is truncated to <i>msgsz</i> bytes if it is larger than <i>msgsz</i> and (<i>msgflg</i> & <code>MSG_NOERROR</code>) is non-zero. The truncated part of the message is lost and no indication of the truncation is given to the calling process.</p> <p>The <i>msgtyp</i> argument specifies the type of message requested as follows:</p> <ul style="list-style-type: none">■ If <i>msgtyp</i> is 0, the first message on the queue is received.■ If <i>msgtyp</i> is greater than 0, the first message of type <i>msgtyp</i> is received.■ If <i>msgtyp</i> is less than 0, the first message of the lowest type that is less than or equal to the absolute value of <i>msgtyp</i> is received. <p>The <i>msgflg</i> argument specifies which of the following actions is to be taken if a message of the desired type is not on the queue:</p> <ul style="list-style-type: none">■ If (<i>msgflg</i> & <code>IPC_NOWAIT</code>) is non-zero, the calling process will return immediately with a return value of -1 and <code>errno</code> set to <code>ENOMSG</code>.■ If (<i>msgflg</i> & <code>IPC_NOWAIT</code>) is 0, the calling process will suspend execution until one of the following occurs:<ul style="list-style-type: none">■ A message of the desired type is placed on the queue.■ The message queue identifier <i>msqid</i> is removed from the system (see <code>msgctl(2)</code>); when this occurs, <code>errno</code> is set equal to <code>EIDRM</code> and -1 is returned.

- The calling process receives a signal that is to be caught; in this case a message is not received and the calling process resumes execution in the manner prescribed in `sigaction(2)`.

Upon successful completion, the following actions are taken with respect to the data structure associated with *msqid* (see `intro(2)`):

- `msg_qnum` is decremented by 1.
- `msg_lrp_id` is set equal to the process ID of the calling process.
- `msg_rtime` is set equal to the current time.

RETURN VALUES

Upon successful completion, `msgrcv()` returns a value equal to the number of bytes actually placed into the buffer *mtext*. Otherwise, `-1` is returned, no message is received, and `errno` is set to indicate the error.

ERRORS

The `msgrcv()` function will fail if:

- | | |
|---------------------|--|
| <code>E2BIG</code> | The value of <i>mtext</i> is greater than <i>msgsz</i> and (<i>msgflg</i> & <code>MSG_NOERROR</code>) is 0. |
| <code>EACCES</code> | Operation permission is denied to the calling process, and the calling process does not have the appropriate privilege override(s) in its set of effective privileges. See <code>intro(2)</code> . |
| <code>EIDRM</code> | The sensitivity label of <i>msqid</i> does not match the sensitivity label of the calling process, and the calling process does not have the appropriate privilege override(s) in its set of effective privileges. |
| <code>EINTR</code> | The message queue identifier <i>msqid</i> is removed from the system. |
| <code>EINVAL</code> | The <code>msgrcv()</code> function was interrupted by a signal. |
| <code>EINVAL</code> | The <i>msqid</i> argument is not a valid message queue identifier; or the value of <i>msgsz</i> is less than 0. |
| <code>ENOMSG</code> | The queue does not contain a message of the desired type and (<i>msgflg</i> & <code>IPC_NOWAIT</code>) is non-zero. |

The `msgrcv()` function may fail if:

- | | |
|---------------------|--|
| <code>EFAULT</code> | The <i>msgp</i> argument points to an illegal address. |
|---------------------|--|

USAGE

The value passed as the *msgp* argument should be converted to type `void *`.

SUMMARY OF TRUSTED SQUARES

Trusted Squares
4/01 Reference Manual
SUNOS 5.10
SunOS 5.10
Reference Manual

Appropriate privilege is required to override access checks.

`intro(2)`, `msgctl(2)`, `msgget(2)`, `msgsnd(2)`

`sigaction(2)`, `signal(3C)`

msgsnd(2)

NAME	msgsnd – message send operation
SYNOPSIS	<pre>#include <sys/msg.h> int msgsnd(int <i>msqid</i>, const void *<i>msgp</i>, size_t <i>msgsz</i>, int <i>msgflg</i>);</pre>
DESCRIPTION	<p>The <code>msgsnd()</code> function is used to send a message to the queue associated with the message queue identifier specified by <i>msqid</i>.</p> <p>The <i>msgp</i> argument points to a user-defined buffer that must contain first a field of type <code>long int</code> that will specify the type of the message, and then a data portion that will hold the data bytes of the message.</p> <p><code>msgsnd()</code> requires either that a process have discretionary and mandatory write access to <i>msqid</i>, or that the effective privilege set of the calling process includes <code>PRIV_IPC_DAC_WRITE</code> and <code>PRIV_IPC_MAC_WRITE</code>.</p> <p>The structure below is an example of what this user-defined buffer might look like:</p> <pre>struct mymsg { long mtype; /* message type */ char mtext[1]; /* message text */ }</pre> <p>The <code>mtype</code> member is a non-zero positive type <code>long int</code> that can be used by the receiving process for message selection.</p> <p>The <code>mtext</code> member is any text of length <i>msgsz</i> bytes. The <i>msgsz</i> argument can range from 0 to a system-imposed maximum.</p> <p>The <i>msgflg</i> argument specifies the action to be taken if one or more of the following are true:</p> <ul style="list-style-type: none">■ The number of bytes already on the queue is equal to <code>msg_qbytes</code>; see <code>intro(2)</code>.■ The total number of messages on all queues system-wide is equal to the system-imposed limit. <p>These actions are as follows:</p> <ul style="list-style-type: none">■ If (<code>msgflg & IPC_NOWAIT</code>) is non-zero, the message will not be sent and the calling process will return immediately.■ If (<code>msgflg & IPC_NOWAIT</code>) is 0, the calling process will suspend execution until one of the following occurs:<ul style="list-style-type: none">■ The condition responsible for the suspension no longer exists, in which case the message is sent.■ The message queue identifier <i>msqid</i> is removed from the system (see <code>msgctl(2)</code>); when this occurs, <code>errno</code> is set equal to <code>EIDRM</code> and <code>-1</code> is returned.■ The calling process receives a signal that is to be caught; in this case the message is not sent and the calling process resumes execution in the manner prescribed in <code>sigaction(2)</code>.

msgsnd(2)

Upon successful completion, the following actions are taken with respect to the data structure associated with *msqid* (see *intro(2)*):

- *msg_qnum* is incremented by 1.
- *msg_lspid* is set equal to the process ID of the calling process.
- *msg_stime* is set equal to the current time.

RETURN VALUES

msgsnd() returns:

- 0 On success.
- 1 On failure, and sets *errno* to indicate the error.

ERRORS

The *msgsnd()* function will fail if:

- EACCES Operation permission is denied to the calling process, and the process did not have the appropriate privilege in its set of effective privileges. See *intro(2)*.

The sensitivity label of *msqid* does not match the sensitivity label of the calling process, and the calling process does not have the appropriate privilege override(s) in its set of effective privileges.
- EAGAIN The message cannot be sent for one of the reasons cited above and (*msgflg*&*IPC_NOWAIT*) is non-zero.
- EIDRM The message queue identifier *msqid* is removed from the system.
- EINTR The *msgsnd()* function was interrupted by a signal.
- EINVAL The value of *msqid* is not a valid message queue identifier, or the value of *mtype* is less than 1; or the value of *msgsz* is less than 0 or greater than the system-imposed limit.

The *msgsnd()* function may fail if:

- EFAULT The *msgp* argument points to an illegal address.

USAGE

The value passed as the *msgp* argument should be converted to type *void **.

SUMMARY OF
TRUSTED

Appropriate privilege is required to override access checks.

intro(2), *msgctl(2)*, *msgget(2)*, *msgrcv(2)*

sigaction(2), *signal(3C)*

nice(2)

NAME	nice – Change priority of a process				
SYNOPSIS	<pre>#include <unistd.h> int nice(int incr);</pre>				
DESCRIPTION	<p>The <code>nice()</code> function allows a process to change its priority. The invoking process must be in a scheduling class that supports the <code>nice()</code>.</p> <p>The <code>nice()</code> function adds the value of <code>incr</code> to the nice value of the calling process. A process's nice value is a non-negative number for which a greater positive value results in lower CPU priority.</p> <p>A maximum nice value of $(2 * \text{NZERO}) - 1$ and a minimum nice value of 0 are imposed by the system. <code>NZERO</code> is defined in <code><limits.h></code> with a default value of 20. Requests for values above or below these limits result in the nice value being set to the corresponding limit. A nice value of 40 is treated as 39.</p>				
RETURN VALUES	<p>Upon successful completion, <code>nice()</code> returns the new nice value minus <code>NZERO</code>. Otherwise, <code>-1</code> is returned, the process's nice value is not changed, and <code>errno</code> is set to indicate the error.</p>				
ERRORS	<p>The <code>nice()</code> function will fail if:</p> <table><tr><td><code>EINVAL</code></td><td>The <code>nice()</code> function is called by a process in a scheduling class other than time-sharing.</td></tr><tr><td><code>EPERM</code></td><td><code>incr</code> is negative or greater than 40 and the <code>PRIV_SYS_CONFIG</code> privilege of the calling process is not asserted.</td></tr></table>	<code>EINVAL</code>	The <code>nice()</code> function is called by a process in a scheduling class other than time-sharing.	<code>EPERM</code>	<code>incr</code> is negative or greater than 40 and the <code>PRIV_SYS_CONFIG</code> privilege of the calling process is not asserted.
<code>EINVAL</code>	The <code>nice()</code> function is called by a process in a scheduling class other than time-sharing.				
<code>EPERM</code>	<code>incr</code> is negative or greater than 40 and the <code>PRIV_SYS_CONFIG</code> privilege of the calling process is not asserted.				
USAGE	<p>The <code>prctl(2)</code> function is a more general interface to scheduler functions.</p> <p>Since <code>-1</code> is a permissible return value in a successful situation, an application wishing to check for error situations should set <code>errno</code> to 0, then call <code>nice()</code>, and if it returns <code>-1</code>, check to see if <code>errno</code> is non-zero.</p>				
SUMMARY OF TRUSTED SQUARES CHANGES	Use of the <code>PRIV_SYS_CONFIG</code> privilege replaces the check for super-user.				
Trusted Squares 4/01 Reference Manual	<code>exec(2)</code> , <code>prctl(2)</code>				
Source	<code>nice(1)</code>				

NAME	open – Open a file										
SYNOPSIS	<pre>#include <sys/types.h> #include <sys/stat.h> #include <fcntl.h> int open(const char *path, int oflag, /* mode_t mode */...);</pre>										
DESCRIPTION	<p>The <code>open()</code> function establishes the connection between a file and a file descriptor. It creates an open file description that refers to a file and a file descriptor that refers to that open file description. The file descriptor is used by other I/O functions to refer to that file. The <i>path</i> argument points to a pathname naming the file.</p> <p>The <code>open()</code> function returns a file descriptor for the named file that is the lowest file descriptor not currently open for that process. The open file description is new, and therefore the file descriptor does not share it with any other process in the system. The <code>FD_CLOEXEC</code> file descriptor flag associated with the new file descriptor is cleared.</p> <p>The file offset used to mark the current position within the file is set to the beginning of the file.</p> <p>The file status flags and file access modes of the open file description is set according to the value of <i>oflag</i>.</p> <p>Values for <i>oflag</i> are constructed by a bitwise-inclusive-OR of flags from the following list, defined in <code><fcntl.h></code>. Applications must specify exactly one of the first three values (file access modes) below in the value of <i>oflag</i>:</p> <table> <tr> <td><code>O_RDONLY</code></td><td>Open for reading only.</td></tr> <tr> <td><code>O_WRONLY</code></td><td>Open for writing only.</td></tr> <tr> <td><code>O_RDWR</code></td><td>Open for reading and writing. The result is undefined if this flag is applied to a FIFO.</td></tr> </table> <p>Any combination of the following may be used:</p> <table> <tr> <td><code>O_APPEND</code></td><td>If set, the file offset is set to the end of the file prior to each write.</td></tr> <tr> <td><code>O_CREAT</code></td><td>If the file exists, this flag has no effect except as noted under <code>O_EXCL</code>. Otherwise, the file is created and the owner ID of the file is set to the effective user ID of the process; the group ID of the file is set to the effective group ID of the process; or if the <code>S_ISGID</code> bit is set in the directory in which the file is being created, the file's group ID is set to the group ID of its parent directory. If the group ID of the new file does not match the effective group ID or one of the supplementary groups IDs, the <code>S_ISGID</code> bit is cleared. The calling process must assert the <code>PRIV_FILE_SETID</code> privilege to override clearing the <code>S_ISGID</code> bit. The access permission bits of the file mode are set to the value of <i>mode</i>, modified as follows: [See <code>creat(2)</code>.]</td></tr> </table>	<code>O_RDONLY</code>	Open for reading only.	<code>O_WRONLY</code>	Open for writing only.	<code>O_RDWR</code>	Open for reading and writing. The result is undefined if this flag is applied to a FIFO.	<code>O_APPEND</code>	If set, the file offset is set to the end of the file prior to each write.	<code>O_CREAT</code>	If the file exists, this flag has no effect except as noted under <code>O_EXCL</code> . Otherwise, the file is created and the owner ID of the file is set to the effective user ID of the process; the group ID of the file is set to the effective group ID of the process; or if the <code>S_ISGID</code> bit is set in the directory in which the file is being created, the file's group ID is set to the group ID of its parent directory. If the group ID of the new file does not match the effective group ID or one of the supplementary groups IDs, the <code>S_ISGID</code> bit is cleared. The calling process must assert the <code>PRIV_FILE_SETID</code> privilege to override clearing the <code>S_ISGID</code> bit. The access permission bits of the file mode are set to the value of <i>mode</i> , modified as follows: [See <code>creat(2)</code> .]
<code>O_RDONLY</code>	Open for reading only.										
<code>O_WRONLY</code>	Open for writing only.										
<code>O_RDWR</code>	Open for reading and writing. The result is undefined if this flag is applied to a FIFO.										
<code>O_APPEND</code>	If set, the file offset is set to the end of the file prior to each write.										
<code>O_CREAT</code>	If the file exists, this flag has no effect except as noted under <code>O_EXCL</code> . Otherwise, the file is created and the owner ID of the file is set to the effective user ID of the process; the group ID of the file is set to the effective group ID of the process; or if the <code>S_ISGID</code> bit is set in the directory in which the file is being created, the file's group ID is set to the group ID of its parent directory. If the group ID of the new file does not match the effective group ID or one of the supplementary groups IDs, the <code>S_ISGID</code> bit is cleared. The calling process must assert the <code>PRIV_FILE_SETID</code> privilege to override clearing the <code>S_ISGID</code> bit. The access permission bits of the file mode are set to the value of <i>mode</i> , modified as follows: [See <code>creat(2)</code> .]										

open(2)

	<ul style="list-style-type: none">■ All bits set in the file mode-creation mask of the process are cleared. [See <code>umask(2)</code>.]■ The “save text image after execution bit” of the mode is cleared. [See <code>chmod(2)</code> .] <code>O_SYNC</code> write I/O operations on the file descriptor complete as defined by synchronized I/O file integrity completion. [See <code>fcntl(5)</code> definition of <code>O_SYNC</code> .]■ The calling process must assert the <code>PRIV_SYS_CONFIG</code> privilege to override clearing the <code>S_ISVTX</code> bit.
<code>O_DSYNC</code>	Write I/O operations on the file descriptor complete as defined by synchronized I/O data integrity completion.
<code>O_EXCL</code>	If <code>O_CREAT</code> and <code>O_EXCL</code> are set, <code>open()</code> fails if the file exists. The check for the existence of the file and the creation of the file if it does not exist is atomic with respect to other processes executing <code>open()</code> naming the same filename in the same directory with <code>O_EXCL</code> and <code>O_CREAT</code> set. If <code>O_CREAT</code> is not set, the effect is undefined.
<code>O_LARGEFILE</code>	If set, the offset maximum in the open file description is the largest value that can be represented correctly in an object of type <code>off64_t</code> .
<code>O_NOCTTY</code>	If set and <i>path</i> identifies a terminal device, <code>open()</code> does not cause the terminal device to become the controlling terminal for the process.
<code>O_NONBLOCK</code> or <code>O_NDELAY</code>	<p>These flags may affect subsequent reads and writes (see <code>read(2)</code> and <code>write(2)</code>). If both <code>O_NDELAY</code> and <code>O_NONBLOCK</code> are set, <code>O_NONBLOCK</code> takes precedence.</p> <p>When opening a FIFO with <code>O_RDONLY</code> or <code>O_WRONLY</code> set:</p> <p>If <code>O_NONBLOCK</code> or <code>O_NDELAY</code> is set: An <code>open()</code> for reading only returns without delay. An <code>open()</code> for writing only returns an error if no process currently has the file open for reading.</p> <p>If <code>O_NONBLOCK</code> and <code>O_NDELAY</code> are clear: An <code>open()</code> for reading only blocks until a process opens the file for writing. An <code>open()</code> for writing only blocks until a process opens the file for reading.</p> <p>After both ends of a FIFO have been opened, there is no guarantee that further calls to <code>open()</code> <code>O_RDONLY</code> (<code>O_WRONLY</code>) will synchronize with later calls to <code>open()</code> <code>O_WRONLY</code> (<code>O_RDONLY</code>) until both ends of the FIFO have been closed by all readers and writers. Any data written into a FIFO will be lost if both ends of the FIFO are closed before the data is read.</p> <p>When opening a block special or character special file that supports non-blocking opens:</p>

If `O_NONBLOCK` or `O_NDELAY` is set:

The `open()` function returns without blocking for the device to be ready or available. Subsequent behavior of the device is device-specific.

If `O_NONBLOCK` and `O_NDELAY` are clear:

The `open()` function blocks until the device is ready or available before returning.

Otherwise, the behavior of `O_NONBLOCK` and `O_NDELAY` is unspecified.

<code>O_RSYNC</code>	Read I/O operations on the file descriptor complete at the same level of integrity as specified by the <code>O_DSYNC</code> and <code>O_SYNC</code> flags. If both <code>O_DSYNC</code> and <code>O_RSYNC</code> are set in <i>oflag</i> , all I/O operations on the file descriptor complete as defined by synchronized I/O data integrity completion. If both <code>O_SYNC</code> and <code>O_RSYNC</code> are set in <i>oflag</i> , all I/O operations on the file descriptor complete as defined by synchronized I/O file integrity completion.
<code>O_SYNC</code>	When opening a regular file, this flag affects subsequent writes. If set, each <code>write(2)</code> will wait for both the file data and file status to be physically updated. Write I/O operations on the file descriptor complete as defined by synchronized I/O file integrity completion.
<code>O_TRUNC</code>	If the file exists and is a regular file, and the file is successfully opened <code>O_RDWR</code> or <code>O_WRONLY</code> , its length is truncated to 0 and the mode and owner are unchanged. It has no effect on FIFO special files or terminal device files. Its effect on other file types is implementation-dependent. The result of using <code>O_TRUNC</code> with <code>O_RDONLY</code> is undefined.

If `O_CREAT` is set and the file did not previously exist, upon successful completion, `open()` marks for update the `st_atime`, `st_ctime`, and `st_mtime` fields of the file and the `st_ctime` and `st_mtime` fields of the parent directory.

If `O_TRUNC` is set and the file did previously exist, upon successful completion, `open()` marks for update the `st_ctime` and `st_mtime` fields of the file.

If *path* refers to a STREAMS file, *oflag* may be constructed from `O_NONBLOCK` or `O_NDELAY` OR-ed with either `O_RDONLY`, `O_WRONLY`, or `O_RDWR`. Other flag values are not applicable to STREAMS devices and have no effect on them. The values `O_NONBLOCK` and `O_NDELAY` affect the operation of STREAMS drivers and certain functions (see `read(2)` and `write(2)`) applied to file descriptors associated with STREAMS files. For STREAMS drivers, the implementation of `O_NONBLOCK` and `O_NDELAY` is device-specific.

When `open()` is invoked to open a named stream, and the `connld` module (see `connld(7M)`) has been pushed on the pipe, `open()` blocks until the server process has issued an `I_RECVFD ioctl()` (see `streamio(7I)`) to receive the file descriptor.

open(2)

If *path* names the master side of a pseudo-terminal device, then it is unspecified whether `open()` locks the slave side so that it cannot be opened. Portable applications must call `unlockpt(3C)` before opening the slave side.

If *path* is a symbolic link and `O_CREAT` and `O_EXCL` are set, the link is not followed.

Certain flag values can be set following `open()` as described in `fcntl(2)`.

The largest value that can be represented correctly in an object of type `off_t` is established as the offset maximum in the open file description.

RETURN VALUES

Upon successful completion, the `open()` function opens the file and return a non-negative integer representing the lowest numbered unused file descriptor. Otherwise, `-1` is returned, `errno` is set to indicate the error, and no files are created or modified.

ERRORS

The `open()` function will fail if:

EACCES	Search permission is denied on a component of the path prefix, or the file exists and the permissions specified by <i>oflag</i> are denied, or the file does not exist and write permission is denied for the parent directory of the file to be created, or <code>O_TRUNC</code> is specified and write permission is denied.
EDQUOT	The file does not exist, <code>O_CREAT</code> is specified, and either the directory where the new file entry is being placed cannot be extended because the user's quota of disk blocks on that file system has been exhausted, or the user's quota of inodes on the file system where the file is being created has been exhausted.
EEXIST	The <code>O_CREAT</code> and <code>O_EXCL</code> flags are set, and the named file exists.
EINTR	A signal was caught during <code>open()</code> .
EFAULT	The <i>path</i> argument points to an illegal address.
EIO	The <i>path</i> argument names a STREAMS file and a hangup or error occurred during the <code>open()</code> .
EISDIR	The named file is a directory and <i>oflag</i> includes <code>O_WRONLY</code> or <code>O_RDWR</code> .
ELOOP	Too many symbolic links were encountered in resolving <i>path</i> .
EMFILE	The process has too many open files. (See <code>getrlimit(2)</code> .)
EMULTIHOP	Components of <i>path</i> require hopping to multiple remote machines and the file system does not allow it.
ENAMETOOLONG	The length of the <i>path</i> argument exceeds <code>PATH_MAX</code> or a pathname component is longer than <code>NAME_MAX</code> .
ENFILE	The maximum allowable number of files is currently open in the system.

ENOENT	The <code>O_CREAT</code> flag is not set and the named file does not exist; or the <code>O_CREAT</code> flag is set and either the path prefix does not exist or the <i>path</i> argument points to an empty string.
ENOLINK	The <i>path</i> argument points to a remote machine, and the link to that machine is no longer active.
ENOSR	The <i>path</i> argument names a STREAMS-based file and the system is unable to allocate a STREAM.
ENOSPC	The directory or file system that would contain the new file cannot be expanded, the file does not exist, and <code>O_CREAT</code> is specified.
ENOTDIR	A component of the path prefix is not a directory.
ENXIO	The <code>O_NONBLOCK</code> flag is set, the named file is a FIFO, the <code>O_WRONLY</code> flag is set, and no process has the file open for reading; or the named file is a character special or block special file and the device associated with this special file does not exist.
EOPNOTSUPP	An attempt was made to open a path that corresponds to a <code>AF_UNIX</code> socket.
EOVERFLOW	The named file is a regular file and either <code>O_LARGEFILE</code> is not set and the size of the file cannot be represented correctly in an object of type <code>off_t</code> or <code>O_LARGEFILE</code> is set and the size of the file cannot be represented correctly in an object of type <code>off64_t</code> .
EROFS	The named file resides on a read-only file system and either <code>O_WRONLY</code> , <code>O_RDWR</code> , <code>O_CREAT</code> (if file does not exist), or <code>O_TRUNC</code> is set in the <i>oflag</i> argument.
The <code>open()</code> function may fail if:	
EAGAIN	The <i>path</i> argument names the slave side of a pseudo-terminal device that is locked.
EINVAL	The value of the <i>oflag</i> argument is not valid.
ENAMETOOLONG	Pathname resolution of a symbolic link produced an intermediate result whose length exceeds <code>PATH_MAX</code> .
ENOMEM	The <i>path</i> argument names a STREAMS file and the system is unable to allocate resources.
ETXTBSY	The file is a pure procedure (shared text) file that is being executed and <i>oflag</i> is <code>O_WRONLY</code> or <code>O_RDWR</code> .

USAGE The `open()` function has a transitional interface for 64-bit file offsets. See `lf64(5)`. Note that using `open64()` is equivalent to using `open()` with `O_LARGEFILE` set in *oflag*.

ATTRIBUTES See `attributes(5)` for descriptions of the following attributes:

open(2)

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Async-Signal-Safe

**SUMMARY OF
TRUSTED
SOLARIS
CHANGES**

Appropriate privilege is required to override access checks.

To open a file system object that supports exclusive open or exclusive access, the calling process may assert the `PRIV_SYS_DEVICES` privilege. In the case of *procfs*, the calling process cannot open a process whose program file has the `S_ISUID` or `S_ISGID` mode bits set or has the use of privilege. The calling process may assert the `PRIV_PROC_OWNER` privilege. When used to create a new file, the calling process may need to assert one or both of these privileges: `PRIV_SYS_CONFIG` to override clearing the `S_ISVTX` bit, and `PRIV_FILE_SETID` to override clearing the `S_ISGID` bit.

**Trusted Solaris 8
4/01 Reference
Manual**

`intro(2)`, `chmod(2)`, `creat(2)`, `exec(2)`, `fcntl(2)`, `getrlimit(2)`, `lseek(2)`, `read(2)`, `stat(2)`, `write(2)`

**SunOS 5.8
Reference Manual**

`close(2)`, `dup(2)`, `getmsg(2)`, `putmsg(2)`, `umask(2)`, `fcntl(5)`, `stat(5)`, `connlld(7M)`, `streamio(7I)`

NOTES

Hierarchical Storage Management (HSM) file systems can sometimes cause long delays when opening a file, since HSM files must be recalled from secondary storage.

NAME

SYNOPSIS

DESCRIPTION

fpathconf, pathconf – Get configurable pathname variables

#include <unistd.h>

long int **fpathconf**(int *fil*des, int *name*) ;

long int **pathconf**(const char **path*, int *name*) ;

The `fpathconf()` and `pathconf()` functions provide a method for the application to determine the current value of a configurable limit or option I (variable) that is associated with a file or directory.

For `pathconf()` , the *path* argument points to the pathname of a file or directory.

For `fpathconf()` , the *fil*des argument is an open file descriptor.

The *name* argument represents the variable to be queried relative to that file or directory. The variables in the following table come from <limits.h> or <unistd.h> and the symbolic constants, defined in <unistd.h>, are the corresponding values used for *name*:

Variable	Value of <i>name</i>	Notes
FILESIZEBITS	_PC_FILESIZEBITS	3,4
LINK_MAX	_PC_LINK_MAX	1
MAX_CANON	_PC_MAX_CANON	2
MAX_INPUT	_PC_MAX_INPUT	2
NAME_MAX	_PC_NAME_MAX	3,4
PATH_MAX	_PC_PATH_MAX	4,5
PIPE_BUF	_PC_PIPE_BUF	6
_POSIX_CHOWN_RESTRICTED	_PC_CHOWN_RESTRICTED	7
_POSIX_NO_TRUNC	_PC_NO_TRUNC	3,4
_POSIX_VDISABLE	_PC_VDISABLE	2
_POSIX_ASYNC_IO	_PC_ASYNC_IO	8
_POSIX_PRIO_IO	_PC_PRIO_IO	8
_POSIX_SYNC_IO	_PC_SYNC_IO	8

Notes:

1. If *path* or *fil*des refers to a directory, the value returned applies to the directory itself.
2. If *path* or *fil*des does not refer to a terminal file, it is unspecified whether an implementation supports an association of the variable name with the specified

pathconf(2)

file.

3. If *path* or *fildes* refers to a directory, the value returned applies to filenames within the directory.
4. If *path* or *fildes* does not refer to a directory, it is unspecified whether an implementation supports an association of the variable name with the specified file.
5. If *path* or *fildes* refers to a directory, the value returned is the maximum length of a relative pathname when the specified directory is the working directory.
6. If *path* refers to a FIFO, or *fildes* refers to a pipe or FIFO, the value returned applies to the referenced object. If *path* or *fildes* refers to a directory, the value returned applies to any FIFO that exists or can be created within the directory. If *path* or *fildes* refers to any other type of file, it is unspecified whether an implementation supports an association of the variable name with the specified file.
7. If *path* or *fildes* refers to a directory, the value returned applies to any files, other than directories, that exist or can be created within the directory.
8. If *path* or *fildes* refers to a directory, it is unspecified whether an implementation supports an association of the variable name with the specified file.

RETURN VALUES

If *name* is an invalid value, both `pathconf()` and `fpathconf()` return `-1` and `errno` is set to indicate the error.

If the variable corresponding to *name* has no limit for the *path* or file descriptor, both `pathconf()` and `fpathconf()` return `-1` without changing `errno`. If the implementation needs to use *path* to determine the value of *name* and the implementation does not support the association of *name* with the file specified by *path*, or if the process did not have appropriate privileges to query the *appropriate privileges* file specified by *path*, or *path* does not exist, `pathconf()` returns `-1` and `errno` is set to indicate the error.

If the implementation needs to use *fildes* to determine the value of *name* and the implementation does not support the association of *name* with the file specified by *fildes*, or if *fildes* is an invalid file descriptor, `fpathconf()` will return `-1` and `errno` is set to indicate the error.

Otherwise `pathconf()` or `fpathconf()` returns the current variable value for the file or directory without changing `errno`. The value returned will not be more restrictive than the corresponding value available to the application when it was compiled with the implementation's `<limits.h>` or `<unistd.h>`.

ERRORS

The `pathconf()` function will fail if:

EINVAL

The value of *name* is not valid.

ELOOP

Too many symbolic links were encountered in resolving *path*.

The `pathconf()` function may fail if:

EACCES	Search permission is denied for a component of the path prefix.
EINVAL	The implementation does not support an association of the variable <i>name</i> with the specified file.
ENAMETOOLONG	The length of the <i>path</i> argument exceeds PATH_MAX or a pathname component is longer than NAME_MAX.
ENAMETOOLONG	Pathname resolution of a symbolic link produced an intermediate result whose length exceeds PATH_MAX.
ENOENT	A component of <i>path</i> does not name an existing file or <i>path</i> is an empty string.
ENOTDIR	A component of the path prefix is not a directory.

The fpathconf () function will fail if:

EINVAL	The value of <i>name</i> is not valid.
EACCES	<i>fildev</i> is open only for writing and the calling process does not have mandatory read access to the object to which the descriptor refers. To override this restriction, the calling process may assert the PRIV_FILE_MAC_READ privilege.

The fpathconf () function may fail if:

EACCES	Search permission is denied for a component of the path prefix. To override this restriction, the calling process may assert one or both of these privileges: PRIV_FILE_DAC_SEARCH and PRIV_FILE_MAC_SEARCH. The calling process does not have mandatory read access to <i>path</i> . To override this restriction, the calling process may assert the PRIV_FILE_MAC_READ privilege.
EBADF	The <i>fildev</i> argument is not a valid file descriptor.
EINVAL	The implementation does not support an association of the variable <i>name</i> with the specified file.

ATTRIBUTES See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	pathconf () is Async-Signal-Safe

pathconf(2)

**SUMMARY OF
TRUSTED
SOURCES
CHANGES
Reference Manual**

Appropriate privilege is required to override access checks.

sysconf(3C), limits(4), attributes(5), standards(5)

NAME	p_online – Return or change processor operational status		
SYNOPSIS	<pre>#include <sys/types.h> #include <sys/processor.h> int p_online(processorid_t processorid, int flag);</pre>		
DESCRIPTION	<p>The <code>p_online()</code> function changes or returns the operational status of processors. The state of the processor specified by the <i>processorid</i> argument is changed to the state represented by the <i>flag</i> argument.</p> <p>Legal values for <i>flag</i> are <code>P_STATUS</code>, <code>P_ONLINE</code>, <code>P_OFFLINE</code>, and <code>P_NOINTR</code>.</p> <p>When <i>flag</i> is <code>P_STATUS</code>, no processor status change occurs, but the current processor status is returned.</p> <p>The <code>P_ONLINE</code>, <code>P_OFFLINE</code>, and <code>P_NOINTR</code> values for <i>flag</i> refer to valid processor states. A processor in the <code>P_ONLINE</code> state is allowed to process LWPs (lightweight processes) and perform system activities. The processor is also interruptible by I/O devices attached to the system. The <code>PRIV_SYS_CONFIG</code> privilege is required.</p> <p>A processor in the <code>P_OFFLINE</code> state is not allowed to process LWPs. The processor is as inactive as possible. If the hardware supports such a feature, the processor is not interruptible by attached I/O devices. The <code>PRIV_SYS_CONFIG</code> privilege is required.</p> <p>A processor in the <code>P_NOINTR</code> state is allowed to process LWPs, but it is not interruptible by attached I/O devices. Typically, interrupts, when they occur are routed to other processors in the system. Not all systems support putting a processor into the <code>P_NOINTR</code> state. It is not permitted to put all the processors of a system into the <code>P_NOINTR</code> state. At least one processor must always be available to service system clock interrupts.</p> <p>Processor numbers are integers, greater than or equal to 0, and are defined by the hardware platform. Processor numbers are not necessarily contiguous, but “not too sparse.” Processor numbers should always be printed in decimal.</p> <p>The number of processors present can be determined by calling <code>sysconf(_SC_NPROCESSORS_CONF)</code>. The list of valid processor numbers can be determined by calling <code>p_online()</code> with <i>processorid</i> values starting at 0 until all processors have been found. The <code>EINVAL</code> error is returned for invalid processor numbers. See <code>EXAMPLES</code> below.</p>		
RETURN VALUES	On successful completion, the value returned is the previous state of the processor, <code>P_ONLINE</code> , <code>P_OFFLINE</code> , <code>P_NOINTR</code> , or <code>P_POWEROFF</code> . Otherwise, <code>-1</code> is returned and <code>errno</code> is set to indicate the error.		
ERRORS	<p>The <code>p_online()</code> function will fail if:</p> <table> <tr> <td><code>EPERM</code></td><td>The calling process does not have the <code>PRIV_SYS_CONFIG</code> privilege.</td></tr> </table>	<code>EPERM</code>	The calling process does not have the <code>PRIV_SYS_CONFIG</code> privilege.
<code>EPERM</code>	The calling process does not have the <code>PRIV_SYS_CONFIG</code> privilege.		

p_online(2)

EINVAL	A non-existent processor ID was specified or <i>flag</i> was invalid.
EBUSY	The <i>flag</i> was P_OFFLINE and the specified processor is the only on-line processor, there are currently LWPs bound to the processor, or the processor performs some essential function that cannot be performed by another processor.
EBUSY	The <i>flag</i> was P_NOINTR and the specified processor is the only interruptible processor in the system, or it handles interrupts that cannot be handled by another processor.
EBUSY	The specified processor is powered off and cannot be powered on because some platform-specific resource is not available.
ENOTSUP	The specified processor is powered off, and the platform does not support power on of individual processors.

EXAMPLES

EXAMPLE 1 List the legal processor numbers.

The following code sample will list the legal processor numbers:

```
#include <sys/unistd.h>
#include <sys/processor.h>
#include <sys/types.h>
#include <stdio.h>
#include <errno.h>

int
main()
{
    processorid_t i;
    int    status;
    int    n = sysconf(_SC_NPROCESSORS_ONLN);
    for (i = 0; n > 0; i++) {
        status = p_online(i, P_STATUS);
        if (status == -1 && errno == EINVAL)
            continue;
        printf("processor %d present\n", i);
        n--;
    }
    return (0);
}
```

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	MT-Safe

SUMMARY OF TRUSTED SOLARIS CHANGES

The calling process must have the PRIV_SYS_CONFIG privilege in order to perform the P_ONLINE and P_OFFLINE operations.

psradm(1M), processor_bind(2)
psrinfo(1M), processor_info(2), pset_create(2), sysconf(3C),
attributes(5)
p_online(2)

pread(2)

NAME	read, readl, pread, preadl, readv, readvl – read from a file
SYNOPSIS	<pre>#include <sys/types.h> #include <sys/uio.h> #include <unistd.h> ssize_t read(int fildes, void *buf, size_t nbyte); ssize_t pread(int fildes, void *buf, size_t nbyte, off_t offset); ssize_t readv(int fildes, struct iovec *iov, int iovcnt); #include <tsol/rdwrl.h> ssize_t readl(int fildes, void *buf, size_t nbyte, bclabel_t *label_p); ssize_t preadl(int fildes, void *buf, size_t nbyte, off_t offset, bclabel_t *label_p); ssize_t readvl(int fildes, struct iovec *iov, int iovcnt, bclabel_t *label_p);</pre>
DESCRIPTION	<p>The <code>read()</code> function attempts to read <i>nbyte</i> bytes from the file associated with the open file descriptor, <i>fildes</i>, into the buffer pointed to by <i>buf</i>.</p> <p>If <i>nbyte</i> is 0, <code>read()</code> will return 0 and have no other results.</p> <p>On files that support seeking (for example, a regular file), the <code>read()</code> starts at a position in the file given by the file offset associated with <i>fildes</i>. The file offset is incremented by the number of bytes actually read.</p> <p>Files that do not support seeking (for example, terminals) always read from the current position. The value of a file offset associated with such a file is undefined.</p> <p>If <i>fildes</i> refers to a socket, <code>read()</code> is equivalent to <code>recv(3SOCKET)</code> with no flags set.</p> <p>No data transfer will occur past the current end-of-file. If the starting position is at or after the end-of-file, 0 will be returned. If the file refers to a device special file, the result of subsequent <code>read()</code> requests is implementation-dependent.</p> <p>If the value of <i>nbyte</i> is greater than <code>SSIZE_MAX</code>, the result is implementation-dependent.</p> <p>When attempting to read from a regular file with mandatory file/record locking set (see <code>chmod(2)</code>), and there is a write lock owned by another process on the segment of the file to be read:</p> <ul style="list-style-type: none">■ If <code>O_NDELAY</code> or <code>O_NONBLOCK</code> is set, <code>read()</code> returns -1 and sets <code>errno</code> to <code>EAGAIN</code>.■ If <code>O_NDELAY</code> and <code>O_NONBLOCK</code> are clear, <code>read()</code> sleeps until the blocking record lock is removed. <p>When attempting to read from an empty pipe (or FIFO):</p>

pread(2)

- If no process has the pipe open for writing, `read()` returns 0 to indicate end-of-file.
- If some process has the pipe open for writing and `O_NDELAY` is set, `read()` returns 0.
- If some process has the pipe open for writing and `O_NONBLOCK` is set, `read()` returns -1 and sets `errno` to `EAGAIN`.
- If `O_NDELAY` and `O_NONBLOCK` are clear, `read()` blocks until data is written to the pipe or the pipe is closed by all processes that had opened the pipe for writing.

When attempting to read a file associated with a terminal that has no data currently available:

- If `O_NDELAY` is set, `read()` returns 0.
- If `O_NONBLOCK` is set, `read()` returns -1 and sets `errno` to `EAGAIN`.
- If `O_NDELAY` and `O_NONBLOCK` are clear, `read()` blocks until data become available.

When attempting to read a file associated with a socket or a stream that is not a pipe, a FIFO, or a terminal, and the file has no data currently available:

- If `O_NDELAY` or `O_NONBLOCK` is set, `read()` returns -1 and sets `errno` to `EAGAIN`.
- If `O_NDELAY` and `O_NONBLOCK` are clear, `read()` blocks until data becomes available.

The `read()` function reads data previously written to a file. If any portion of a regular file prior to the end-of-file has not been written, `read()` returns bytes with value 0. For example, `lseek(2)` allows the file offset to be set beyond the end of existing data in the file. If data is later written at this point, subsequent reads in the gap between the previous end of data and the newly written data will return bytes with value 0 until data is written into the gap.

For regular files, no data transfer will occur past the offset maximum established in the open file description associated with *fil*des.

Upon successful completion, where *nbyte* is greater than 0, `read()` will mark for update the `st_atime` field of the file, and return the number of bytes read. This number will never be greater than *nbyte*. The value returned may be less than *nbyte* if the number of bytes left in the file is less than *nbyte*, if the `read()` request was interrupted by a signal, or if the file is a pipe or FIFO or special file and has fewer than *nbyte* bytes immediately available for reading. For example, a `read()` from a file associated with a terminal may return one typed line of data.

If a `read()` is interrupted by a signal before it reads any data, it will return -1 with `errno` set to `EINTR`.

pread(2)

If a `read()` is interrupted by a signal after it has successfully read some data, it will return the number of bytes read.

A `read()` or `readv()` from a STREAMS (see `intro(2)`) file can read data in three different modes: byte-stream mode, message-nondiscard mode, and message-discard mode. The default is byte-stream mode. This can be changed using the `I_SRDOPT ioctl(2)` request, and can be tested with the `I_GRDOPT ioctl()`. In byte-stream mode, `read()` retrieves data from the STREAM until as many bytes as were requested are transferred, or until there is no more data to be retrieved. Byte-stream mode ignores message boundaries.

In STREAMS message-nondiscard mode, `read()` retrieves data until as many bytes as were requested are transferred, or until a message boundary is reached. If `read()` does not retrieve all the data in a message, the remaining data is left on the STREAM, and can be retrieved by the next `read()` call. Message-discard mode also retrieves data until as many bytes as were requested are transferred, or a message boundary is reached. However, unread data remaining in a message after the `read()` returns is discarded, and is not available for a subsequent `read()`, `readv()` or `getmsg(2)` call.

How `read()` handles zero-byte STREAMS messages is determined by the current read mode setting. In byte-stream mode, `read()` accepts data until it has read *nbyte* bytes, or until there is no more data to read, or until a zero-byte message block is encountered. The `read()` function then returns the number of bytes read, and places the zero-byte message back on the STREAM to be retrieved by the next `read()`, `readv()` or `getmsg(2)`. In message-nondiscard mode or message-discard mode, a zero-byte message returns 0 and the message is removed from the STREAM. When a zero-byte message is read as the first message on a STREAM, the message is removed from the STREAM and 0 is returned, regardless of the read mode.

A `read()` from a STREAMS file returns the data in the message at the front of the STREAM head read queue, regardless of the priority band of the message.

By default, STREAMS are in control-normal mode, in which a `read()` from a STREAMS file can only process messages that contain a data part but do not contain a control part. The `read()` fails if a message containing a control part is encountered at the STREAM head. This default action can be changed by placing the STREAM in either control-data mode or control-discard mode with the `I_SRDOPT ioctl()` command. In control-data mode, `read()` converts any control part to data and passes it to the application before passing any data part originally present in the same message. In control-discard mode, `read()` discards message control parts but returns to the process any data part in the message.

`readl()`, `preadl()`, and `readvl()` perform the same actions as `read()`, `pread()`, and `readv()`, respectively, and additionally return in *label_p* the CMW label of the data read. The label returned is determined according to these conditions:

- If the descriptor refers to a regular file or FIFO, the sensitivity label portion of *label_p* is set to the sensitivity label associated with the filesystem object.

In all other respects, the `readl()`, `preadl()`, and `readvl()` interfaces are analogous to the `read()`, `pread()`, and `readv()` interfaces.

In the Solaris environment, `read()` normally allows a process to read the contents of directories on some local file systems. This functionality is not supported in the Trusted Solaris operating environment. If the file descriptor refers to a directory, `read()` will return `EISDIR`.

The last access time is updated only when the calling process has both mandatory read and write access to the filesystem object. There is no privilege to override this restriction.

In addition, `read()` and `readv()` will fail if the STREAM head had processed an asynchronous error before the call. In this case, the value of `errno` does not reflect the result of `read()` or `readv()` but reflects the prior error. If a hangup occurs on the STREAM being read, `read()` continues to operate normally until the STREAM head read queue is empty. Thereafter, it returns 0.

`readv()` The `readv()` function is equivalent to `read()`, but places the input data into the *iovcnt* buffers specified by the members of the *iov* array: *iov0*, *iov1*, ..., *iov[iovcnt-1]*. The *iovcnt* argument is valid if greater than 0 and less than or equal to `IOV_MAX`.

The `iovec` structure contains the following members:

```
caddr_t    iov_base;
int         iov_len;
```

Each `iovec` entry specifies the base address and length of an area in memory where data should be placed. The `readv()` function always fills an area completely before proceeding to the next.

Upon successful completion, `readv()` marks for update the `st_atime` field of the file.

`pread()` The `pread()` function performs the same action as `read()`, except that it reads from a given position in the file without changing the file pointer. The first three arguments to `pread()` are the same as `read()` with the addition of a fourth argument *offset* for the desired position inside the file. `pread()` will read up to the maximum offset value that can be represented in an `off_t` for regular files. An attempt to perform a `pread()` on a file that is incapable of seeking results in an error.

RETURN VALUES Upon successful completion, `read()` and `readv()` return a non-negative integer indicating the number of bytes actually read. Otherwise, the functions return `-1` and set `errno` to indicate the error.

ERRORS `read()`, `readl()`, `pread()`, `preadl()`, `readv()`, and `readvl()` fail if any of these conditions is true:

<code>EAGAIN</code>	Mandatory file/record locking was set, <code>O_NDELAY</code> or <code>O_NONBLOCK</code> was set, and there was a blocking record lock; total
---------------------	--

pread(2)

	amount of system memory available when reading using raw I/O is temporarily insufficient; no data is waiting to be read on a file associated with a tty device and <code>O_NONBLOCK</code> was set; or no message is waiting to be read on a stream and <code>O_NDELAY</code> or <code>O_NONBLOCK</code> was set.
EBADF	The <i>fil</i> des argument is not a valid file descriptor open for reading.
EBADMSG	Message waiting to be read on a stream is not a data message.
EDEADLK	The read was going to go to sleep and cause a deadlock to occur.
EFAULT	The <i>buf</i> argument points to an illegal address.
EINTR	A signal was caught during the read operation and no data was transferred.
EINVAL	An attempt was made to read from a stream linked to a multiplexor.
EIO	A physical I/O error has occurred, or the process is in a background process group and is attempting to read from its controlling terminal, and either the process is ignoring or blocking the SIGTTIN signal or the process group of the process is orphaned.
EISDIR	The <i>fil</i> des argument refers to a directory on a file system type that does not support read operations on directories.
ENOLCK	The system record lock table was full, so the <code>read()</code> or <code>readv()</code> could not go to sleep until the blocking record lock was removed.
ENOLINK	The <i>fil</i> des argument is on a remote machine and the link to that machine is no longer active.
ENXIO	The device associated with <i>fil</i> des is a block special or character special file and the value of the file pointer is out of range.
The <code>read()</code> and <code>readv()</code> functions will fail if:	
EOVERFLOW	The file is a regular file, <i>nbyte</i> is greater than 0, the starting position is before the end-of-file, and the starting position is greater than or equal to the offset maximum established in the open file description associated with <i>fil</i> des.
The <code>readv()</code> function may fail if:	
EFAULT	The <i>iov</i> argument points outside the allocated address space.
EINVAL	The <i>iovcnt</i> argument was less than or equal to 0, or greater than or equal to <code>IOV_MAX</code> . (See <code>intro(2)</code> for a definition of <code>IOV_MAX</code>).
EINVAL	The sum of the <i>iov_len</i> values in the <i>iov</i> array overflowed an int.
The <code>pread()</code> function will fail and the file pointer remain unchanged if:	

EFAULT *label_p* points to an illegal address.

USAGE The `pread()` function has a transitional interface for 64-bit file offsets. See `lf64(5)`.

ATTRIBUTES See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	<code>read()</code> is Async-Signal-Safe

SUMMARY OF TRUSTED SOLARIS CHANGES `readl()`, `preadl()`, and `readvl()` return in the buffer that was referenced by *label_p*, the CMW label associated with the data that was read.

The last access time is updated only when the calling process has both mandatory read and write access to the filesystem object. There is no privilege to override this restriction.

For conduits, a sensitivity label is associated with each byte of data.

Trusted Solaris 8
4/01 Reference
Manual
SunOS 5.9
Reference Manual

`intro(2)`, `chmod(2)`, `creat(2)`, `fcntl(2)`, `open(2)`

`dup(2)`, `getmsg(2)`, `ioctl(2)`, `pipe(2)`, `attributes(5)`, `streamio(7I)`, `termio(7I)`

preadl(2)

NAME	read, readl, pread, preadl, readv, readvl – read from a file
SYNOPSIS	<pre>#include <sys/types.h> #include <sys/uio.h> #include <unistd.h> ssize_t read(int fildes, void *buf, size_t nbyte); ssize_t pread(int fildes, void *buf, size_t nbyte, off_t offset); ssize_t readv(int fildes, struct iovec *iov, int iovcnt); #include <tsol/rdwrl.h> ssize_t readl(int fildes, void *buf, size_t nbyte, bclabel_t *label_p); ssize_t preadl(int fildes, void *buf, size_t nbyte, off_t offset, bclabel_t *label_p); ssize_t readvl(int fildes, struct iovec *iov, int iovcnt, bclabel_t *label_p);</pre>
DESCRIPTION	<p>The <code>read()</code> function attempts to read <i>nbyte</i> bytes from the file associated with the open file descriptor, <i>fildes</i>, into the buffer pointed to by <i>buf</i>.</p> <p>If <i>nbyte</i> is 0, <code>read()</code> will return 0 and have no other results.</p> <p>On files that support seeking (for example, a regular file), the <code>read()</code> starts at a position in the file given by the file offset associated with <i>fildes</i>. The file offset is incremented by the number of bytes actually read.</p> <p>Files that do not support seeking (for example, terminals) always read from the current position. The value of a file offset associated with such a file is undefined.</p> <p>If <i>fildes</i> refers to a socket, <code>read()</code> is equivalent to <code>recv(3SOCKET)</code> with no flags set.</p> <p>No data transfer will occur past the current end-of-file. If the starting position is at or after the end-of-file, 0 will be returned. If the file refers to a device special file, the result of subsequent <code>read()</code> requests is implementation-dependent.</p> <p>If the value of <i>nbyte</i> is greater than <code>SSIZE_MAX</code>, the result is implementation-dependent.</p> <p>When attempting to read from a regular file with mandatory file/record locking set (see <code>chmod(2)</code>), and there is a write lock owned by another process on the segment of the file to be read:</p> <ul style="list-style-type: none">■ If <code>O_NDELAY</code> or <code>O_NONBLOCK</code> is set, <code>read()</code> returns -1 and sets <code>errno</code> to <code>EAGAIN</code>.■ If <code>O_NDELAY</code> and <code>O_NONBLOCK</code> are clear, <code>read()</code> sleeps until the blocking record lock is removed. <p>When attempting to read from an empty pipe (or FIFO):</p>

- If no process has the pipe open for writing, `read()` returns 0 to indicate end-of-file.
- If some process has the pipe open for writing and `O_NDELAY` is set, `read()` returns 0.
- If some process has the pipe open for writing and `O_NONBLOCK` is set, `read()` returns -1 and sets `errno` to `EAGAIN`.
- If `O_NDELAY` and `O_NONBLOCK` are clear, `read()` blocks until data is written to the pipe or the pipe is closed by all processes that had opened the pipe for writing.

When attempting to read a file associated with a terminal that has no data currently available:

- If `O_NDELAY` is set, `read()` returns 0.
- If `O_NONBLOCK` is set, `read()` returns -1 and sets `errno` to `EAGAIN`.
- If `O_NDELAY` and `O_NONBLOCK` are clear, `read()` blocks until data become available.

When attempting to read a file associated with a socket or a stream that is not a pipe, a FIFO, or a terminal, and the file has no data currently available:

- If `O_NDELAY` or `O_NONBLOCK` is set, `read()` returns -1 and sets `errno` to `EAGAIN`.
- If `O_NDELAY` and `O_NONBLOCK` are clear, `read()` blocks until data becomes available.

The `read()` function reads data previously written to a file. If any portion of a regular file prior to the end-of-file has not been written, `read()` returns bytes with value 0. For example, `lseek(2)` allows the file offset to be set beyond the end of existing data in the file. If data is later written at this point, subsequent reads in the gap between the previous end of data and the newly written data will return bytes with value 0 until data is written into the gap.

For regular files, no data transfer will occur past the offset maximum established in the open file description associated with *fil*des.

Upon successful completion, where *nbyte* is greater than 0, `read()` will mark for update the `st_atime` field of the file, and return the number of bytes read. This number will never be greater than *nbyte*. The value returned may be less than *nbyte* if the number of bytes left in the file is less than *nbyte*, if the `read()` request was interrupted by a signal, or if the file is a pipe or FIFO or special file and has fewer than *nbyte* bytes immediately available for reading. For example, a `read()` from a file associated with a terminal may return one typed line of data.

If a `read()` is interrupted by a signal before it reads any data, it will return -1 with `errno` set to `EINTR`.

preadl(2)

If a `read()` is interrupted by a signal after it has successfully read some data, it will return the number of bytes read.

A `read()` or `readv()` from a STREAMS (see `intro(2)`) file can read data in three different modes: byte-stream mode, message-nondiscard mode, and message-discard mode. The default is byte-stream mode. This can be changed using the `I_SRDOPT ioctl(2)` request, and can be tested with the `I_GRDOPT ioctl()`. In byte-stream mode, `read()` retrieves data from the STREAM until as many bytes as were requested are transferred, or until there is no more data to be retrieved. Byte-stream mode ignores message boundaries.

In STREAMS message-nondiscard mode, `read()` retrieves data until as many bytes as were requested are transferred, or until a message boundary is reached. If `read()` does not retrieve all the data in a message, the remaining data is left on the STREAM, and can be retrieved by the next `read()` call. Message-discard mode also retrieves data until as many bytes as were requested are transferred, or a message boundary is reached. However, unread data remaining in a message after the `read()` returns is discarded, and is not available for a subsequent `read()`, `readv()` or `getmsg(2)` call.

How `read()` handles zero-byte STREAMS messages is determined by the current read mode setting. In byte-stream mode, `read()` accepts data until it has read *nbyte* bytes, or until there is no more data to read, or until a zero-byte message block is encountered. The `read()` function then returns the number of bytes read, and places the zero-byte message back on the STREAM to be retrieved by the next `read()`, `readv()` or `getmsg(2)`. In message-nondiscard mode or message-discard mode, a zero-byte message returns 0 and the message is removed from the STREAM. When a zero-byte message is read as the first message on a STREAM, the message is removed from the STREAM and 0 is returned, regardless of the read mode.

A `read()` from a STREAMS file returns the data in the message at the front of the STREAM head read queue, regardless of the priority band of the message.

By default, STREAMS are in control-normal mode, in which a `read()` from a STREAMS file can only process messages that contain a data part but do not contain a control part. The `read()` fails if a message containing a control part is encountered at the STREAM head. This default action can be changed by placing the STREAM in either control-data mode or control-discard mode with the `I_SRDOPT ioctl()` command. In control-data mode, `read()` converts any control part to data and passes it to the application before passing any data part originally present in the same message. In control-discard mode, `read()` discards message control parts but returns to the process any data part in the message.

`readl()`, `preadl()`, and `readvl()` perform the same actions as `read()`, `pread()`, and `readv()`, respectively, and additionally return in *label_p* the CMW label of the data read. The label returned is determined according to these conditions:

- If the descriptor refers to a regular file or FIFO, the sensitivity label portion of *label_p* is set to the sensitivity label associated with the filesystem object.

In all other respects, the `readl()`, `preadl()`, and `readvl()` interfaces are analogous to the `read()`, `pread()`, and `readv()` interfaces.

In the Solaris environment, `read()` normally allows a process to read the contents of directories on some local file systems. This functionality is not supported in the Trusted Solaris operating environment. If the file descriptor refers to a directory, `read()` will return `EISDIR`.

The last access time is updated only when the calling process has both mandatory read and write access to the filesystem object. There is no privilege to override this restriction.

In addition, `read()` and `readv()` will fail if the STREAM head had processed an asynchronous error before the call. In this case, the value of `errno` does not reflect the result of `read()` or `readv()` but reflects the prior error. If a hangup occurs on the STREAM being read, `read()` continues to operate normally until the STREAM head read queue is empty. Thereafter, it returns 0.

`readv()` The `readv()` function is equivalent to `read()`, but places the input data into the *iovcnt* buffers specified by the members of the *iov* array: *iov0*, *iov1*, ..., *iov[iovcnt-1]*. The *iovcnt* argument is valid if greater than 0 and less than or equal to `IOV_MAX`.

The `iovec` structure contains the following members:

```
caddr_t    iov_base;
int         iov_len;
```

Each `iovec` entry specifies the base address and length of an area in memory where data should be placed. The `readv()` function always fills an area completely before proceeding to the next.

Upon successful completion, `readv()` marks for update the `st_atime` field of the file.

`pread()` The `pread()` function performs the same action as `read()`, except that it reads from a given position in the file without changing the file pointer. The first three arguments to `pread()` are the same as `read()` with the addition of a fourth argument *offset* for the desired position inside the file. `pread()` will read up to the maximum offset value that can be represented in an `off_t` for regular files. An attempt to perform a `pread()` on a file that is incapable of seeking results in an error.

RETURN VALUES Upon successful completion, `read()` and `readv()` return a non-negative integer indicating the number of bytes actually read. Otherwise, the functions return `-1` and set `errno` to indicate the error.

ERRORS `read()`, `readl()`, `pread()`, `preadl()`, `readv()`, and `readvl()` fail if any of these conditions is true:

<code>EAGAIN</code>	Mandatory file/record locking was set, <code>O_NDELAY</code> or <code>O_NONBLOCK</code> was set, and there was a blocking record lock; total
---------------------	--

preadl(2)

	amount of system memory available when reading using raw I/O is temporarily insufficient; no data is waiting to be read on a file associated with a tty device and <code>O_NONBLOCK</code> was set; or no message is waiting to be read on a stream and <code>O_NDELAY</code> or <code>O_NONBLOCK</code> was set.
EBADF	The <i>fil</i> des argument is not a valid file descriptor open for reading.
EBADMSG	Message waiting to be read on a stream is not a data message.
EDEADLK	The read was going to go to sleep and cause a deadlock to occur.
EFAULT	The <i>buf</i> argument points to an illegal address.
EINTR	A signal was caught during the read operation and no data was transferred.
EINVAL	An attempt was made to read from a stream linked to a multiplexor.
EIO	A physical I/O error has occurred, or the process is in a background process group and is attempting to read from its controlling terminal, and either the process is ignoring or blocking the SIGTTIN signal or the process group of the process is orphaned.
EISDIR	The <i>fil</i> des argument refers to a directory on a file system type that does not support read operations on directories.
ENOLCK	The system record lock table was full, so the <code>read()</code> or <code>readv()</code> could not go to sleep until the blocking record lock was removed.
ENOLINK	The <i>fil</i> des argument is on a remote machine and the link to that machine is no longer active.
ENXIO	The device associated with <i>fil</i> des is a block special or character special file and the value of the file pointer is out of range.
The <code>read()</code> and <code>readv()</code> functions will fail if:	
EOVERFLOW	The file is a regular file, <i>nbyte</i> is greater than 0, the starting position is before the end-of-file, and the starting position is greater than or equal to the offset maximum established in the open file description associated with <i>fil</i> des.
The <code>readv()</code> function may fail if:	
EFAULT	The <i>iov</i> argument points outside the allocated address space.
EINVAL	The <i>iovcnt</i> argument was less than or equal to 0, or greater than or equal to <code>IOV_MAX</code> . (See <code>intro(2)</code> for a definition of <code>IOV_MAX</code>).
EINVAL	The sum of the <i>iov_len</i> values in the <i>iov</i> array overflowed an int.
The <code>pread()</code> function will fail and the file pointer remain unchanged if:	

EFAULT *label_p* points to an illegal address.

USAGE The `pread()` function has a transitional interface for 64-bit file offsets. See `lf64(5)`.

ATTRIBUTES See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	<code>read()</code> is Async-Signal-Safe

SUMMARY OF TRUSTED SOLARIS CHANGES

`readl()`, `preadl()`, and `readvl()` return in the buffer that was referenced by *label_p*, the CMW label associated with the data that was read.

The last access time is updated only when the calling process has both mandatory read and write access to the filesystem object. There is no privilege to override this restriction.

For conduits, a sensitivity label is associated with each byte of data.

Trusted Solaris 8 4/01 Reference Manual

`intro(2)`, `chmod(2)`, `creat(2)`, `fcntl(2)`, `open(2)`

`dup(2)`, `getmsg(2)`, `ioctl(2)`, `pipe(2)`, `attributes(5)`, `streamio(7I)`, `termio(7I)`

priocntl(2)

NAME	priocntl – process scheduler control										
SYNOPSIS	<pre>#include <sys/types.h> #include <sys/priocntl.h> #include <sys/rtpriocntl.h> #include <sys/tspriocntl.h> long priocntl(idtype_t <i>idtype</i>, id_t <i>id</i>, int <i>cmd</i>, /* arg */ ...);</pre>										
DESCRIPTION	<p>The <code>priocntl()</code> function provides for control over the scheduling of an active light weight process (LWP).</p> <p> LWPs fall into distinct classes with a separate scheduling policy applied to each class. The two classes currently supported are the realtime class and the time-sharing class. The characteristics of these classes are described under the corresponding headings below. The class attribute of an LWP is inherited across the <code>fork(2)</code> and <code>_lwp_create(2)</code> functions and the <i>exec</i> family of functions (see <code>exec(2)</code>). The <code>priocntl()</code> function can be used to dynamically change the class and other scheduling parameters associated with a running LWP or set of LWPs given the appropriate permissions as explained below.</p> <p>In the default configuration, a runnable realtime LWP runs before any other LWP. Therefore, inappropriate use of realtime LWP can have a dramatic negative impact on system performance.</p> <p>The <code>priocntl()</code> function provides an interface for specifying a process, set of processes or an LWP to which the function is to apply. The <code>priocntlset(2)</code> function provides the same functions as <code>priocntl()</code>, but allows a more general interface for specifying the set of LWPs to which the function is to apply.</p> <p>For <code>priocntl()</code>, the <i>idtype</i> and <i>id</i> arguments are used together to specify the set of LWPs. The interpretation of <i>id</i> depends on the value of <i>idtype</i>. The possible values for <i>idtype</i> and corresponding interpretations of <i>id</i> are as follows:</p> <table><tr><td>P_LWPID</td><td>The <i>id</i> argument is an LWP ID. The <i>priocntl</i> function applies to the LWP with the specified ID within the calling process.</td></tr><tr><td>P_PID</td><td>The <i>id</i> argument is a process ID specifying a single process. The <i>priocntl()</i> function applies to all LWPs currently associated with the specified process.</td></tr><tr><td>P_PPID</td><td>The <i>id</i> argument is a parent process ID. The <i>priocntl()</i> function applies to all LWPs currently associated with processes with the specified parent process ID.</td></tr><tr><td>P_PGID</td><td>The <i>id</i> argument is a process group ID. The <i>priocntl()</i> function applies to all LWPs currently associated with processes in the specified process group.</td></tr><tr><td>P_SID</td><td>The <i>id</i> argument is a session ID. The <i>priocntl()</i> function applies to all LWPs currently associated with processes in the specified session.</td></tr></table>	P_LWPID	The <i>id</i> argument is an LWP ID. The <i>priocntl</i> function applies to the LWP with the specified ID within the calling process.	P_PID	The <i>id</i> argument is a process ID specifying a single process. The <i>priocntl()</i> function applies to all LWPs currently associated with the specified process.	P_PPID	The <i>id</i> argument is a parent process ID. The <i>priocntl()</i> function applies to all LWPs currently associated with processes with the specified parent process ID.	P_PGID	The <i>id</i> argument is a process group ID. The <i>priocntl()</i> function applies to all LWPs currently associated with processes in the specified process group.	P_SID	The <i>id</i> argument is a session ID. The <i>priocntl()</i> function applies to all LWPs currently associated with processes in the specified session.
P_LWPID	The <i>id</i> argument is an LWP ID. The <i>priocntl</i> function applies to the LWP with the specified ID within the calling process.										
P_PID	The <i>id</i> argument is a process ID specifying a single process. The <i>priocntl()</i> function applies to all LWPs currently associated with the specified process.										
P_PPID	The <i>id</i> argument is a parent process ID. The <i>priocntl()</i> function applies to all LWPs currently associated with processes with the specified parent process ID.										
P_PGID	The <i>id</i> argument is a process group ID. The <i>priocntl()</i> function applies to all LWPs currently associated with processes in the specified process group.										
P_SID	The <i>id</i> argument is a session ID. The <i>priocntl()</i> function applies to all LWPs currently associated with processes in the specified session.										

P_CID	The <i>id</i> argument is a class ID (returned by the prioset() PC_GETCID command as explained below). The prioset() function applies to all LWPs in the specified class.
P_UID	The <i>id</i> argument is a user ID. The prioset() function applies to all LWPs with this effective user ID.
P_GID	The <i>id</i> argument is a group ID. The prioset() function applies to all LWPs with this effective group ID.
P_ALL	The prioset() function applies to all existing LWPs. The value of <i>id</i> is ignored. The permission restrictions described below still apply.

An *id* value of P_MYID can be used in conjunction with the *idtype* value to specify the calling LWP's LWP ID, parent process ID, process group ID, session ID, class ID, user ID, or group ID.

In order to change the scheduling parameters of an LWP (using the PC_SETPARMS command as explained hereafter) the calling LWP must have process MAC write access, and the real or effective user ID of the LWP calling prioset must match the real or effective user ID of the receiving LWP or the calling LWP must have the PRIV_PROC_OWNER privilege. These are the minimum permission requirements enforced for all classes. An individual class may impose additional permissions requirements when setting LWPs to that class and/or when setting class-specific scheduling parameters.

A special sys scheduling class exists for the purpose of scheduling the execution of certain special system processes (such as the swapper process). It is not possible to change the class of any LWP to sys. In addition, any processes in the sys class that are included in a specified set of processes are disregarded by prioset(). For example, an *idtype* of P_UID and an *id* value of 0 would specify all processes with a user ID of 0 except processes in the sys class and (if changing the parameters using PC_SETPARMS) the init(1M) process.

The *init* process is a special case. In order for a prioset() call to change the class or other scheduling parameters of the *init* process (process ID 1), it must be the only process specified by *idtype* and *id*. The *init* process may be assigned to any class configured on the system, but the time-sharing class is almost always the appropriate choice. (Other choices may be highly undesirable; see the *System Administration Guide, Volume 1* for more information.)

The data type and value of *arg* are specific to the type of command specified by *cmd*.

A structure with the following members is used by the PC_GETCID and PC_GETCLINFO commands.

```
id_t    pc_cid;                /* Class id */
char    pc_clname[PC_CLNMSZ]; /* Class name */
int     pc_clinfo[PC_CLINFOSZ]; /* Class information */
```

priocntl(2)

The `pc_cid` member is a class ID returned by the `priocntl()` `PC_GETCID` command. The `pc_clname` member is a buffer of size `PC_CLNMSZ` (defined in `<sys/priocntl.h>`) used to hold the class name (RT for realtime or TS for time-sharing).

The `pc_clinfo` member is a buffer of size `PC_CLINFOSZ` (defined in `<sys/priocntl.h>`) used to return data describing the attributes of a specific class. The format of this data is class-specific and is described under the appropriate heading (REALTIME CLASS or TIME-SHARING CLASS) below.

A structure with the following elements is used by the `PC_SETPARMS` and `PC_GETPARMS` commands.

```
id_t   pc_cid;                /* LWP class */
int     pc_clparms[PC_CLPARMSZ]; /* Class-specific params */
```

The `pc_cid` member is a class ID (returned by the `priocntl()` `PC_GETCID` command). The special class ID `PC_CLNULL` can also be assigned to `pc_cid` when using the `PC_GETPARMS` command as explained below.

The `pc_clparms` buffer holds class-specific scheduling parameters. The format of this parameter data for a particular class is described under the appropriate heading below. `PC_CLPARMSZ` is the length of the `pc_clparms` buffer and is defined in `<sys/priocntl.h>`.

COMMANDS

Available `priocntl()` commands are:

PC_GETCID

Get class ID and class attributes for a specific class given class name. The *idtype* and *id* arguments are ignored. If *arg* is non-null, it points to a structure of type `pcinfo_t`. The `pc_clname` buffer contains the name of the class whose attributes you are getting.

On success, the class ID is returned in `pc_cid`, the class attributes are returned in the `pc_clinfo` buffer, and the `priocntl()` call returns the total number of classes configured in the system (including the `sys` class). If the class specified by `pc_clname` is invalid or is not currently configured the `priocntl()` call returns `-1` with `errno` set to `EINVAL`. The format of the attribute data returned for a given class is defined in the `<sys/rtpriocntl.h>` or `<sys/tpriocntl.h>` header and described under the appropriate heading below.

If *arg* is a null pointer, no attribute data is returned but the `priocntl()` call still returns the number of configured classes.

PC_GETCLINFO

Get class name and class attributes for a specific class given class ID. The *idtype* and *id* arguments are ignored. If *arg* is non-null, it points to a structure of type `pcinfo_t`. The `pc_cid` member is the class ID of the class whose attributes you are getting.

On success, the class name is returned in the `pc_clname` buffer, the class attributes are returned in the `pc_clinfo` buffer, and the `prioset()` call returns the total number of classes configured in the system (including the `sys` class). The format of the attribute data returned for a given class is defined in the `<sys/rtprioset.h>` or `<sys/tsprioset.h>` header file and described under the appropriate heading below.

If *arg* is a null pointer, no attribute data is returned but the `prioset()` call still returns the number of configured classes.

PC_SETPARMS

Set the class and class-specific scheduling parameters of the specified LWP(s) associated with the specified process(es). When this command is used with the *idtype* of `P_LWPID`, it will set the class and class-specific scheduling parameters of the LWP. The *arg* argument points to a structure of type `pcparms_t`. The `pc_cid` member specifies the class you are setting and the `pc_clparms` buffer contains the class-specific parameters you are setting. The format of the class-specific parameter data is defined in the `<sys/rtprioset.h>` or `<sys/tsprioset.h>` header and described under the appropriate class heading below.

When setting parameters for a set of LWPs, `prioset()` acts on the LWPs in the set in an implementation-specific order. If `prioset()` encounters an error for one or more of the target processes, it may or may not continue through the set of LWPs, depending on the nature of the error. If the error is related to permissions (`EPERM`), `prioset()` continues through the LWP set, resetting the parameters for all target LWPs for which the calling LWP has appropriate permissions. The `prioset()` function then returns `-1` with `errno` set to `EPERM` to indicate that the operation failed for one or more of the target LWPs. If `prioset()` encounters an error other than permissions, it does not continue through the set of target LWPs but returns the error immediately.

PC_GETPARMS

Get the class and/or class-specific scheduling parameters of an LWP. The *arg* member points to a structure of type `pcparms_t`.

If `pc_cid` specifies a configured class and a single LWP belonging to that class is specified by the *idtype* and *id* values or the `procset` structure, then the scheduling parameters of that LWP are returned in the `pc_clparms` buffer. If the LWP specified does not exist or does not belong to the specified class, the `prioset()` call returns `-1` with `errno` set to `ESRCH`.

If `pc_cid` specifies a configured class and a set of LWPs is specified, the scheduling parameters of one of the specified LWP belonging to the specified class are returned in the `pc_clparms` buffer and the `prioset()` call returns the process ID of the selected LWP. The criteria for selecting an LWP to return in this case is class dependent. If none of the specified LWPs exist or none of them belong to the specified class the `prioset()` call returns `-1` with `errno` set to `ESRCH`.

priocntl(2)

REALTIME CLASS

If `pc_cid` is `PC_CLNULL` and a single LWP is specified the class of the specified LWP is returned in `pc_cid` and its scheduling parameters are returned in the `pc_clparms` buffer.

PC_ADMIN

This command provides functionality needed for the implementation of the `dispadmin(1)` utility. It is not intended for general use by other applications.

The realtime class provides a fixed priority preemptive scheduling policy for those LWPs requiring fast and deterministic response and absolute user/application control of scheduling priorities. If the realtime class is configured in the system it should have exclusive control of the highest range of scheduling priorities on the system. This ensures that a runnable realtime LWP is given CPU service before any LWP belonging to any other class.

The realtime class has a range of realtime priority (`rt_pri`) values that may be assigned to an LWP within the class. Real-time priorities range from 0 to x , where the value of x is configurable and can be determined for a specific installation by using the `priocntl()` `PC_GETCID` or `PC_GETCLINFO` command.

The realtime scheduling policy is a fixed priority policy. The scheduling priority of a realtime LWP is never changed except as the result of an explicit request by the user/application to change the `rt_pri` value of the LWP.

For an LWP in the realtime class, the `rt_pri` value is, for all practical purposes, equivalent to the scheduling priority of the LWP. The `rt_pri` value completely determines the scheduling priority of a realtime LWP relative to other LWPs within its class. Numerically higher `rt_pri` values represent higher priorities. Since the realtime class controls the highest range of scheduling priorities in the system it is guaranteed that the runnable realtime LWP with the highest `rt_pri` value is always selected to run before any other LWPs in the system.

In addition to providing control over priority, `priocntl()` provides for control over the length of the time quantum allotted to the LWP in the realtime class. The time quantum value specifies the maximum amount of time an LWP may run assuming that it does not complete or enter a resource or event wait state (*sleep*). Note that if another LWP becomes runnable at a higher priority, the currently running LWP may be preempted before receiving its full time quantum.

The system's process scheduler keeps the runnable realtime LWPs on a set of scheduling queues. There is a separate queue for each configured realtime priority and all realtime LWPs with a given `rt_pri` value are kept together on the appropriate queue. The LWPs on a given queue are ordered in FIFO order (that is, the LWP at the front of the queue has been waiting longest for service and receives the CPU first). Real-time LWPs that wake up after sleeping, LWPs which change to the realtime class from some other class, LWPs which have used their full time quantum, and runnable LWPs whose priority is reset by `priocntl()` are all placed at the back of the appropriate queue for their priority. An LWP that is preempted by a higher priority LWP remains at the front of the queue (with whatever time is remaining in its time

quantum) and runs before any other LWP at this priority. Following a `fork(2)` or `_lwp_create(2)` function call by a realtime LWP, the parent LWP continues to run while the child LWP (which inherits its parent's `rt_pri` value) is placed at the back of the queue.

A structure with the following members (defined in `<sys/rtprioctl.h>`) defines the format used for the attribute data for the realtime class.

```
short    rt_maxpri;        /* Maximum realtime priority */
```

The `prioctl()` `PC_GETCID` and `PC_GETCLINFO` commands return realtime class attributes in the `pc_clinfo` buffer in this format.

The `rt_maxpri` member specifies the configured maximum `rt_pri` value for the realtime class (if `rt_maxpri` is *x*, the valid realtime priorities range from 0 to *x*).

A structure with the following members (defined in `<sys/rtprioctl.h>`) defines the format used to specify the realtime class-specific scheduling parameters of an LWP.

```
short    rt_pri;           /* Real-Time priority */
uint_t   rt_tqsecs;        /* Seconds in time quantum */
int       rt_tqnsecs;       /* Additional nanoseconds in quantum */
```

When using the `prioctl()` `PC_SETPARMS` or `PC_GETPARMS` commands, if `pc_cid` specifies the realtime class, the data in the `pc_clparms` buffer is in this format.

The above commands can be used to set the realtime priority to the specified value or get the current `rt_pri` value. Setting the `rt_pri` value of an LWP that is currently running or runnable (not sleeping) causes the LWP to be placed at the back of the scheduling queue for the specified priority. The LWP is placed at the back of the appropriate queue regardless of whether the priority being set is different from the previous `rt_pri` value of the LWP. Note that a running LWP can voluntarily release the CPU and go to the back of the scheduling queue at the same priority by resetting its `rt_pri` value to its current realtime priority value. In order to change the time quantum of an LWP without setting the priority or affecting the LWP's position on the queue, the `rt_pri` member should be set to the special value `RT_NOCHANGE` (defined in `<sys/rtprioctl.h>`). Specifying `RT_NOCHANGE` when changing the class of an LWP to realtime from some other class results in the realtime priority being set to 0.

For the `prioctl()` `PC_GETPARMS` command, if `pc_cid` specifies the realtime class and more than one realtime LWP is specified, the scheduling parameters of the realtime LWP with the highest `rt_pri` value among the specified LWPs are returned and the LWP ID of this LWP is returned by the `prioctl()` call. If there is more than one LWP sharing the highest priority, the one returned is implementation-dependent.

The `rt_tqsecs` and `rt_tqnsecs` members are used for getting or setting the time quantum associated with an LWP or group of LWPs. `rt_tqsecs` is the number of seconds in the time quantum and `rt_tqnsecs` is the number of additional nanoseconds in the quantum. For example setting `rt_tqsecs` to 2 and `rt_tqnsecs` to 500,000,000 (decimal) would result in a time quantum of two and one-half seconds.

priocntl(2)

Specifying a value of 1,000,000,000 or greater in the `rt_tqnsecs` member results in an error return with `errno` set to `EINVAL`. Although the resolution of the `tq_nsecs` member is very fine, the specified time quantum length is rounded up by the system to the next integral multiple of the system clock's resolution. The maximum time quantum that can be specified is implementation-specific and equal to `LONG_MAX` ticks (defined in `<limits.h>`). Requesting a quantum greater than this maximum results in an error return with `errno` set to `ERANGE` (although infinite quanta may be requested using a special value as explained below). Requesting a time quantum of 0 (setting both `rt_tqsecs` and `rt_tqnsecs` to 0) results in an error return with `errno` set to `EINVAL`.

The `rt_tqnsecs` member can also be set to one of the following special values (defined in `<sys/rtpriocntl.h>`), in which case the value of `rt_tqsecs` is ignored:

<code>RT_TQINF</code>	Set an infinite time quantum.
<code>RT_TQDEF</code>	Set the time quantum to the default for this priority (see <code>rt_dptbl(4)</code>).
<code>RT_NOCHANGE</code>	Do not set the time quantum. This value is useful when you wish to change the realtime priority of an LWP without affecting the time quantum. Specifying this value when changing the class of an LWP to realtime from some other class is equivalent to specifying <code>RT_TQDEF</code> .

In order to change the class of an LWP to real-time (from any other class), the LWP invoking `priocntl()` must have the `PRIV_SYS_CONFIG` privilege. In order to change the priority or time quantum setting of a real-time LWP, the LWP invoking `priocntl()` must have the `PRIV_PROC_OWNER` privilege or must itself be a real-time LWP whose real or effective user ID matches the real or effective user ID of the target LWP.

The real-time priority and time quantum are inherited across the `fork(2)` and `exec(2)` system calls.

TIME-SHARING CLASS

The time-sharing scheduling policy provides for a fair and effective allocation of the CPU resource among LWPs with varying CPU consumption characteristics. The objectives of the time-sharing policy are to provide good response time to interactive LWPs and good throughput to CPU-bound jobs while providing a degree of user/application control over scheduling.

The time-sharing class has a range of time-sharing user priority (see `ts_upri` below) values that may be assigned to LWPs within the class. A `ts_upri` value of 0 is defined as the default base priority for the time-sharing class. User priorities range from $-x$ to $+x$ where the value of x is configurable and can be determined for a specific installation by using the `priocntl()` `PC_GETCID` or `PC_GETCLINFO` command.

The purpose of the user priority is to provide some degree of user/application control over the scheduling of LWPs in the time-sharing class. Raising or lowering the

`ts_upri` value of an LWP in the time-sharing class raises or lowers the scheduling priority of the LWP. It is not guaranteed, however, that an LWP with a higher `ts_upri` value will run before one with a lower `ts_upri` value. This is because the `ts_upri` value is just one factor used to determine the scheduling priority of a time-sharing LWP. The system may dynamically adjust the internal scheduling priority of a time-sharing LWP based on other factors such as recent CPU usage.

In addition to the system-wide limits on user priority (returned by the `PC_GETCID` and `PC_GETCLINFO` commands) there is a per LWP user priority limit (see `ts_uprilm` below), which specifies the maximum `ts_upri` value that may be set for a given LWP; by default, `ts_uprilm` is 0.

A structure with the following members (defined in `<sys/tspriocntl.h>`) defines the format used for the attribute data for the time-sharing class.

```
short    ts_maxupri;    /* Limits of user priority range */
```

The `priocntl()` `PC_GETCID` and `PC_GETCLINFO` commands return time-sharing class attributes in the `pc_clinfo` buffer in this format.

`ts_maxupri` specifies the configured maximum user priority value for the time-sharing class. If `ts_maxupri` is x , the valid range for both user priorities and user priority limits is from $-x$ to $+x$.

A structure with the following members (defined in `<sys/tspriocntl.h>`) defines the format used to specify the time-sharing class-specific scheduling parameters of an LWP.

```
short    ts_uprilm;     /* Time-Sharing user priority limit */
short    ts_upri;       /* Time-Sharing user priority */
```

When using the `priocntl()` `PC_SETPARMS` or `PC_GETPARMS` commands, if `pc_cid` specifies the time-sharing class, the data in the `pc_clparms` buffer is in this format.

For the `priocntl()` `PC_GETPARMS` command, if `pc_cid` specifies the time-sharing class and more than one time-sharing LWP is specified, the scheduling parameters of the time-sharing LWP with the highest `ts_upri` value among the specified LWPs is returned and the LWP ID of this LWP is returned by the `priocntl()` call. If there is more than one LWP sharing the highest user priority, the one returned is implementation-dependent.

Any time-sharing LWP may lower its own `ts_uprilm` (or that of another LWP with the same user ID). Only a time-sharing LWP with the `PRIV_SYS_CONFIG` privilege may raise a `ts_uprilm`. When changing the class of an LWP to time-sharing from some other class, the `PRIV_SYS_CONFIG` privilege is required in order to set the initial `ts_uprilm` to a value greater than zero. Attempts by a nonprivileged LWP to raise a `ts_uprilm` or set an initial `ts_uprilm` greater than zero fail with a return value of `-1` and `errno` set to `EPERM`.

priocntl(2)

Any time-sharing LWP may set its own `ts_upri` (or that of another LWP with the same user ID) to any value less than or equal to the LWP's `ts_uprilm`. Attempts to set the `ts_upri` above the `ts_uprilm` (and/or set the `ts_uprilm` below the `ts_upri`) result in the `ts_upri` being set equal to the `ts_uprilm`.

Either of the `ts_uprilm` or `ts_upri` members may be set to the special value `TS_NOCHANGE` (defined in `<sys/tspriocntl.h>`) in order to set one of the values without affecting the other. Specifying `TS_NOCHANGE` for the `ts_upri` when the `ts_uprilm` is being set to a value below the current `ts_upri` causes the `ts_upri` to be set equal to the `ts_uprilm` being set. Specifying `TS_NOCHANGE` for a parameter when changing the class of an LWP to time-sharing (from some other class) causes the parameter to be set to a default value. The default value for the `ts_uprilm` is 0 and the default for the `ts_upri` is to set it equal to the `ts_uprilm` which is being set.

The time-sharing user priority and user priority limit are inherited across `fork()` and the `exec` family of functions.

RETURN VALUES

Unless otherwise noted above, `priocntl()` returns a value of 0 on success. On failure, `priocntl()` returns -1 and sets `errno` to indicate the error.

ERRORS

The `priocntl()` function fails if:

EAGAIN	An attempt to change the class of an LWP failed because of insufficient resources other than memory (for example, class-specific kernel data structures).
EFAULT	One of the arguments points to an illegal address.
EINVAL	The argument <i>cmd</i> was invalid, an invalid or unconfigured class was specified, or one of the parameters specified was invalid.
ENOMEM	An attempt to change the class of an LWP failed because of insufficient memory.
EPERM	The calling LWP does not have required privileges.
ERANGE	The requested time quantum is out of range.
ESRCH	None of the specified LWPs exist.

SUMMARY OF TRUSTED SOLARIS CHANGES Trusted Solaris 8 4/01 Reference Manual SunOS 5.10 Reference Manual

The Trusted Solaris environment replaces the checks of super-user in the Solaris environment with privilege checks. MAC policy is enforced in addition to DAC.

`dispadm(1M)`, `init(1M)`, `exec(2)`, `fork(2)`, `nice(2)`, `priocntlset(2)`

`priocntl(1)`, `_lwp_create(2)`, `rt_dptbl(4)`

System Administration Guide, Volume 1 System Interface Guide

NAME	priocntlset – Generalized process scheduler control								
SYNOPSIS	<pre>#include <sys/types.h> #include <sys/procset.h> #include <sys/priocntl.h> #include <sys/rtpriocntl.h> #include <sys/tspriocntl.h> long priocntlset(procset_t *psp, int cmd, /* arg */ ...);</pre>								
DESCRIPTION	<p>The <code>priocntlset()</code> function changes the scheduling properties of running processes. <code>priocntlset()</code> has the same functions as the <code>priocntl()</code> function, but a more general way of specifying the set of processes whose scheduling properties are to be changed.</p> <p><i>cmd</i> specifies the function to be performed. <i>arg</i> is a pointer to a structure whose type depends on <i>cmd</i>. See <code>priocntl(2)</code> for the valid values of <i>cmd</i> and the corresponding <i>arg</i> structures.</p> <p><i>psp</i> is a pointer to a <code>procset</code> structure, which <code>priocntlset()</code> uses to specify the set of processes whose scheduling properties are to be changed. The <code>procset</code> structure contains the following members:</p> <pre>idop_t p_op; /* operator connecting left/right sets */ idtype_t p_lidtype; /* left set ID type */ id_t p_lid; /* left set ID */ idtype_t p_ridtype; /* right set ID type */ id_t p_rid; /* right set ID */</pre> <p>The <code>p_lidtype</code> and <code>p_lid</code> members specify the ID type and ID of one (“left”) set of processes; the <code>p_ridtype</code> and <code>p_rid</code> members specify the ID type and ID of a second (“right”) set of processes. ID types and IDs are specified just as for the <code>priocntl()</code> function. The <code>p_op</code> member specifies the operation to be performed on the two sets of processes to get the set of processes the function is to apply to. The valid values for <code>p_op</code> and the processes they specify are:</p> <table> <tr> <td>POP_DIFF</td><td>Set difference: processes in left set and not in right set.</td></tr> <tr> <td>POP_AND</td><td>Set intersection: processes in both left and right sets.</td></tr> <tr> <td>POP_OR</td><td>Set union: processes in either left or right sets or both.</td></tr> <tr> <td>POP_XOR</td><td>Set exclusive-or: processes in left or right set but not in both.</td></tr> </table> <p>The following macro, which is defined in <code><procset.h></code>, offers a convenient way to initialize a <code>procset</code> structure:</p> <pre>#define setprocset(psp, op, ltype, lid, rtype, rid) \ (psp)⇒p_op = (op), \ (psp)⇒p_lidtype = (ltype), \ (psp)⇒p_lid = (lid), \ (psp)⇒p_ridtype = (rtype), \ (psp)⇒p_rid = (rid),</pre>	POP_DIFF	Set difference: processes in left set and not in right set.	POP_AND	Set intersection: processes in both left and right sets.	POP_OR	Set union: processes in either left or right sets or both.	POP_XOR	Set exclusive-or: processes in left or right set but not in both.
POP_DIFF	Set difference: processes in left set and not in right set.								
POP_AND	Set intersection: processes in both left and right sets.								
POP_OR	Set union: processes in either left or right sets or both.								
POP_XOR	Set exclusive-or: processes in left or right set but not in both.								

prionctlset(2)

RETURN VALUES

Unless otherwise noted above, `prionctlset()` returns 0 on success. Otherwise, it returns -1 and sets `errno` to indicate the error.

ERRORS

The `prionctlset()` function will fail if:

EAGAIN	An attempt to change the class of a process failed because of insufficient resources other than memory (for example, class-specific kernel data structures).
EFAULT	One of the arguments points to an illegal address.
EINVAL	The argument <i>cmd</i> was invalid, an invalid or unconfigured class was specified, or one of the parameters specified was invalid.
ENOMEM	An attempt to change the class of a process failed because of insufficient memory.
EPERM	The calling process does not have required privileges.
ERANGE	The requested time quantum is out of range.
ESRCH	None of the specified processes exist.

SUMMARY OF TRUSTED SOLARIS CHANGES

Trusted Solaris 8
4/01 Reference
Manual
SunOS 5.8
Reference Manual

The Trusted Solaris environment replaces the checks of super-user in the Solaris environment with privilege checks.

`prionctl(2)`

`prionctl(1)`

NAME	processor_bind – bind LWP's to a processor								
SYNOPSIS	<pre>#include <sys/types.h> #include <sys/processor.h> #include <sys/procset.h> int processor_bind(idtype_t <i>idtype</i>, id_t <i>id</i>, processorid_t <i>processorid</i>, processorid_t *<i>obind</i>);</pre>								
DESCRIPTION	<p>The <code>processor_bind()</code> function binds the LWP (lightweight process) or set of LWP's specified by <i>idtype</i> and <i>id</i> to the processor specified by <i>processorid</i>. If <i>obind</i> is not NULL, this function also sets the <code>processorid_t</code> variable pointed to by <i>obind</i> to the previous binding of one of the specified LWP's, or to <code>PBIND_NONE</code> if the selected LWP was not bound.</p> <p>If <i>idtype</i> is <code>P_PID</code>, the binding affects all LWP's of the process with process ID (PID) <i>id</i>.</p> <p>If <i>idtype</i> is <code>P_LWPID</code>, the binding affects the LWP of the current process with LWP ID <i>id</i>.</p> <p>If <i>idtype</i> is <code>P_TASKID</code>, the binding affects all LWP's of all processes with task ID <i>id</i>.</p> <p>If <i>id</i> is <code>P_MYID</code>, the specified LWP, process, or task is the current one.</p> <p>If <i>processorid</i> is <code>PBIND_NONE</code>, the processor bindings of the specified LWP's are cleared.</p> <p>If <i>processorid</i> is <code>PBIND_QUERY</code>, the processor bindings are not changed.</p> <p>The effective user of the calling process must be superuser, or its real or effective user ID must match the real or effective user ID of the LWP's being bound. If the calling process does not have permission to change all of the specified LWP's, the bindings of the LWP's for which it does have permission will be changed even though an error is returned.</p>								
RETURN VALUES	Upon successful completion, 0 is returned. Otherwise, -1 is returned and <code>errno</code> is set to indicate the error.								
ERRORS	<p>The <code>processor_bind()</code> function will fail if:</p> <table> <tr> <td><code>EFAULT</code></td><td>The location pointed to by <i>obind</i> was not NULL and not writable by the user.</td></tr> <tr> <td><code>EINVAL</code></td><td>The specified processor is not on-line, or the <i>idtype</i> argument was not <code>P_PID</code>, <code>P_LWPID</code>, or <code>P_TASKID</code>.</td></tr> <tr> <td><code>EPERM</code></td><td>The effective user of the calling process is not superuser, and its real or effective user ID does not match the real or effective user ID of one of the LWP's being bound.</td></tr> <tr> <td><code>ESRCH</code></td><td>No processes, LWP's, or tasks were found to match the criteria specified by <i>idtype</i> and <i>id</i>.</td></tr> </table>	<code>EFAULT</code>	The location pointed to by <i>obind</i> was not NULL and not writable by the user.	<code>EINVAL</code>	The specified processor is not on-line, or the <i>idtype</i> argument was not <code>P_PID</code> , <code>P_LWPID</code> , or <code>P_TASKID</code> .	<code>EPERM</code>	The effective user of the calling process is not superuser, and its real or effective user ID does not match the real or effective user ID of one of the LWP's being bound.	<code>ESRCH</code>	No processes, LWP's, or tasks were found to match the criteria specified by <i>idtype</i> and <i>id</i> .
<code>EFAULT</code>	The location pointed to by <i>obind</i> was not NULL and not writable by the user.								
<code>EINVAL</code>	The specified processor is not on-line, or the <i>idtype</i> argument was not <code>P_PID</code> , <code>P_LWPID</code> , or <code>P_TASKID</code> .								
<code>EPERM</code>	The effective user of the calling process is not superuser, and its real or effective user ID does not match the real or effective user ID of one of the LWP's being bound.								
<code>ESRCH</code>	No processes, LWP's, or tasks were found to match the criteria specified by <i>idtype</i> and <i>id</i> .								

processor_bind(2)

SEE ALSO psradm(1M), psrinfo(1M), p_online(2), pset_bind(2), sysconf(3C)

NAME	write, pwrite, writev, writel, pwriterl, writevl – write on a file
SYNOPSIS	<pre> #include <unistd.h> ssize_t write(int fildes, const void *buf, size_t nbyte); ssize_t pwrite(int fildes, const void *buf, size_t nbyte, off_t offset); #include <sys/uio.h> ssize_t writev(int fildes, const struct iovec *iov, int iovcnt); #include <tsol/rdwrl.h> ssize_t writel(int fildes, void *buf, size_t nbyte, bclabel_t *label_p); ssize_t pwriterl(int fildes, void *buf, size_t nbyte, off_t offset, bclabel_t *label_p); ssize_t writevl(int fildes, struct iovec *iov, int iovcnt, bclabel_t *label_p); </pre>
DESCRIPTION	<p>The <code>write()</code> function attempts to write <i>nbyte</i> bytes from the buffer pointed to by <i>buf</i> to the file associated with the open file descriptor, <i>fildes</i>.</p> <p>If <i>nbyte</i> is 0, <code>write()</code> will return 0 and have no other results if the file is a regular file; otherwise, the results are unspecified.</p> <p>On a regular file or other file capable of seeking, the actual writing of data proceeds from the position in the file indicated by the file offset associated with <i>fildes</i>. Before successful return from <code>write()</code>, the file offset is incremented by the number of bytes actually written. On a regular file, if this incremented file offset is greater than the length of the file, the length of the file will be set to this file offset.</p> <p>If the <code>O_SYNC</code> flag of the file status flags is set and <i>fildes</i> refers to a regular file, a successful <code>write()</code> does not return until the data is delivered to the underlying hardware.</p> <p>If <i>fildes</i> refers to a socket, <code>write()</code> is equivalent to <code>send(3SOCKET)</code> with no flags set.</p> <p>On a file not capable of seeking, writing always takes place starting at the current position. The value of a file offset associated with such a device is undefined.</p> <p>If the <code>O_APPEND</code> flag of the file status flags is set, the file offset will be set to the end of the file prior to each write and no intervening file modification operation will occur between changing the file offset and the write operation.</p> <p>For regular files, no data transfer will occur past the offset maximum established in the open file description with <i>fildes</i>.</p> <p>A <code>write()</code> to a regular file is blocked if mandatory file/record locking is set (see <code>chmod(2)</code>), and there is a record lock owned by another process on the segment of the file to be written:</p>

pwrite(2)

- If `O_NDELAY` or `O_NONBLOCK` is set, `write()` returns `-1` and sets `errno` to `EAGAIN`.
- If `O_NDELAY` and `O_NONBLOCK` are clear, `write()` sleeps until all blocking locks are removed or the `write()` is terminated by a signal.

If a `write()` requests that more bytes be written than there is room for—for example, if the write would exceed the process file size limit (see `getrlimit(2)` and `ulimit(2)`), the system file size limit, or the free space on the device—only as many bytes as there is room for will be written. For example, suppose there is space for 20 bytes more in a file before reaching a limit. A `write()` of 512-bytes returns 20. The next `write()` of a non-zero number of bytes gives a failure return (except as noted for pipes and FIFO below).

If `write()` is interrupted by a signal before it writes any data, it will return `-1` with `errno` set to `EINTR`.

If `write()` is interrupted by a signal after it successfully writes some data, it will return the number of bytes written.

If the value of *nbyte* is greater than `SSIZE_MAX`, the result is implementation-dependent.

After a `write()` to a regular file has successfully returned:

- Any successful `read(2)` from each byte position in the file that was modified by that write will return the data specified by the `write()` for that position until such byte positions are again modified.
- Any subsequent successful `write()` to the same byte position in the file will overwrite that file data.

Write requests to a pipe or FIFO are handled the same as a regular file with the following exceptions:

- There is no file offset associated with a pipe, hence each write request appends to the end of the pipe.
- Write requests of `{PIPE_BUF}` bytes or less are guaranteed not to be interleaved with data from other processes doing writes on the same pipe. Writes of greater than `{PIPE_BUF}` bytes may have data interleaved, on arbitrary boundaries, with writes by other processes, whether or not the `O_NONBLOCK` or `O_NDELAY` flags are set.
- If `O_NONBLOCK` and `O_NDELAY` are clear, a write request may cause the process to block, but on normal completion it returns *nbyte*.
- If `O_NONBLOCK` and `O_NDELAY` are set, `write()` does not block the process. If a `write()` request for `PIPE_BUF` or fewer bytes succeeds completely `write()` returns *nbyte*. Otherwise, if `O_NONBLOCK` is set, it returns `-1` and sets `errno` to `EAGAIN` or if `O_NDELAY` is set, it returns 0. A `write()` request for greater than `{PIPE_BUF}` bytes transfers what it can and returns the number of bytes written.

or it transfers no data and, if `O_NONBLOCK` is set, returns `-1` with `errno` set to `EAGAIN` or if `O_NDELAY` is set, it returns `0`. Finally, if a request is greater than `PIPE_BUF` bytes and all data previously written to the pipe has been read, `write()` transfers at least `PIPE_BUF` bytes.

When attempting to write to a file descriptor (other than a pipe, a FIFO, a socket, or a STREAM) that supports nonblocking writes and cannot accept the data immediately:

- If `O_NONBLOCK` and `O_NDELAY` are clear, `write()` blocks until the data can be accepted.
- If `O_NONBLOCK` or `O_NDELAY` is set, `write()` does not block the process. If some data can be written without blocking the process, `write()` writes what it can and returns the number of bytes written. Otherwise, if `O_NONBLOCK` is set, it returns `-1` and sets `errno` to `EAGAIN` or if `O_NDELAY` is set, it returns `0`.

Upon successful completion, where *nbyte* is greater than 0, `write()` will mark for update the `st_ctime` and `st_mtime` fields of the file, and if the file is a regular file, the `S_ISUID` and `S_ISGID` bits of the file mode may be cleared.

For STREAMS files (see `intro(3)` and `streamio(7I)`), the operation of `write()` is determined by the values of the minimum and maximum *nbyte* range ("packet size") accepted by the STREAM. These values are contained in the topmost STREAM module, and cannot be set or tested from user level. If *nbyte* falls within the packet size range, *nbyte* bytes are written. If *nbyte* does not fall within the range and the minimum packet size value is zero, `write()` breaks the buffer into maximum packet size segments prior to sending the data downstream (the last segment may be smaller than the maximum packet size). If *nbyte* does not fall within the range and the minimum value is non-zero, `write()` fails and sets `errno` to `ERANGE`. Writing a zero-length buffer (*nbyte* is zero) to a STREAMS device sends a zero length message with zero returned. However, writing a zero-length buffer to a pipe or FIFO sends no message and zero is returned. The user program may issue the `I_SWROPT` `ioctl(2)` to enable zero-length messages to be sent across the pipe or FIFO (see `streamio(7I)`).

When writing to a STREAM, data messages are created with a priority band of zero. When writing to a socket or to a STREAM that is not a pipe or a FIFO:

- If `O_NDELAY` and `O_NONBLOCK` are not set, and the STREAM cannot accept data (the STREAM write queue is full due to internal flow control conditions), `write()` blocks until data can be accepted.
- If `O_NDELAY` or `O_NONBLOCK` is set and the STREAM cannot accept data, `write()` returns `-1` and sets `errno` to `EAGAIN`.
- If `O_NDELAY` or `O_NONBLOCK` is set and part of the buffer has already been written when a condition occurs in which the STREAM cannot accept additional data, `write()` terminates and returns the number of bytes written.

The `write()` and `writew()` functions will fail if the STREAM head had processed an asynchronous error before the call. In this case, the value of `errno` does not reflect the result of `write()` or `writew()` but reflects the prior error.

pwrite(2)

pwrite()	<p>The <code>pwrite()</code> function performs the same action as <code>write()</code>, except that it writes into a given position without changing the file pointer. The first three arguments to <code>pwrite()</code> are the same as <code>write()</code> with the addition of a fourth argument <i>offset</i> for the desired position inside the file.</p>
writev()	<p>The <code>writev()</code> function performs the same action as <code>write()</code>, but gathers the output data from the <i>iovcnt</i> buffers specified by the members of the <i>iov</i> array: <i>iov</i>[0], <i>iov</i>[1], ..., <i>iov</i>[<i>iovcnt</i> - 1]. The <i>iovcnt</i> buffer is valid if greater than 0 and less than or equal to <code>IOV_MAX</code>. See <code>intro(2)</code> for a definition of <code>IOV_MAX</code>.</p> <p>The <code>iovec</code> structure contains the following members:</p> <pre>caddr_t iov_base; int iov_len;</pre> <p>Each <code>iovec</code> entry specifies the base address and length of an area in memory from which data should be written. The <code>writev()</code> function always writes all data from an area before proceeding to the next.</p> <p>If <i>fdes</i> refers to a regular file and all of the <i>iov_len</i> members in the array pointed to by <i>iov</i> are 0, <code>writev()</code> will return 0 and have no other effect. For other file types, the behavior is unspecified.</p> <p>If the sum of the <i>iov_len</i> values is greater than <code>SSIZE_MAX</code>, the operation fails and no data is transferred.</p>
writel(), pwwritel(), and writevl()	<p><code>writel()</code>, <code>pwwritel()</code>, and <code>writevl()</code> perform the same actions as <code>write()</code>, <code>pwrite()</code>, and <code>writev()</code>, respectively, and additionally provide the CMW label <i>label_p</i> to associate with the data that is written. The label associated with the data that is written to <i>fd</i> has this restriction:</p> <ul style="list-style-type: none">■ If the descriptor refers to a file or a FIFO, then the sensitivity label portion of <i>label_p</i> is ignored. <p>In all other respects, the <code>writel()</code>, <code>pwwritel()</code>, and <code>writevl()</code> interfaces are analogous to the <code>write()</code>, <code>pwrite()</code>, and <code>writev()</code> interfaces.</p> <p>If the set-user-ID or get-group-ID bits of <i>fdes</i> are set, they are cleared by the write. The calling process may assert the <code>PRIV_FILE_SETID</code> privilege to suppress this action.</p> <p>If the forced or allowed privilege sets of <i>fdes</i> are not empty, they are cleared by the write. The calling process may assert the <code>PRIV_FILE_SETPRIV</code> privilege to suppress this action.</p> <p>If the public object attributes flag, <code>FAF_PUBLIC</code>, of <i>fdes</i> is set, the flag is cleared by the write. The calling process may assert the <code>PRIV_FILE_AUDIT</code> privilege to suppress this action.</p>

	If the write causes the file-system free space to fall below its minimum level, the write fails. The calling process may assert the <code>PRIV_SYS_MINFREE</code> privilege to bypass this restriction.	
RETURN VALUES	Upon successful completion, <code>write()</code> returns the number of bytes actually written to the file associated with <i>fil</i> . This number is never greater than <i>nbyte</i> . Otherwise, <code>-1</code> is returned, the file-pointer remains unchanged, and <code>errno</code> is set to indicate the error.	
	Upon successful completion, <code>writew()</code> returns the number of bytes actually written. Otherwise, it returns <code>-1</code> , the file-pointer remains unchanged, and <code>errno</code> is set to indicate an error.	
ERRORS	The <code>write()</code> , <code>pwrite()</code> , <code>writew()</code> , <code>writel()</code> , <code>pritel()</code> , and <code>writewl()</code> functions will fail if:	
	EAGAIN	Mandatory file/record locking is set, <code>O_NDELAY</code> or <code>O_NONBLOCK</code> is set, and there is a blocking record lock; total amount of system memory available when reading using raw I/O is temporarily insufficient; an attempt is made to write to a <code>STREAM</code> that can not accept data with the <code>O_NDELAY</code> or <code>O_NONBLOCK</code> flag set; or a write to a pipe or FIFO of <code>PIPE_BUF</code> bytes or less is requested and less than <i>nbytes</i> of free space is available.
	EBADF	The <i>fil</i> argument is not a valid file descriptor open for writing.
	EDEADLK	The write was going to go to sleep and cause a deadlock situation to occur.
	EDQUOT	The user's quota of disk blocks on the file system containing the file has been exhausted.
	EFAULT	The <i>buf</i> argument points to an illegal address.
	EFBIG	An attempt is made to write a file that exceeds the process's file size limit or the maximum file size (see <code>getrlimit(2)</code> and <code>ulimit(2)</code>).
		The file is a regular file, <i>nbyte</i> is greater than 0, and the starting position is greater than or equal to the offset maximum established in the file description associated with <i>fil</i> .
	EINTR	A signal was caught during the write operation and no data was transferred.
	EIO	The process is in the background and is attempting to write to its controlling terminal whose <code>TOSTOP</code> flag is set, or the process is neither ignoring nor blocking <code>SIGTTOU</code> signals and the process group of the process is orphaned.

pwrite(2)

ENOLCK	Enforced record locking was enabled and LOCK_MAX regions are already locked in the system, or the system record lock table was full and the write could not go to sleep until the blocking record lock was removed.
ENOLINK	The <i>fildev</i> argument is on a remote machine and the link to that machine is no longer active.
ENOSPC	During a write to an ordinary file, there is no free space left on the device.
ENOSR	An attempt is made to write to a STREAMS with insufficient STREAMS memory resources available in the system.
ENXIO	A hangup occurred on the STREAM being written to.
EPIPE	An attempt is made to write to a pipe or a FIFO that is not open for reading by any process, or that has only one end open (or to a file descriptor created by <code>socket(3SOCKET)</code> , using type <code>SOCK_STREAM</code> that is no longer connected to a peer endpoint). A <code>SIGPIPE</code> signal will also be sent to the process. The process dies unless special provisions were taken to catch or ignore the signal.
ERANGE	The transfer request size was outside the range supported by the STREAMS file associated with <i>fildev</i> .

The `pwrite()` function fails and the file pointer remains unchanged if:

ESPIPE	The <i>fildev</i> argument is associated with a pipe or FIFO.
--------	---

The `writev()` function will fail if:

EINVAL	The sum of the <code>iov_len</code> values in the <i>iov</i> array would overflow an <code>ssize_t</code> .
--------	---

The `write()` and `writev()` functions may fail if:

EINVAL	The STREAM or multiplexer referenced by <i>fildev</i> is linked (directly or indirectly) downstream from a multiplexer.
ENXIO	A request was made of a non-existent device, or the request was outside the capabilities of the device.
ENXIO	A hangup occurred on the STREAM being written to.

A write to a STREAMS file may fail if an error message has been received at the STREAM head. In this case, `errno` is set to the value included in the error message.

The `writev()` function may fail if:

EINVAL	The <i>iovcnt</i> argument was less than or equal to 0 or greater than <code>IOV_MAX</code> ; one of the <code>iov_len</code> values in the <i>iov</i> array was negative; or the sum of the <code>iov_len</code> values in the <i>iov</i> array overflowed an <code>int</code> .
--------	---

In addition, `writel()`, `pwritel()`, and `writewl()` may set `errno` to:

`EFAULT` *label_p* points outside the allocated address space of the process.
The seek pointer remains unchanged if this error occurs.

USAGE The `pwrite()` function has a transitional interface for 64-bit file offsets. See `lf64(5)`.

ATTRIBUTES See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	<code>write()</code> is Async-Signal-Safe

SUMMARY OF TRUSTED SOLARIS CHANGES

If set-user-ID or get-group-ID permission bits of *fildev* are set, they are cleared by the write. The calling process may assert the `PRIV_FILE_SETID` privilege to suppress this action.

If the forced or allowed privilege set of *fildev* is not empty, it is cleared by the write. The calling process may assert the `PRIV_FILE_SETPRIV` privilege to suppress this action.

If the public object attributes flag, `FAF_PUBLIC`, of *fildev* is set, the flag is cleared by the write. The calling process may assert the `PRIV_FILE_AUDIT` privilege to suppress this action.

If the write causes the file-system free space to fall below its minimum level, the write fails. The calling process may assert the `PRIV_SYS_MINFREE` privilege to bypass this restriction.

Mandatory and discretionary access checks have already been performed when the object was opened.

**Trusted Solaris 8
4/01 Reference
Manual**

**SunOS 5.8
Reference Manual**

`chmod(2)`, `creat(2)`, `fcntl(2)`, `getrlimit(2)`, `lseek(2)`, `open(2)`, `ulimit(2)`,
`intro(3)`, `send(3SOCKET)`, `socket(3SOCKET)`

`dup(2)`, `ioctl(2)`, `pipe(2)`, `attributes(5)`, `lf64(5)`, `streamio(7I)`

pwritel(2)

NAME	write, pwrite, writev, writel, pwritel, writev1 – write on a file
SYNOPSIS	<pre>#include <unistd.h> ssize_t write(int fildes, const void *buf, size_t nbyte); ssize_t pwrite(int fildes, const void *buf, size_t nbyte, off_t offset); #include <sys/uio.h> ssize_t writev(int fildes, const struct iovec *iov, int iovcnt); #include <tsol/rdwrl.h> ssize_t writel(int fildes, void *buf, size_t nbyte, bclabel_t *label_p); ssize_t pwritel(int fildes, void *buf, size_t nbyte, off_t offset, bclabel_t *label_p); ssize_t writev1(int fildes, struct iovec *iov, int iovcnt, bclabel_t *label_p);</pre>
DESCRIPTION	<p>The <code>write()</code> function attempts to write <i>nbyte</i> bytes from the buffer pointed to by <i>buf</i> to the file associated with the open file descriptor, <i>fildes</i>.</p> <p>If <i>nbyte</i> is 0, <code>write()</code> will return 0 and have no other results if the file is a regular file; otherwise, the results are unspecified.</p> <p>On a regular file or other file capable of seeking, the actual writing of data proceeds from the position in the file indicated by the file offset associated with <i>fildes</i>. Before successful return from <code>write()</code>, the file offset is incremented by the number of bytes actually written. On a regular file, if this incremented file offset is greater than the length of the file, the length of the file will be set to this file offset.</p> <p>If the <code>O_SYNC</code> flag of the file status flags is set and <i>fildes</i> refers to a regular file, a successful <code>write()</code> does not return until the data is delivered to the underlying hardware.</p> <p>If <i>fildes</i> refers to a socket, <code>write()</code> is equivalent to <code>send(3SOCKET)</code> with no flags set.</p> <p>On a file not capable of seeking, writing always takes place starting at the current position. The value of a file offset associated with such a device is undefined.</p> <p>If the <code>O_APPEND</code> flag of the file status flags is set, the file offset will be set to the end of the file prior to each write and no intervening file modification operation will occur between changing the file offset and the write operation.</p> <p>For regular files, no data transfer will occur past the offset maximum established in the open file description with <i>fildes</i>.</p> <p>A <code>write()</code> to a regular file is blocked if mandatory file/record locking is set (see <code>chmod(2)</code>), and there is a record lock owned by another process on the segment of the file to be written:</p>

- If `O_NDELAY` or `O_NONBLOCK` is set, `write()` returns `-1` and sets `errno` to `EAGAIN`.
- If `O_NDELAY` and `O_NONBLOCK` are clear, `write()` sleeps until all blocking locks are removed or the `write()` is terminated by a signal.

If a `write()` requests that more bytes be written than there is room for—for example, if the write would exceed the process file size limit (see `getrlimit(2)` and `ulimit(2)`), the system file size limit, or the free space on the device—only as many bytes as there is room for will be written. For example, suppose there is space for 20 bytes more in a file before reaching a limit. A `write()` of 512-bytes returns 20. The next `write()` of a non-zero number of bytes gives a failure return (except as noted for pipes and FIFO below).

If `write()` is interrupted by a signal before it writes any data, it will return `-1` with `errno` set to `EINTR`.

If `write()` is interrupted by a signal after it successfully writes some data, it will return the number of bytes written.

If the value of *nbyte* is greater than `SSIZE_MAX`, the result is implementation-dependent.

After a `write()` to a regular file has successfully returned:

- Any successful `read(2)` from each byte position in the file that was modified by that write will return the data specified by the `write()` for that position until such byte positions are again modified.
- Any subsequent successful `write()` to the same byte position in the file will overwrite that file data.

Write requests to a pipe or FIFO are handled the same as a regular file with the following exceptions:

- There is no file offset associated with a pipe, hence each write request appends to the end of the pipe.
- Write requests of `{PIPE_BUF}` bytes or less are guaranteed not to be interleaved with data from other processes doing writes on the same pipe. Writes of greater than `{PIPE_BUF}` bytes may have data interleaved, on arbitrary boundaries, with writes by other processes, whether or not the `O_NONBLOCK` or `O_NDELAY` flags are set.
- If `O_NONBLOCK` and `O_NDELAY` are clear, a write request may cause the process to block, but on normal completion it returns *nbyte*.
- If `O_NONBLOCK` and `O_NDELAY` are set, `write()` does not block the process. If a `write()` request for `PIPE_BUF` or fewer bytes succeeds completely `write()` returns *nbyte*. Otherwise, if `O_NONBLOCK` is set, it returns `-1` and sets `errno` to `EAGAIN` or if `O_NDELAY` is set, it returns 0. A `write()` request for greater than `{PIPE_BUF}` bytes transfers what it can and returns the number of bytes written.

pwrite(2)

or it transfers no data and, if `O_NONBLOCK` is set, returns `-1` with `errno` set to `EAGAIN` or if `O_NDELAY` is set, it returns `0`. Finally, if a request is greater than `PIPE_BUF` bytes and all data previously written to the pipe has been read, `write()` transfers at least `PIPE_BUF` bytes.

When attempting to write to a file descriptor (other than a pipe, a FIFO, a socket, or a STREAM) that supports nonblocking writes and cannot accept the data immediately:

- If `O_NONBLOCK` and `O_NDELAY` are clear, `write()` blocks until the data can be accepted.
- If `O_NONBLOCK` or `O_NDELAY` is set, `write()` does not block the process. If some data can be written without blocking the process, `write()` writes what it can and returns the number of bytes written. Otherwise, if `O_NONBLOCK` is set, it returns `-1` and sets `errno` to `EAGAIN` or if `O_NDELAY` is set, it returns `0`.

Upon successful completion, where *nbyte* is greater than 0, `write()` will mark for update the `st_ctime` and `st_mtime` fields of the file, and if the file is a regular file, the `S_ISUID` and `S_ISGID` bits of the file mode may be cleared.

For STREAMS files (see `intro(3)` and `streamio(7I)`), the operation of `write()` is determined by the values of the minimum and maximum *nbyte* range ("packet size") accepted by the STREAM. These values are contained in the topmost STREAM module, and cannot be set or tested from user level. If *nbyte* falls within the packet size range, *nbyte* bytes are written. If *nbyte* does not fall within the range and the minimum packet size value is zero, `write()` breaks the buffer into maximum packet size segments prior to sending the data downstream (the last segment may be smaller than the maximum packet size). If *nbyte* does not fall within the range and the minimum value is non-zero, `write()` fails and sets `errno` to `ERANGE`. Writing a zero-length buffer (*nbyte* is zero) to a STREAMS device sends a zero length message with zero returned. However, writing a zero-length buffer to a pipe or FIFO sends no message and zero is returned. The user program may issue the `I_SWROPT` `ioctl(2)` to enable zero-length messages to be sent across the pipe or FIFO (see `streamio(7I)`).

When writing to a STREAM, data messages are created with a priority band of zero. When writing to a socket or to a STREAM that is not a pipe or a FIFO:

- If `O_NDELAY` and `O_NONBLOCK` are not set, and the STREAM cannot accept data (the STREAM write queue is full due to internal flow control conditions), `write()` blocks until data can be accepted.
- If `O_NDELAY` or `O_NONBLOCK` is set and the STREAM cannot accept data, `write()` returns `-1` and sets `errno` to `EAGAIN`.
- If `O_NDELAY` or `O_NONBLOCK` is set and part of the buffer has already been written when a condition occurs in which the STREAM cannot accept additional data, `write()` terminates and returns the number of bytes written.

The `write()` and `writew()` functions will fail if the STREAM head had processed an asynchronous error before the call. In this case, the value of `errno` does not reflect the result of `write()` or `writew()` but reflects the prior error.

pwrite()	<p>The pwrite() function performs the same action as write(), except that it writes into a given position without changing the file pointer. The first three arguments to pwrite() are the same as write() with the addition of a fourth argument <i>offset</i> for the desired position inside the file.</p>
writev()	<p>The writev() function performs the same action as write(), but gathers the output data from the <i>iovcnt</i> buffers specified by the members of the <i>iov</i> array: <i>iov</i>[0], <i>iov</i>[1], ..., <i>iov</i>[<i>iovcnt</i> - 1]. The <i>iovcnt</i> buffer is valid if greater than 0 and less than or equal to IOV_MAX. See intro(2) for a definition of IOV_MAX.</p> <p>The <i>iovec</i> structure contains the following members:</p> <pre>caddr_t iov_base; int iov_len;</pre> <p>Each <i>iovec</i> entry specifies the base address and length of an area in memory from which data should be written. The writev() function always writes all data from an area before proceeding to the next.</p> <p>If <i>files</i> refers to a regular file and all of the <i>iov_len</i> members in the array pointed to by <i>iov</i> are 0, writev() will return 0 and have no other effect. For other file types, the behavior is unspecified.</p> <p>If the sum of the <i>iov_len</i> values is greater than SSIZE_MAX, the operation fails and no data is transferred.</p>
writel(), pwrite(), and writev()	<p>writel(), pwrite(), and writev() perform the same actions as write(), pwrite(), and writev(), respectively, and additionally provide the CMW label <i>label_p</i> to associate with the data that is written. The label associated with the data that is written to <i>fd</i> has this restriction:</p> <ul style="list-style-type: none"> ■ If the descriptor refers to a file or a FIFO, then the sensitivity label portion of <i>label_p</i> is ignored. <p>In all other respects, the writel(), pwrite(), and writev() interfaces are analogous to the write(), pwrite(), and writev() interfaces.</p> <p>If the set-user-ID or get-group-ID bits of <i>files</i> are set, they are cleared by the write. The calling process may assert the PRIV_FILE_SETID privilege to suppress this action.</p> <p>If the forced or allowed privilege sets of <i>files</i> are not empty, they are cleared by the write. The calling process may assert the PRIV_FILE_SETPRIV privilege to suppress this action.</p> <p>If the public object attributes flag, FAF_PUBLIC, of <i>files</i> is set, the flag is cleared by the write. The calling process may assert the PRIV_FILE_AUDIT privilege to suppress this action.</p>

pwritel(2)

	<p>If the write causes the file-system free space to fall below its minimum level, the write fails. The calling process may assert the <code>PRIV_SYS_MINFREE</code> privilege to bypass this restriction.</p>																		
RETURN VALUES	<p>Upon successful completion, <code>write()</code> returns the number of bytes actually written to the file associated with <i>fil</i>. This number is never greater than <i>nbyte</i>. Otherwise, <code>-1</code> is returned, the file-pointer remains unchanged, and <code>errno</code> is set to indicate the error.</p> <p>Upon successful completion, <code>writew()</code> returns the number of bytes actually written. Otherwise, it returns <code>-1</code>, the file-pointer remains unchanged, and <code>errno</code> is set to indicate an error.</p>																		
ERRORS	<p>The <code>write()</code>, <code>pwrite()</code>, <code>writew()</code>, <code>writel()</code>, <code>pwritel()</code>, and <code>writewl()</code> functions will fail if:</p> <table><tr><td>EAGAIN</td><td>Mandatory file/record locking is set, <code>O_NDELAY</code> or <code>O_NONBLOCK</code> is set, and there is a blocking record lock; total amount of system memory available when reading using raw I/O is temporarily insufficient; an attempt is made to write to a STREAM that can not accept data with the <code>O_NDELAY</code> or <code>O_NONBLOCK</code> flag set; or a write to a pipe or FIFO of <code>PIPE_BUF</code> bytes or less is requested and less than <i>nbytes</i> of free space is available.</td></tr><tr><td>EBADF</td><td>The <i>fil</i> argument is not a valid file descriptor open for writing.</td></tr><tr><td>EDEADLK</td><td>The write was going to go to sleep and cause a deadlock situation to occur.</td></tr><tr><td>EDQUOT</td><td>The user's quota of disk blocks on the file system containing the file has been exhausted.</td></tr><tr><td>EFAULT</td><td>The <i>buf</i> argument points to an illegal address.</td></tr><tr><td>EFBIG</td><td>An attempt is made to write a file that exceeds the process's file size limit or the maximum file size (see <code>getrlimit(2)</code> and <code>ulimit(2)</code>).</td></tr><tr><td></td><td>The file is a regular file, <i>nbyte</i> is greater than 0, and the starting position is greater than or equal to the offset maximum established in the file description associated with <i>fil</i>.</td></tr><tr><td>EINTR</td><td>A signal was caught during the write operation and no data was transferred.</td></tr><tr><td>EIO</td><td>The process is in the background and is attempting to write to its controlling terminal whose <code>TOSTOP</code> flag is set, or the process is neither ignoring nor blocking <code>SIGTTOU</code> signals and the process group of the process is orphaned.</td></tr></table>	EAGAIN	Mandatory file/record locking is set, <code>O_NDELAY</code> or <code>O_NONBLOCK</code> is set, and there is a blocking record lock; total amount of system memory available when reading using raw I/O is temporarily insufficient; an attempt is made to write to a STREAM that can not accept data with the <code>O_NDELAY</code> or <code>O_NONBLOCK</code> flag set; or a write to a pipe or FIFO of <code>PIPE_BUF</code> bytes or less is requested and less than <i>nbytes</i> of free space is available.	EBADF	The <i>fil</i> argument is not a valid file descriptor open for writing.	EDEADLK	The write was going to go to sleep and cause a deadlock situation to occur.	EDQUOT	The user's quota of disk blocks on the file system containing the file has been exhausted.	EFAULT	The <i>buf</i> argument points to an illegal address.	EFBIG	An attempt is made to write a file that exceeds the process's file size limit or the maximum file size (see <code>getrlimit(2)</code> and <code>ulimit(2)</code>).		The file is a regular file, <i>nbyte</i> is greater than 0, and the starting position is greater than or equal to the offset maximum established in the file description associated with <i>fil</i> .	EINTR	A signal was caught during the write operation and no data was transferred.	EIO	The process is in the background and is attempting to write to its controlling terminal whose <code>TOSTOP</code> flag is set, or the process is neither ignoring nor blocking <code>SIGTTOU</code> signals and the process group of the process is orphaned.
EAGAIN	Mandatory file/record locking is set, <code>O_NDELAY</code> or <code>O_NONBLOCK</code> is set, and there is a blocking record lock; total amount of system memory available when reading using raw I/O is temporarily insufficient; an attempt is made to write to a STREAM that can not accept data with the <code>O_NDELAY</code> or <code>O_NONBLOCK</code> flag set; or a write to a pipe or FIFO of <code>PIPE_BUF</code> bytes or less is requested and less than <i>nbytes</i> of free space is available.																		
EBADF	The <i>fil</i> argument is not a valid file descriptor open for writing.																		
EDEADLK	The write was going to go to sleep and cause a deadlock situation to occur.																		
EDQUOT	The user's quota of disk blocks on the file system containing the file has been exhausted.																		
EFAULT	The <i>buf</i> argument points to an illegal address.																		
EFBIG	An attempt is made to write a file that exceeds the process's file size limit or the maximum file size (see <code>getrlimit(2)</code> and <code>ulimit(2)</code>).																		
	The file is a regular file, <i>nbyte</i> is greater than 0, and the starting position is greater than or equal to the offset maximum established in the file description associated with <i>fil</i> .																		
EINTR	A signal was caught during the write operation and no data was transferred.																		
EIO	The process is in the background and is attempting to write to its controlling terminal whose <code>TOSTOP</code> flag is set, or the process is neither ignoring nor blocking <code>SIGTTOU</code> signals and the process group of the process is orphaned.																		

ENOLCK	Enforced record locking was enabled and LOCK_MAX regions are already locked in the system, or the system record lock table was full and the write could not go to sleep until the blocking record lock was removed.
ENOLINK	The <i>fildev</i> argument is on a remote machine and the link to that machine is no longer active.
ENOSPC	During a write to an ordinary file, there is no free space left on the device.
ENOSR	An attempt is made to write to a STREAMS with insufficient STREAMS memory resources available in the system.
ENXIO	A hangup occurred on the STREAM being written to.
EPIPE	An attempt is made to write to a pipe or a FIFO that is not open for reading by any process, or that has only one end open (or to a file descriptor created by <code>socket(3SOCKET)</code> , using type <code>SOCK_STREAM</code> that is no longer connected to a peer endpoint). A <code>SIGPIPE</code> signal will also be sent to the process. The process dies unless special provisions were taken to catch or ignore the signal.
ERANGE	The transfer request size was outside the range supported by the STREAMS file associated with <i>fildev</i> .

The `pwrite()` function fails and the file pointer remains unchanged if:

ESPIPE	The <i>fildev</i> argument is associated with a pipe or FIFO.
--------	---

The `writev()` function will fail if:

EINVAL	The sum of the <code>iov_len</code> values in the <i>iov</i> array would overflow an <code>ssize_t</code> .
--------	---

The `write()` and `writev()` functions may fail if:

EINVAL	The STREAM or multiplexer referenced by <i>fildev</i> is linked (directly or indirectly) downstream from a multiplexer.
ENXIO	A request was made of a non-existent device, or the request was outside the capabilities of the device.
ENXIO	A hangup occurred on the STREAM being written to.

A write to a STREAMS file may fail if an error message has been received at the STREAM head. In this case, `errno` is set to the value included in the error message.

The `writev()` function may fail if:

EINVAL	The <i>iovcnt</i> argument was less than or equal to 0 or greater than <code>IOV_MAX</code> ; one of the <code>iov_len</code> values in the <i>iov</i> array was negative; or the sum of the <code>iov_len</code> values in the <i>iov</i> array overflowed an <code>int</code> .
--------	---

`pwritel(2)`

In addition, `writel()`, `pwritel()`, and `writevl()` may set `errno` to:

`EFAULT` *label_p* points outside the allocated address space of the process.
The seek pointer remains unchanged if this error occurs.

USAGE The `pwrite()` function has a transitional interface for 64-bit file offsets. See `lf64(5)`.

ATTRIBUTES See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	<code>write()</code> is Async-Signal-Safe

**SUMMARY OF
TRUSTED
SOLARIS
CHANGES**

If set-user-ID or get-group-ID permission bits of *fildev* are set, they are cleared by the write. The calling process may assert the `PRIV_FILE_SETID` privilege to suppress this action.

If the forced or allowed privilege set of *fildev* is not empty, it is cleared by the write. The calling process may assert the `PRIV_FILE_SETPRIV` privilege to suppress this action.

If the public object attributes flag, `FAF_PUBLIC`, of *fildev* is set, the flag is cleared by the write. The calling process may assert the `PRIV_FILE_AUDIT` privilege to suppress this action.

If the write causes the file-system free space to fall below its minimum level, the write fails. The calling process may assert the `PRIV_SYS_MINFREE` privilege to bypass this restriction.

Mandatory and discretionary access checks have already been performed when the object was opened.

**Trusted Solaris 8
4/01 Reference
Manual**
**SunOS 5.8
Reference Manual**

`chmod(2)`, `creat(2)`, `fcntl(2)`, `getrlimit(2)`, `lseek(2)`, `open(2)`, `ulimit(2)`,
`intro(3)`, `send(3SOCKET)`, `socket(3SOCKET)`

`dup(2)`, `ioctl(2)`, `pipe(2)`, `attributes(5)`, `lf64(5)`, `streamio(7I)`

NAME	read, readl, pread, preadl, readv, readvl – read from a file
SYNOPSIS	<pre> #include <sys/types.h> #include <sys/uio.h> #include <unistd.h> ssize_t read(int fildes, void *buf, size_t nbyte); ssize_t pread(int fildes, void *buf, size_t nbyte, off_t offset); ssize_t readv(int fildes, struct iovec *iov, int iovcnt); #include <tsol/rdwrl.h> ssize_t readl(int fildes, void *buf, size_t nbyte, bclabel_t *label_p); ssize_t preadl(int fildes, void *buf, size_t nbyte, off_t offset, bclabel_t *label_p); ssize_t readvl(int fildes, struct iovec *iov, int iovcnt, bclabel_t *label_p); </pre>
DESCRIPTION	<p>The <code>read()</code> function attempts to read <i>nbyte</i> bytes from the file associated with the open file descriptor, <i>fildes</i>, into the buffer pointed to by <i>buf</i>.</p> <p>If <i>nbyte</i> is 0, <code>read()</code> will return 0 and have no other results.</p> <p>On files that support seeking (for example, a regular file), the <code>read()</code> starts at a position in the file given by the file offset associated with <i>fildes</i>. The file offset is incremented by the number of bytes actually read.</p> <p>Files that do not support seeking (for example, terminals) always read from the current position. The value of a file offset associated with such a file is undefined.</p> <p>If <i>fildes</i> refers to a socket, <code>read()</code> is equivalent to <code>recv(3SOCKET)</code> with no flags set.</p> <p>No data transfer will occur past the current end-of-file. If the starting position is at or after the end-of-file, 0 will be returned. If the file refers to a device special file, the result of subsequent <code>read()</code> requests is implementation-dependent.</p> <p>If the value of <i>nbyte</i> is greater than <code>SSIZE_MAX</code>, the result is implementation-dependent.</p> <p>When attempting to read from a regular file with mandatory file/record locking set (see <code>chmod(2)</code>), and there is a write lock owned by another process on the segment of the file to be read:</p> <ul style="list-style-type: none"> ■ If <code>O_NDELAY</code> or <code>O_NONBLOCK</code> is set, <code>read()</code> returns -1 and sets <code>errno</code> to <code>EAGAIN</code>. ■ If <code>O_NDELAY</code> and <code>O_NONBLOCK</code> are clear, <code>read()</code> sleeps until the blocking record lock is removed. <p>When attempting to read from an empty pipe (or FIFO):</p>

read(2)

- If no process has the pipe open for writing, `read()` returns 0 to indicate end-of-file.
- If some process has the pipe open for writing and `O_NDELAY` is set, `read()` returns 0.
- If some process has the pipe open for writing and `O_NONBLOCK` is set, `read()` returns -1 and sets `errno` to `EAGAIN`.
- If `O_NDELAY` and `O_NONBLOCK` are clear, `read()` blocks until data is written to the pipe or the pipe is closed by all processes that had opened the pipe for writing.

When attempting to read a file associated with a terminal that has no data currently available:

- If `O_NDELAY` is set, `read()` returns 0.
- If `O_NONBLOCK` is set, `read()` returns -1 and sets `errno` to `EAGAIN`.
- If `O_NDELAY` and `O_NONBLOCK` are clear, `read()` blocks until data become available.

When attempting to read a file associated with a socket or a stream that is not a pipe, a FIFO, or a terminal, and the file has no data currently available:

- If `O_NDELAY` or `O_NONBLOCK` is set, `read()` returns -1 and sets `errno` to `EAGAIN`.
- If `O_NDELAY` and `O_NONBLOCK` are clear, `read()` blocks until data becomes available.

The `read()` function reads data previously written to a file. If any portion of a regular file prior to the end-of-file has not been written, `read()` returns bytes with value 0. For example, `lseek(2)` allows the file offset to be set beyond the end of existing data in the file. If data is later written at this point, subsequent reads in the gap between the previous end of data and the newly written data will return bytes with value 0 until data is written into the gap.

For regular files, no data transfer will occur past the offset maximum established in the open file description associated with *fil*des.

Upon successful completion, where *nbyte* is greater than 0, `read()` will mark for update the `st_atime` field of the file, and return the number of bytes read. This number will never be greater than *nbyte*. The value returned may be less than *nbyte* if the number of bytes left in the file is less than *nbyte*, if the `read()` request was interrupted by a signal, or if the file is a pipe or FIFO or special file and has fewer than *nbyte* bytes immediately available for reading. For example, a `read()` from a file associated with a terminal may return one typed line of data.

If a `read()` is interrupted by a signal before it reads any data, it will return -1 with `errno` set to `EINTR`.

If a `read()` is interrupted by a signal after it has successfully read some data, it will return the number of bytes read.

A `read()` or `readv()` from a STREAMS (see `intro(2)`) file can read data in three different modes: byte-stream mode, message-nondiscard mode, and message-discard mode. The default is byte-stream mode. This can be changed using the `I_SRDOPT` `ioctl(2)` request, and can be tested with the `I_GRDOPT` `ioctl()`. In byte-stream mode, `read()` retrieves data from the STREAM until as many bytes as were requested are transferred, or until there is no more data to be retrieved. Byte-stream mode ignores message boundaries.

In STREAMS message-nondiscard mode, `read()` retrieves data until as many bytes as were requested are transferred, or until a message boundary is reached. If `read()` does not retrieve all the data in a message, the remaining data is left on the STREAM, and can be retrieved by the next `read()` call. Message-discard mode also retrieves data until as many bytes as were requested are transferred, or a message boundary is reached. However, unread data remaining in a message after the `read()` returns is discarded, and is not available for a subsequent `read()`, `readv()` or `getmsg(2)` call.

How `read()` handles zero-byte STREAMS messages is determined by the current read mode setting. In byte-stream mode, `read()` accepts data until it has read *nbyte* bytes, or until there is no more data to read, or until a zero-byte message block is encountered. The `read()` function then returns the number of bytes read, and places the zero-byte message back on the STREAM to be retrieved by the next `read()`, `readv()` or `getmsg(2)`. In message-nondiscard mode or message-discard mode, a zero-byte message returns 0 and the message is removed from the STREAM. When a zero-byte message is read as the first message on a STREAM, the message is removed from the STREAM and 0 is returned, regardless of the read mode.

A `read()` from a STREAMS file returns the data in the message at the front of the STREAM head read queue, regardless of the priority band of the message.

By default, STREAMS are in control-normal mode, in which a `read()` from a STREAMS file can only process messages that contain a data part but do not contain a control part. The `read()` fails if a message containing a control part is encountered at the STREAM head. This default action can be changed by placing the STREAM in either control-data mode or control-discard mode with the `I_SRDOPT` `ioctl()` command. In control-data mode, `read()` converts any control part to data and passes it to the application before passing any data part originally present in the same message. In control-discard mode, `read()` discards message control parts but returns to the process any data part in the message.

`readl()`, `preadl()`, and `readvl()` perform the same actions as `read()`, `pread()`, and `readv()`, respectively, and additionally return in *label_p* the CMW label of the data read. The label returned is determined according to these conditions:

- If the descriptor refers to a regular file or FIFO, the sensitivity label portion of *label_p* is set to the sensitivity label associated with the filesystem object.

read(2)

	<p>In all other respects, the <code>readl()</code>, <code>preadl()</code>, and <code>readvl()</code> interfaces are analogous to the <code>read()</code>, <code>pread()</code>, and <code>readv()</code> interfaces.</p> <p>In the Solaris environment, <code>read()</code> normally allows a process to read the contents of directories on some local file systems. This functionality is not supported in the Trusted Solaris operating environment. If the file descriptor refers to a directory, <code>read()</code> will return <code>EISDIR</code>.</p> <p>The last access time is updated only when the calling process has both mandatory read and write access to the filesystem object. There is no privilege to override this restriction.</p> <p>In addition, <code>read()</code> and <code>readv()</code> will fail if the STREAM head had processed an asynchronous error before the call. In this case, the value of <code>errno</code> does not reflect the result of <code>read()</code> or <code>readv()</code> but reflects the prior error. If a hangup occurs on the STREAM being read, <code>read()</code> continues to operate normally until the STREAM head read queue is empty. Thereafter, it returns 0.</p>		
<code>readv()</code>	<p>The <code>readv()</code> function is equivalent to <code>read()</code>, but places the input data into the <i>iovcnt</i> buffers specified by the members of the <i>iov</i> array: <i>iov0</i>, <i>iov1</i>, ..., <i>iov[iovcnt-1]</i>. The <i>iovcnt</i> argument is valid if greater than 0 and less than or equal to <code>IOV_MAX</code>.</p> <p>The <code>iovec</code> structure contains the following members:</p> <pre>caddr_t iov_base; int iov_len;</pre> <p>Each <code>iovec</code> entry specifies the base address and length of an area in memory where data should be placed. The <code>readv()</code> function always fills an area completely before proceeding to the next.</p> <p>Upon successful completion, <code>readv()</code> marks for update the <code>st_atime</code> field of the file.</p>		
<code>pread()</code>	<p>The <code>pread()</code> function performs the same action as <code>read()</code>, except that it reads from a given position in the file without changing the file pointer. The first three arguments to <code>pread()</code> are the same as <code>read()</code> with the addition of a fourth argument <i>offset</i> for the desired position inside the file. <code>pread()</code> will read up to the maximum offset value that can be represented in an <code>off_t</code> for regular files. An attempt to perform a <code>pread()</code> on a file that is incapable of seeking results in an error.</p>		
RETURN VALUES	<p>Upon successful completion, <code>read()</code> and <code>readv()</code> return a non-negative integer indicating the number of bytes actually read. Otherwise, the functions return <code>-1</code> and set <code>errno</code> to indicate the error.</p>		
ERRORS	<p><code>read()</code>, <code>readl()</code>, <code>pread()</code>, <code>preadl()</code>, <code>readv()</code>, and <code>readvl()</code> fail if any of these conditions is true:</p> <table><tr><td><code>EAGAIN</code></td><td>Mandatory file/record locking was set, <code>O_NDELAY</code> or <code>O_NONBLOCK</code> was set, and there was a blocking record lock; total</td></tr></table>	<code>EAGAIN</code>	Mandatory file/record locking was set, <code>O_NDELAY</code> or <code>O_NONBLOCK</code> was set, and there was a blocking record lock; total
<code>EAGAIN</code>	Mandatory file/record locking was set, <code>O_NDELAY</code> or <code>O_NONBLOCK</code> was set, and there was a blocking record lock; total		

	amount of system memory available when reading using raw I/O is temporarily insufficient; no data is waiting to be read on a file associated with a tty device and <code>O_NONBLOCK</code> was set; or no message is waiting to be read on a stream and <code>O_NDELAY</code> or <code>O_NONBLOCK</code> was set.
EBADF	The <i>fdes</i> argument is not a valid file descriptor open for reading.
EBADMSG	Message waiting to be read on a stream is not a data message.
EDEADLK	The read was going to go to sleep and cause a deadlock to occur.
EFAULT	The <i>buf</i> argument points to an illegal address.
EINTR	A signal was caught during the read operation and no data was transferred.
EINVAL	An attempt was made to read from a stream linked to a multiplexor.
EIO	A physical I/O error has occurred, or the process is in a background process group and is attempting to read from its controlling terminal, and either the process is ignoring or blocking the SIGTTIN signal or the process group of the process is orphaned.
EISDIR	The <i>fdes</i> argument refers to a directory on a file system type that does not support read operations on directories.
ENOLCK	The system record lock table was full, so the <code>read()</code> or <code>readv()</code> could not go to sleep until the blocking record lock was removed.
ENOLINK	The <i>fdes</i> argument is on a remote machine and the link to that machine is no longer active.
ENXIO	The device associated with <i>fdes</i> is a block special or character special file and the value of the file pointer is out of range.
The <code>read()</code> and <code>readv()</code> functions will fail if:	
EOVERFLOW	The file is a regular file, <i>nbyte</i> is greater than 0, the starting position is before the end-of-file, and the starting position is greater than or equal to the offset maximum established in the open file description associated with <i>fdes</i> .
The <code>readv()</code> function may fail if:	
EFAULT	The <i>iov</i> argument points outside the allocated address space.
EINVAL	The <i>iovcnt</i> argument was less than or equal to 0, or greater than or equal to <code>IOV_MAX</code> . (See <code>intro(2)</code> for a definition of <code>IOV_MAX</code>).
EINVAL	The sum of the <code>iov_len</code> values in the <i>iov</i> array overflowed an int.
The <code>pread()</code> function will fail and the file pointer remain unchanged if:	

read(2)

USAGE EFAULT *label_p* points to an illegal address.

ATTRIBUTES The `pread()` function has a transitional interface for 64-bit file offsets. See `lfs64(5)`.

See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	<code>read()</code> is Async-Signal-Safe

SUMMARY OF TRUSTED SOLARIS CHANGES

`readl()`, `preadl()`, and `readvl()` return in the buffer that was referenced by *label_p*, the CMW label associated with the data that was read.

The last access time is updated only when the calling process has both mandatory read and write access to the filesystem object. There is no privilege to override this restriction.

For conduits, a sensitivity label is associated with each byte of data.

Trusted Solaris 8 4/01 Reference Manual

`intro(2)`, `chmod(2)`, `creat(2)`, `fcntl(2)`, `open(2)`

`dup(2)`, `getmsg(2)`, `ioctl(2)`, `pipe(2)`, `attributes(5)`, `streamio(7I)`, `termio(7I)`

NAME	read, readl, pread, preadl, readv, readvl – read from a file
SYNOPSIS	<pre> #include <sys/types.h> #include <sys/uio.h> #include <unistd.h> ssize_t read(int fildes, void *buf, size_t nbyte); ssize_t pread(int fildes, void *buf, size_t nbyte, off_t offset); ssize_t readv(int fildes, struct iovec *iov, int iovcnt); #include <tsol/rdwrl.h> ssize_t readl(int fildes, void *buf, size_t nbyte, bclabel_t *label_p); ssize_t preadl(int fildes, void *buf, size_t nbyte, off_t offset, bclabel_t *label_p); ssize_t readvl(int fildes, struct iovec *iov, int iovcnt, bclabel_t *label_p); </pre>
DESCRIPTION	<p>The <code>read()</code> function attempts to read <i>nbyte</i> bytes from the file associated with the open file descriptor, <i>fildes</i>, into the buffer pointed to by <i>buf</i>.</p> <p>If <i>nbyte</i> is 0, <code>read()</code> will return 0 and have no other results.</p> <p>On files that support seeking (for example, a regular file), the <code>read()</code> starts at a position in the file given by the file offset associated with <i>fildes</i>. The file offset is incremented by the number of bytes actually read.</p> <p>Files that do not support seeking (for example, terminals) always read from the current position. The value of a file offset associated with such a file is undefined.</p> <p>If <i>fildes</i> refers to a socket, <code>read()</code> is equivalent to <code>recv(3SOCKET)</code> with no flags set.</p> <p>No data transfer will occur past the current end-of-file. If the starting position is at or after the end-of-file, 0 will be returned. If the file refers to a device special file, the result of subsequent <code>read()</code> requests is implementation-dependent.</p> <p>If the value of <i>nbyte</i> is greater than <code>SSIZE_MAX</code>, the result is implementation-dependent.</p> <p>When attempting to read from a regular file with mandatory file/record locking set (see <code>chmod(2)</code>), and there is a write lock owned by another process on the segment of the file to be read:</p> <ul style="list-style-type: none"> ■ If <code>O_NDELAY</code> or <code>O_NONBLOCK</code> is set, <code>read()</code> returns -1 and sets <code>errno</code> to <code>EAGAIN</code>. ■ If <code>O_NDELAY</code> and <code>O_NONBLOCK</code> are clear, <code>read()</code> sleeps until the blocking record lock is removed. <p>When attempting to read from an empty pipe (or FIFO):</p>

readl(2)

- If no process has the pipe open for writing, `read()` returns 0 to indicate end-of-file.
- If some process has the pipe open for writing and `O_NDELAY` is set, `read()` returns 0.
- If some process has the pipe open for writing and `O_NONBLOCK` is set, `read()` returns -1 and sets `errno` to `EAGAIN`.
- If `O_NDELAY` and `O_NONBLOCK` are clear, `read()` blocks until data is written to the pipe or the pipe is closed by all processes that had opened the pipe for writing.

When attempting to read a file associated with a terminal that has no data currently available:

- If `O_NDELAY` is set, `read()` returns 0.
- If `O_NONBLOCK` is set, `read()` returns -1 and sets `errno` to `EAGAIN`.
- If `O_NDELAY` and `O_NONBLOCK` are clear, `read()` blocks until data become available.

When attempting to read a file associated with a socket or a stream that is not a pipe, a FIFO, or a terminal, and the file has no data currently available:

- If `O_NDELAY` or `O_NONBLOCK` is set, `read()` returns -1 and sets `errno` to `EAGAIN`.
- If `O_NDELAY` and `O_NONBLOCK` are clear, `read()` blocks until data becomes available.

The `read()` function reads data previously written to a file. If any portion of a regular file prior to the end-of-file has not been written, `read()` returns bytes with value 0. For example, `lseek(2)` allows the file offset to be set beyond the end of existing data in the file. If data is later written at this point, subsequent reads in the gap between the previous end of data and the newly written data will return bytes with value 0 until data is written into the gap.

For regular files, no data transfer will occur past the offset maximum established in the open file description associated with *fil*des.

Upon successful completion, where *nbyte* is greater than 0, `read()` will mark for update the `st_atime` field of the file, and return the number of bytes read. This number will never be greater than *nbyte*. The value returned may be less than *nbyte* if the number of bytes left in the file is less than *nbyte*, if the `read()` request was interrupted by a signal, or if the file is a pipe or FIFO or special file and has fewer than *nbyte* bytes immediately available for reading. For example, a `read()` from a file associated with a terminal may return one typed line of data.

If a `read()` is interrupted by a signal before it reads any data, it will return -1 with `errno` set to `EINTR`.

If a `read()` is interrupted by a signal after it has successfully read some data, it will return the number of bytes read.

A `read()` or `readv()` from a STREAMS (see `intro(2)`) file can read data in three different modes: byte-stream mode, message-nondiscard mode, and message-discard mode. The default is byte-stream mode. This can be changed using the `I_SRDOPT ioctl(2)` request, and can be tested with the `I_GRDOPT ioctl()`. In byte-stream mode, `read()` retrieves data from the STREAM until as many bytes as were requested are transferred, or until there is no more data to be retrieved. Byte-stream mode ignores message boundaries.

In STREAMS message-nondiscard mode, `read()` retrieves data until as many bytes as were requested are transferred, or until a message boundary is reached. If `read()` does not retrieve all the data in a message, the remaining data is left on the STREAM, and can be retrieved by the next `read()` call. Message-discard mode also retrieves data until as many bytes as were requested are transferred, or a message boundary is reached. However, unread data remaining in a message after the `read()` returns is discarded, and is not available for a subsequent `read()`, `readv()` or `getmsg(2)` call.

How `read()` handles zero-byte STREAMS messages is determined by the current read mode setting. In byte-stream mode, `read()` accepts data until it has read *nbyte* bytes, or until there is no more data to read, or until a zero-byte message block is encountered. The `read()` function then returns the number of bytes read, and places the zero-byte message back on the STREAM to be retrieved by the next `read()`, `readv()` or `getmsg(2)`. In message-nondiscard mode or message-discard mode, a zero-byte message returns 0 and the message is removed from the STREAM. When a zero-byte message is read as the first message on a STREAM, the message is removed from the STREAM and 0 is returned, regardless of the read mode.

A `read()` from a STREAMS file returns the data in the message at the front of the STREAM head read queue, regardless of the priority band of the message.

By default, STREAMS are in control-normal mode, in which a `read()` from a STREAMS file can only process messages that contain a data part but do not contain a control part. The `read()` fails if a message containing a control part is encountered at the STREAM head. This default action can be changed by placing the STREAM in either control-data mode or control-discard mode with the `I_SRDOPT ioctl()` command. In control-data mode, `read()` converts any control part to data and passes it to the application before passing any data part originally present in the same message. In control-discard mode, `read()` discards message control parts but returns to the process any data part in the message.

`readl()`, `preadl()`, and `readvl()` perform the same actions as `read()`, `pread()`, and `readv()`, respectively, and additionally return in *label_p* the CMW label of the data read. The label returned is determined according to these conditions:

- If the descriptor refers to a regular file or FIFO, the sensitivity label portion of *label_p* is set to the sensitivity label associated with the filesystem object.

readl(2)

	<p>In all other respects, the <code>readl()</code>, <code>preadl()</code>, and <code>readvl()</code> interfaces are analogous to the <code>read()</code>, <code>pread()</code>, and <code>readv()</code> interfaces.</p> <p>In the Solaris environment, <code>read()</code> normally allows a process to read the contents of directories on some local file systems. This functionality is not supported in the Trusted Solaris operating environment. If the file descriptor refers to a directory, <code>read()</code> will return <code>EISDIR</code>.</p> <p>The last access time is updated only when the calling process has both mandatory read and write access to the filesystem object. There is no privilege to override this restriction.</p> <p>In addition, <code>read()</code> and <code>readv()</code> will fail if the STREAM head had processed an asynchronous error before the call. In this case, the value of <code>errno</code> does not reflect the result of <code>read()</code> or <code>readv()</code> but reflects the prior error. If a hangup occurs on the STREAM being read, <code>read()</code> continues to operate normally until the STREAM head read queue is empty. Thereafter, it returns 0.</p>		
<code>readv()</code>	<p>The <code>readv()</code> function is equivalent to <code>read()</code>, but places the input data into the <i>iovcnt</i> buffers specified by the members of the <i>iov</i> array: <i>iov0</i>, <i>iov1</i>, ..., <i>iov[iovcnt-1]</i>. The <i>iovcnt</i> argument is valid if greater than 0 and less than or equal to <code>IOV_MAX</code>.</p> <p>The <code>iovec</code> structure contains the following members:</p> <pre>caddr_t iov_base; int iov_len;</pre> <p>Each <code>iovec</code> entry specifies the base address and length of an area in memory where data should be placed. The <code>readv()</code> function always fills an area completely before proceeding to the next.</p> <p>Upon successful completion, <code>readv()</code> marks for update the <code>st_atime</code> field of the file.</p>		
<code>pread()</code>	<p>The <code>pread()</code> function performs the same action as <code>read()</code>, except that it reads from a given position in the file without changing the file pointer. The first three arguments to <code>pread()</code> are the same as <code>read()</code> with the addition of a fourth argument <i>offset</i> for the desired position inside the file. <code>pread()</code> will read up to the maximum offset value that can be represented in an <code>off_t</code> for regular files. An attempt to perform a <code>pread()</code> on a file that is incapable of seeking results in an error.</p>		
RETURN VALUES	<p>Upon successful completion, <code>read()</code> and <code>readv()</code> return a non-negative integer indicating the number of bytes actually read. Otherwise, the functions return -1 and set <code>errno</code> to indicate the error.</p>		
ERRORS	<p><code>read()</code>, <code>readl()</code>, <code>pread()</code>, <code>preadl()</code>, <code>readv()</code>, and <code>readvl()</code> fail if any of these conditions is true:</p> <table><tr><td><code>EAGAIN</code></td><td>Mandatory file/record locking was set, <code>O_NDELAY</code> or <code>O_NONBLOCK</code> was set, and there was a blocking record lock; total</td></tr></table>	<code>EAGAIN</code>	Mandatory file/record locking was set, <code>O_NDELAY</code> or <code>O_NONBLOCK</code> was set, and there was a blocking record lock; total
<code>EAGAIN</code>	Mandatory file/record locking was set, <code>O_NDELAY</code> or <code>O_NONBLOCK</code> was set, and there was a blocking record lock; total		

	amount of system memory available when reading using raw I/O is temporarily insufficient; no data is waiting to be read on a file associated with a tty device and <code>O_NONBLOCK</code> was set; or no message is waiting to be read on a stream and <code>O_NDELAY</code> or <code>O_NONBLOCK</code> was set.
EBADF	The <i>fil</i> des argument is not a valid file descriptor open for reading.
EBADMSG	Message waiting to be read on a stream is not a data message.
EDEADLK	The read was going to go to sleep and cause a deadlock to occur.
EFAULT	The <i>buf</i> argument points to an illegal address.
EINTR	A signal was caught during the read operation and no data was transferred.
EINVAL	An attempt was made to read from a stream linked to a multiplexor.
EIO	A physical I/O error has occurred, or the process is in a background process group and is attempting to read from its controlling terminal, and either the process is ignoring or blocking the SIGTTIN signal or the process group of the process is orphaned.
EISDIR	The <i>fil</i> des argument refers to a directory on a file system type that does not support read operations on directories.
ENOLCK	The system record lock table was full, so the <code>read()</code> or <code>readv()</code> could not go to sleep until the blocking record lock was removed.
ENOLINK	The <i>fil</i> des argument is on a remote machine and the link to that machine is no longer active.
ENXIO	The device associated with <i>fil</i> des is a block special or character special file and the value of the file pointer is out of range.
The <code>read()</code> and <code>readv()</code> functions will fail if:	
EOVERFLOW	The file is a regular file, <i>nbyte</i> is greater than 0, the starting position is before the end-of-file, and the starting position is greater than or equal to the offset maximum established in the open file description associated with <i>fil</i> des.
The <code>readv()</code> function may fail if:	
EFAULT	The <i>iov</i> argument points outside the allocated address space.
EINVAL	The <i>iovcnt</i> argument was less than or equal to 0, or greater than or equal to <code>IOV_MAX</code> . (See <code>intro(2)</code> for a definition of <code>IOV_MAX</code>).
EINVAL	The sum of the <i>iov_len</i> values in the <i>iov</i> array overflowed an int.
The <code>pread()</code> function will fail and the file pointer remain unchanged if:	

readl(2)

USAGE EFAULT *label_p* points to an illegal address.

ATTRIBUTES The `pread()` function has a transitional interface for 64-bit file offsets. See `lf64(5)`.

See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	<code>read()</code> is Async-Signal-Safe

SUMMARY OF TRUSTED SOLARIS CHANGES

`readl()`, `preadl()`, and `readvl()` return in the buffer that was referenced by *label_p*, the CMW label associated with the data that was read.

The last access time is updated only when the calling process has both mandatory read and write access to the filesystem object. There is no privilege to override this restriction.

For conduits, a sensitivity label is associated with each byte of data.

Trusted Solaris 8 4/01 Reference Manual

`intro(2)`, `chmod(2)`, `creat(2)`, `fcntl(2)`, `open(2)`

`dup(2)`, `getmsg(2)`, `ioctl(2)`, `pipe(2)`, `attributes(5)`, `streamio(7I)`, `termio(7I)`

NAME	readlink – read the contents of a symbolic link																								
SYNOPSIS	<pre>#include <unistd.h> int readlink(const char *path, char *buf, size_t bufsiz);</pre>																								
DESCRIPTION	The <code>readlink()</code> function places the contents of the symbolic link referred to by <i>path</i> in the buffer <i>buf</i> which has size <i>bufsiz</i> . If the number of bytes in the symbolic link is less than <i>bufsiz</i> , the contents of the remainder of <i>buf</i> are unspecified.																								
RETURN VALUES	Upon successful completion, <code>readlink()</code> returns the count of bytes placed in the buffer. Otherwise, it returns <code>-1</code> , leaves the buffer unchanged, and sets <code>errno</code> to indicate the error.																								
ERRORS	<p>The <code>readlink()</code> function will fail if:</p> <table> <tr> <td>EACCES</td><td>Search permission is denied for a component of the path prefix of <i>path</i>. To override this restriction, the calling process may assert one or both of these privileges: <code>PRIV_FILE_DAC_SEARCH</code> and <code>PRIV_FILE_MAC_SEARCH</code>.</td></tr> <tr> <td></td><td>Read permission is denied to the link. To override this restriction, the calling process may assert the <code>PRIV_FILE_MAC_READ</code> privilege.</td></tr> <tr> <td>EFAULT</td><td><i>path</i> or <i>buf</i> points to an illegal address.</td></tr> <tr> <td>EINVAL</td><td>The <i>path</i> argument names a file that is not a symbolic link.</td></tr> <tr> <td>EIO</td><td>An I/O error occurred while reading from the file system.</td></tr> <tr> <td>ENOENT</td><td>A component of <i>path</i> does not name an existing file or <i>path</i> is an empty string.</td></tr> <tr> <td>ELOOP</td><td>Too many symbolic links were encountered in resolving <i>path</i>.</td></tr> <tr> <td>ENAMETOOLONG</td><td>The length of <i>path</i> exceeds <code>PATH_MAX</code>, or a pathname component is longer than <code>NAME_MAX</code> while <code>_POSIX_NO_TRUNC</code> is in effect.</td></tr> <tr> <td>ENOTDIR</td><td>A component of the path prefix is not a directory.</td></tr> <tr> <td>ENOSYS</td><td>The file system does not support symbolic links.</td></tr> </table> <p>The <code>readlink()</code> function may fail if:</p> <table> <tr> <td>EACCES</td><td>Read permission is denied for the directory.</td></tr> <tr> <td>ENAMETOOLONG</td><td>Path name resolution of a symbolic link produced an intermediate result whose length exceeds <code>PATH_MAX</code>.</td></tr> </table>	EACCES	Search permission is denied for a component of the path prefix of <i>path</i> . To override this restriction, the calling process may assert one or both of these privileges: <code>PRIV_FILE_DAC_SEARCH</code> and <code>PRIV_FILE_MAC_SEARCH</code> .		Read permission is denied to the link. To override this restriction, the calling process may assert the <code>PRIV_FILE_MAC_READ</code> privilege.	EFAULT	<i>path</i> or <i>buf</i> points to an illegal address.	EINVAL	The <i>path</i> argument names a file that is not a symbolic link.	EIO	An I/O error occurred while reading from the file system.	ENOENT	A component of <i>path</i> does not name an existing file or <i>path</i> is an empty string.	ELOOP	Too many symbolic links were encountered in resolving <i>path</i> .	ENAMETOOLONG	The length of <i>path</i> exceeds <code>PATH_MAX</code> , or a pathname component is longer than <code>NAME_MAX</code> while <code>_POSIX_NO_TRUNC</code> is in effect.	ENOTDIR	A component of the path prefix is not a directory.	ENOSYS	The file system does not support symbolic links.	EACCES	Read permission is denied for the directory.	ENAMETOOLONG	Path name resolution of a symbolic link produced an intermediate result whose length exceeds <code>PATH_MAX</code> .
EACCES	Search permission is denied for a component of the path prefix of <i>path</i> . To override this restriction, the calling process may assert one or both of these privileges: <code>PRIV_FILE_DAC_SEARCH</code> and <code>PRIV_FILE_MAC_SEARCH</code> .																								
	Read permission is denied to the link. To override this restriction, the calling process may assert the <code>PRIV_FILE_MAC_READ</code> privilege.																								
EFAULT	<i>path</i> or <i>buf</i> points to an illegal address.																								
EINVAL	The <i>path</i> argument names a file that is not a symbolic link.																								
EIO	An I/O error occurred while reading from the file system.																								
ENOENT	A component of <i>path</i> does not name an existing file or <i>path</i> is an empty string.																								
ELOOP	Too many symbolic links were encountered in resolving <i>path</i> .																								
ENAMETOOLONG	The length of <i>path</i> exceeds <code>PATH_MAX</code> , or a pathname component is longer than <code>NAME_MAX</code> while <code>_POSIX_NO_TRUNC</code> is in effect.																								
ENOTDIR	A component of the path prefix is not a directory.																								
ENOSYS	The file system does not support symbolic links.																								
EACCES	Read permission is denied for the directory.																								
ENAMETOOLONG	Path name resolution of a symbolic link produced an intermediate result whose length exceeds <code>PATH_MAX</code> .																								

readlink(2)

USAGE

Portable applications should not assume that the returned contents of the symbolic link are null-terminated.

**SUMMARY OF
TRUSTED**

Appropriate privilege is required to override access checks.

STATISTICS
Trusted Squares
4/01 Reference
Manual

stat(2), symlink(2)

NAME	read, readl, pread, preadl, readv, readvl – read from a file
SYNOPSIS	<pre> #include <sys/types.h> #include <sys/uio.h> #include <unistd.h> ssize_t read(int fildes, void *buf, size_t nbyte); ssize_t pread(int fildes, void *buf, size_t nbyte, off_t offset); ssize_t readv(int fildes, struct iovec *iov, int iovcnt); #include <tsol/rdwrl.h> ssize_t readl(int fildes, void *buf, size_t nbyte, bclabel_t *label_p); ssize_t preadl(int fildes, void *buf, size_t nbyte, off_t offset, bclabel_t *label_p); ssize_t readvl(int fildes, struct iovec *iov, int iovcnt, bclabel_t *label_p); </pre>
DESCRIPTION	<p>The <code>read()</code> function attempts to read <i>nbyte</i> bytes from the file associated with the open file descriptor, <i>fildes</i>, into the buffer pointed to by <i>buf</i>.</p> <p>If <i>nbyte</i> is 0, <code>read()</code> will return 0 and have no other results.</p> <p>On files that support seeking (for example, a regular file), the <code>read()</code> starts at a position in the file given by the file offset associated with <i>fildes</i>. The file offset is incremented by the number of bytes actually read.</p> <p>Files that do not support seeking (for example, terminals) always read from the current position. The value of a file offset associated with such a file is undefined.</p> <p>If <i>fildes</i> refers to a socket, <code>read()</code> is equivalent to <code>recv(3SOCKET)</code> with no flags set.</p> <p>No data transfer will occur past the current end-of-file. If the starting position is at or after the end-of-file, 0 will be returned. If the file refers to a device special file, the result of subsequent <code>read()</code> requests is implementation-dependent.</p> <p>If the value of <i>nbyte</i> is greater than <code>SSIZE_MAX</code>, the result is implementation-dependent.</p> <p>When attempting to read from a regular file with mandatory file/record locking set (see <code>chmod(2)</code>), and there is a write lock owned by another process on the segment of the file to be read:</p> <ul style="list-style-type: none"> ■ If <code>O_NDELAY</code> or <code>O_NONBLOCK</code> is set, <code>read()</code> returns -1 and sets <code>errno</code> to <code>EAGAIN</code>. ■ If <code>O_NDELAY</code> and <code>O_NONBLOCK</code> are clear, <code>read()</code> sleeps until the blocking record lock is removed. <p>When attempting to read from an empty pipe (or FIFO):</p>

readv(2)

- If no process has the pipe open for writing, `read()` returns 0 to indicate end-of-file.
- If some process has the pipe open for writing and `O_NDELAY` is set, `read()` returns 0.
- If some process has the pipe open for writing and `O_NONBLOCK` is set, `read()` returns -1 and sets `errno` to `EAGAIN`.
- If `O_NDELAY` and `O_NONBLOCK` are clear, `read()` blocks until data is written to the pipe or the pipe is closed by all processes that had opened the pipe for writing.

When attempting to read a file associated with a terminal that has no data currently available:

- If `O_NDELAY` is set, `read()` returns 0.
- If `O_NONBLOCK` is set, `read()` returns -1 and sets `errno` to `EAGAIN`.
- If `O_NDELAY` and `O_NONBLOCK` are clear, `read()` blocks until data become available.

When attempting to read a file associated with a socket or a stream that is not a pipe, a FIFO, or a terminal, and the file has no data currently available:

- If `O_NDELAY` or `O_NONBLOCK` is set, `read()` returns -1 and sets `errno` to `EAGAIN`.
- If `O_NDELAY` and `O_NONBLOCK` are clear, `read()` blocks until data becomes available.

The `read()` function reads data previously written to a file. If any portion of a regular file prior to the end-of-file has not been written, `read()` returns bytes with value 0. For example, `lseek(2)` allows the file offset to be set beyond the end of existing data in the file. If data is later written at this point, subsequent reads in the gap between the previous end of data and the newly written data will return bytes with value 0 until data is written into the gap.

For regular files, no data transfer will occur past the offset maximum established in the open file description associated with *fil*des.

Upon successful completion, where *nbyte* is greater than 0, `read()` will mark for update the `st_atime` field of the file, and return the number of bytes read. This number will never be greater than *nbyte*. The value returned may be less than *nbyte* if the number of bytes left in the file is less than *nbyte*, if the `read()` request was interrupted by a signal, or if the file is a pipe or FIFO or special file and has fewer than *nbyte* bytes immediately available for reading. For example, a `read()` from a file associated with a terminal may return one typed line of data.

If a `read()` is interrupted by a signal before it reads any data, it will return -1 with `errno` set to `EINTR`.

If a `read()` is interrupted by a signal after it has successfully read some data, it will return the number of bytes read.

A `read()` or `readv()` from a STREAMS (see `intro(2)`) file can read data in three different modes: byte-stream mode, message-nondiscard mode, and message-discard mode. The default is byte-stream mode. This can be changed using the `I_SRDOPT ioctl(2)` request, and can be tested with the `I_GRDOPT ioctl()`. In byte-stream mode, `read()` retrieves data from the STREAM until as many bytes as were requested are transferred, or until there is no more data to be retrieved. Byte-stream mode ignores message boundaries.

In STREAMS message-nondiscard mode, `read()` retrieves data until as many bytes as were requested are transferred, or until a message boundary is reached. If `read()` does not retrieve all the data in a message, the remaining data is left on the STREAM, and can be retrieved by the next `read()` call. Message-discard mode also retrieves data until as many bytes as were requested are transferred, or a message boundary is reached. However, unread data remaining in a message after the `read()` returns is discarded, and is not available for a subsequent `read()`, `readv()` or `getmsg(2)` call.

How `read()` handles zero-byte STREAMS messages is determined by the current read mode setting. In byte-stream mode, `read()` accepts data until it has read *nbyte* bytes, or until there is no more data to read, or until a zero-byte message block is encountered. The `read()` function then returns the number of bytes read, and places the zero-byte message back on the STREAM to be retrieved by the next `read()`, `readv()` or `getmsg(2)`. In message-nondiscard mode or message-discard mode, a zero-byte message returns 0 and the message is removed from the STREAM. When a zero-byte message is read as the first message on a STREAM, the message is removed from the STREAM and 0 is returned, regardless of the read mode.

A `read()` from a STREAMS file returns the data in the message at the front of the STREAM head read queue, regardless of the priority band of the message.

By default, STREAMS are in control-normal mode, in which a `read()` from a STREAMS file can only process messages that contain a data part but do not contain a control part. The `read()` fails if a message containing a control part is encountered at the STREAM head. This default action can be changed by placing the STREAM in either control-data mode or control-discard mode with the `I_SRDOPT ioctl()` command. In control-data mode, `read()` converts any control part to data and passes it to the application before passing any data part originally present in the same message. In control-discard mode, `read()` discards message control parts but returns to the process any data part in the message.

`readl()`, `preadl()`, and `readvl()` perform the same actions as `read()`, `pread()`, and `readv()`, respectively, and additionally return in *label_p* the CMW label of the data read. The label returned is determined according to these conditions:

- If the descriptor refers to a regular file or FIFO, the sensitivity label portion of *label_p* is set to the sensitivity label associated with the filesystem object.

readv(2)

	<p>In all other respects, the <code>readl()</code>, <code>preadl()</code>, and <code>readvl()</code> interfaces are analogous to the <code>read()</code>, <code>pread()</code>, and <code>readv()</code> interfaces.</p> <p>In the Solaris environment, <code>read()</code> normally allows a process to read the contents of directories on some local file systems. This functionality is not supported in the Trusted Solaris operating environment. If the file descriptor refers to a directory, <code>read()</code> will return <code>EISDIR</code>.</p> <p>The last access time is updated only when the calling process has both mandatory read and write access to the filesystem object. There is no privilege to override this restriction.</p> <p>In addition, <code>read()</code> and <code>readv()</code> will fail if the STREAM head had processed an asynchronous error before the call. In this case, the value of <code>errno</code> does not reflect the result of <code>read()</code> or <code>readv()</code> but reflects the prior error. If a hangup occurs on the STREAM being read, <code>read()</code> continues to operate normally until the STREAM head read queue is empty. Thereafter, it returns 0.</p>		
<code>readv()</code>	<p>The <code>readv()</code> function is equivalent to <code>read()</code>, but places the input data into the <i>iovcnt</i> buffers specified by the members of the <i>iov</i> array: <i>iov0</i>, <i>iov1</i>, ..., <i>iov[iovcnt-1]</i>. The <i>iovcnt</i> argument is valid if greater than 0 and less than or equal to <code>IOV_MAX</code>.</p> <p>The <code>iovec</code> structure contains the following members:</p> <pre>caddr_t iov_base; int iov_len;</pre> <p>Each <code>iovec</code> entry specifies the base address and length of an area in memory where data should be placed. The <code>readv()</code> function always fills an area completely before proceeding to the next.</p> <p>Upon successful completion, <code>readv()</code> marks for update the <code>st_atime</code> field of the file.</p>		
<code>pread()</code>	<p>The <code>pread()</code> function performs the same action as <code>read()</code>, except that it reads from a given position in the file without changing the file pointer. The first three arguments to <code>pread()</code> are the same as <code>read()</code> with the addition of a fourth argument <i>offset</i> for the desired position inside the file. <code>pread()</code> will read up to the maximum offset value that can be represented in an <code>off_t</code> for regular files. An attempt to perform a <code>pread()</code> on a file that is incapable of seeking results in an error.</p>		
RETURN VALUES	<p>Upon successful completion, <code>read()</code> and <code>readv()</code> return a non-negative integer indicating the number of bytes actually read. Otherwise, the functions return <code>-1</code> and set <code>errno</code> to indicate the error.</p>		
ERRORS	<p><code>read()</code>, <code>readl()</code>, <code>pread()</code>, <code>preadl()</code>, <code>readv()</code>, and <code>readvl()</code> fail if any of these conditions is true:</p> <table><tr><td><code>EAGAIN</code></td><td>Mandatory file/record locking was set, <code>O_NDELAY</code> or <code>O_NONBLOCK</code> was set, and there was a blocking record lock; total</td></tr></table>	<code>EAGAIN</code>	Mandatory file/record locking was set, <code>O_NDELAY</code> or <code>O_NONBLOCK</code> was set, and there was a blocking record lock; total
<code>EAGAIN</code>	Mandatory file/record locking was set, <code>O_NDELAY</code> or <code>O_NONBLOCK</code> was set, and there was a blocking record lock; total		

	amount of system memory available when reading using raw I/O is temporarily insufficient; no data is waiting to be read on a file associated with a tty device and <code>O_NONBLOCK</code> was set; or no message is waiting to be read on a stream and <code>O_NDELAY</code> or <code>O_NONBLOCK</code> was set.
EBADF	The <i>fil</i> des argument is not a valid file descriptor open for reading.
EBADMSG	Message waiting to be read on a stream is not a data message.
EDEADLK	The read was going to go to sleep and cause a deadlock to occur.
EFAULT	The <i>buf</i> argument points to an illegal address.
EINTR	A signal was caught during the read operation and no data was transferred.
EINVAL	An attempt was made to read from a stream linked to a multiplexor.
EIO	A physical I/O error has occurred, or the process is in a background process group and is attempting to read from its controlling terminal, and either the process is ignoring or blocking the SIGTTIN signal or the process group of the process is orphaned.
EISDIR	The <i>fil</i> des argument refers to a directory on a file system type that does not support read operations on directories.
ENOLCK	The system record lock table was full, so the <code>read()</code> or <code>readv()</code> could not go to sleep until the blocking record lock was removed.
ENOLINK	The <i>fil</i> des argument is on a remote machine and the link to that machine is no longer active.
ENXIO	The device associated with <i>fil</i> des is a block special or character special file and the value of the file pointer is out of range.
The <code>read()</code> and <code>readv()</code> functions will fail if:	
EOVERFLOW	The file is a regular file, <i>nbyte</i> is greater than 0, the starting position is before the end-of-file, and the starting position is greater than or equal to the offset maximum established in the open file description associated with <i>fil</i> des.
The <code>readv()</code> function may fail if:	
EFAULT	The <i>iov</i> argument points outside the allocated address space.
EINVAL	The <i>iovcnt</i> argument was less than or equal to 0, or greater than or equal to <code>IOV_MAX</code> . (See <code>intro(2)</code> for a definition of <code>IOV_MAX</code>).
EINVAL	The sum of the <i>iov_len</i> values in the <i>iov</i> array overflowed an int.
The <code>pread()</code> function will fail and the file pointer remain unchanged if:	

readv(2)

USAGE EFAULT *label_p* points to an illegal address.

ATTRIBUTES The `pread()` function has a transitional interface for 64-bit file offsets. See `lfs64(5)`.

See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	<code>read()</code> is Async-Signal-Safe

SUMMARY OF TRUSTED SOLARIS CHANGES

`readl()`, `preadl()`, and `readvl()` return in the buffer that was referenced by *label_p*, the CMW label associated with the data that was read.

The last access time is updated only when the calling process has both mandatory read and write access to the filesystem object. There is no privilege to override this restriction.

For conduits, a sensitivity label is associated with each byte of data.

Trusted Solaris 8 4/01 Reference Manual

`intro(2)`, `chmod(2)`, `creat(2)`, `fcntl(2)`, `open(2)`

`dup(2)`, `getmsg(2)`, `ioctl(2)`, `pipe(2)`, `attributes(5)`, `streamio(7I)`, `termio(7I)`

NAME	read, readl, pread, preadl, readv, readvl – read from a file
SYNOPSIS	<pre> #include <sys/types.h> #include <sys/uio.h> #include <unistd.h> ssize_t read(int fildes, void *buf, size_t nbyte); ssize_t pread(int fildes, void *buf, size_t nbyte, off_t offset); ssize_t readv(int fildes, struct iovec *iov, int iovcnt); #include <tsol/rdwrl.h> ssize_t readl(int fildes, void *buf, size_t nbyte, bclabel_t *label_p); ssize_t preadl(int fildes, void *buf, size_t nbyte, off_t offset, bclabel_t *label_p); ssize_t readvl(int fildes, struct iovec *iov, int iovcnt, bclabel_t *label_p); </pre>
DESCRIPTION	<p>The <code>read()</code> function attempts to read <i>nbyte</i> bytes from the file associated with the open file descriptor, <i>fildes</i>, into the buffer pointed to by <i>buf</i>.</p> <p>If <i>nbyte</i> is 0, <code>read()</code> will return 0 and have no other results.</p> <p>On files that support seeking (for example, a regular file), the <code>read()</code> starts at a position in the file given by the file offset associated with <i>fildes</i>. The file offset is incremented by the number of bytes actually read.</p> <p>Files that do not support seeking (for example, terminals) always read from the current position. The value of a file offset associated with such a file is undefined.</p> <p>If <i>fildes</i> refers to a socket, <code>read()</code> is equivalent to <code>recv(3SOCKET)</code> with no flags set.</p> <p>No data transfer will occur past the current end-of-file. If the starting position is at or after the end-of-file, 0 will be returned. If the file refers to a device special file, the result of subsequent <code>read()</code> requests is implementation-dependent.</p> <p>If the value of <i>nbyte</i> is greater than <code>SSIZE_MAX</code>, the result is implementation-dependent.</p> <p>When attempting to read from a regular file with mandatory file/record locking set (see <code>chmod(2)</code>), and there is a write lock owned by another process on the segment of the file to be read:</p> <ul style="list-style-type: none"> ■ If <code>O_NDELAY</code> or <code>O_NONBLOCK</code> is set, <code>read()</code> returns -1 and sets <code>errno</code> to <code>EAGAIN</code>. ■ If <code>O_NDELAY</code> and <code>O_NONBLOCK</code> are clear, <code>read()</code> sleeps until the blocking record lock is removed. <p>When attempting to read from an empty pipe (or FIFO):</p>

readv(2)

- If no process has the pipe open for writing, `read()` returns 0 to indicate end-of-file.
- If some process has the pipe open for writing and `O_NDELAY` is set, `read()` returns 0.
- If some process has the pipe open for writing and `O_NONBLOCK` is set, `read()` returns -1 and sets `errno` to `EAGAIN`.
- If `O_NDELAY` and `O_NONBLOCK` are clear, `read()` blocks until data is written to the pipe or the pipe is closed by all processes that had opened the pipe for writing.

When attempting to read a file associated with a terminal that has no data currently available:

- If `O_NDELAY` is set, `read()` returns 0.
- If `O_NONBLOCK` is set, `read()` returns -1 and sets `errno` to `EAGAIN`.
- If `O_NDELAY` and `O_NONBLOCK` are clear, `read()` blocks until data become available.

When attempting to read a file associated with a socket or a stream that is not a pipe, a FIFO, or a terminal, and the file has no data currently available:

- If `O_NDELAY` or `O_NONBLOCK` is set, `read()` returns -1 and sets `errno` to `EAGAIN`.
- If `O_NDELAY` and `O_NONBLOCK` are clear, `read()` blocks until data becomes available.

The `read()` function reads data previously written to a file. If any portion of a regular file prior to the end-of-file has not been written, `read()` returns bytes with value 0. For example, `lseek(2)` allows the file offset to be set beyond the end of existing data in the file. If data is later written at this point, subsequent reads in the gap between the previous end of data and the newly written data will return bytes with value 0 until data is written into the gap.

For regular files, no data transfer will occur past the offset maximum established in the open file description associated with *fil*des.

Upon successful completion, where *nbyte* is greater than 0, `read()` will mark for update the `st_atime` field of the file, and return the number of bytes read. This number will never be greater than *nbyte*. The value returned may be less than *nbyte* if the number of bytes left in the file is less than *nbyte*, if the `read()` request was interrupted by a signal, or if the file is a pipe or FIFO or special file and has fewer than *nbyte* bytes immediately available for reading. For example, a `read()` from a file associated with a terminal may return one typed line of data.

If a `read()` is interrupted by a signal before it reads any data, it will return -1 with `errno` set to `EINTR`.

If a `read()` is interrupted by a signal after it has successfully read some data, it will return the number of bytes read.

A `read()` or `readv()` from a STREAMS (see `intro(2)`) file can read data in three different modes: byte-stream mode, message-nondiscard mode, and message-discard mode. The default is byte-stream mode. This can be changed using the `I_SRDOPT` `ioctl(2)` request, and can be tested with the `I_GRDOPT` `ioctl()`. In byte-stream mode, `read()` retrieves data from the STREAM until as many bytes as were requested are transferred, or until there is no more data to be retrieved. Byte-stream mode ignores message boundaries.

In STREAMS message-nondiscard mode, `read()` retrieves data until as many bytes as were requested are transferred, or until a message boundary is reached. If `read()` does not retrieve all the data in a message, the remaining data is left on the STREAM, and can be retrieved by the next `read()` call. Message-discard mode also retrieves data until as many bytes as were requested are transferred, or a message boundary is reached. However, unread data remaining in a message after the `read()` returns is discarded, and is not available for a subsequent `read()`, `readv()` or `getmsg(2)` call.

How `read()` handles zero-byte STREAMS messages is determined by the current read mode setting. In byte-stream mode, `read()` accepts data until it has read *nbyte* bytes, or until there is no more data to read, or until a zero-byte message block is encountered. The `read()` function then returns the number of bytes read, and places the zero-byte message back on the STREAM to be retrieved by the next `read()`, `readv()` or `getmsg(2)`. In message-nondiscard mode or message-discard mode, a zero-byte message returns 0 and the message is removed from the STREAM. When a zero-byte message is read as the first message on a STREAM, the message is removed from the STREAM and 0 is returned, regardless of the read mode.

A `read()` from a STREAMS file returns the data in the message at the front of the STREAM head read queue, regardless of the priority band of the message.

By default, STREAMS are in control-normal mode, in which a `read()` from a STREAMS file can only process messages that contain a data part but do not contain a control part. The `read()` fails if a message containing a control part is encountered at the STREAM head. This default action can be changed by placing the STREAM in either control-data mode or control-discard mode with the `I_SRDOPT` `ioctl()` command. In control-data mode, `read()` converts any control part to data and passes it to the application before passing any data part originally present in the same message. In control-discard mode, `read()` discards message control parts but returns to the process any data part in the message.

`readl()`, `preadl()`, and `readvl()` perform the same actions as `read()`, `pread()`, and `readv()`, respectively, and additionally return in *label_p* the CMW label of the data read. The label returned is determined according to these conditions:

- If the descriptor refers to a regular file or FIFO, the sensitivity label portion of *label_p* is set to the sensitivity label associated with the filesystem object.

readv(2)

	<p>In all other respects, the <code>readl()</code>, <code>preadl()</code>, and <code>readvl()</code> interfaces are analogous to the <code>read()</code>, <code>pread()</code>, and <code>readv()</code> interfaces.</p> <p>In the Solaris environment, <code>read()</code> normally allows a process to read the contents of directories on some local file systems. This functionality is not supported in the Trusted Solaris operating environment. If the file descriptor refers to a directory, <code>read()</code> will return <code>EISDIR</code>.</p> <p>The last access time is updated only when the calling process has both mandatory read and write access to the filesystem object. There is no privilege to override this restriction.</p> <p>In addition, <code>read()</code> and <code>readv()</code> will fail if the STREAM head had processed an asynchronous error before the call. In this case, the value of <code>errno</code> does not reflect the result of <code>read()</code> or <code>readv()</code> but reflects the prior error. If a hangup occurs on the STREAM being read, <code>read()</code> continues to operate normally until the STREAM head read queue is empty. Thereafter, it returns 0.</p>		
<code>readv()</code>	<p>The <code>readv()</code> function is equivalent to <code>read()</code>, but places the input data into the <i>iovcnt</i> buffers specified by the members of the <i>iov</i> array: <i>iov0</i>, <i>iov1</i>, ..., <i>iov[iovcnt-1]</i>. The <i>iovcnt</i> argument is valid if greater than 0 and less than or equal to <code>IOV_MAX</code>.</p> <p>The <code>iovec</code> structure contains the following members:</p> <pre>caddr_t iov_base; int iov_len;</pre> <p>Each <code>iovec</code> entry specifies the base address and length of an area in memory where data should be placed. The <code>readv()</code> function always fills an area completely before proceeding to the next.</p> <p>Upon successful completion, <code>readv()</code> marks for update the <code>st_atime</code> field of the file.</p>		
<code>pread()</code>	<p>The <code>pread()</code> function performs the same action as <code>read()</code>, except that it reads from a given position in the file without changing the file pointer. The first three arguments to <code>pread()</code> are the same as <code>read()</code> with the addition of a fourth argument <i>offset</i> for the desired position inside the file. <code>pread()</code> will read up to the maximum offset value that can be represented in an <code>off_t</code> for regular files. An attempt to perform a <code>pread()</code> on a file that is incapable of seeking results in an error.</p>		
RETURN VALUES	<p>Upon successful completion, <code>read()</code> and <code>readv()</code> return a non-negative integer indicating the number of bytes actually read. Otherwise, the functions return <code>-1</code> and set <code>errno</code> to indicate the error.</p>		
ERRORS	<p><code>read()</code>, <code>readl()</code>, <code>pread()</code>, <code>preadl()</code>, <code>readv()</code>, and <code>readvl()</code> fail if any of these conditions is true:</p> <table><tr><td><code>EAGAIN</code></td><td>Mandatory file/record locking was set, <code>O_NDELAY</code> or <code>O_NONBLOCK</code> was set, and there was a blocking record lock; total</td></tr></table>	<code>EAGAIN</code>	Mandatory file/record locking was set, <code>O_NDELAY</code> or <code>O_NONBLOCK</code> was set, and there was a blocking record lock; total
<code>EAGAIN</code>	Mandatory file/record locking was set, <code>O_NDELAY</code> or <code>O_NONBLOCK</code> was set, and there was a blocking record lock; total		

	amount of system memory available when reading using raw I/O is temporarily insufficient; no data is waiting to be read on a file associated with a tty device and <code>O_NONBLOCK</code> was set; or no message is waiting to be read on a stream and <code>O_NDELAY</code> or <code>O_NONBLOCK</code> was set.
EBADF	The <i>fdes</i> argument is not a valid file descriptor open for reading.
EBADMSG	Message waiting to be read on a stream is not a data message.
EDEADLK	The read was going to go to sleep and cause a deadlock to occur.
EFAULT	The <i>buf</i> argument points to an illegal address.
EINTR	A signal was caught during the read operation and no data was transferred.
EINVAL	An attempt was made to read from a stream linked to a multiplexor.
EIO	A physical I/O error has occurred, or the process is in a background process group and is attempting to read from its controlling terminal, and either the process is ignoring or blocking the <code>SIGTTIN</code> signal or the process group of the process is orphaned.
EISDIR	The <i>fdes</i> argument refers to a directory on a file system type that does not support read operations on directories.
ENOLCK	The system record lock table was full, so the <code>read()</code> or <code>readv()</code> could not go to sleep until the blocking record lock was removed.
ENOLINK	The <i>fdes</i> argument is on a remote machine and the link to that machine is no longer active.
ENXIO	The device associated with <i>fdes</i> is a block special or character special file and the value of the file pointer is out of range.
The <code>read()</code> and <code>readv()</code> functions will fail if:	
EOVERFLOW	The file is a regular file, <i>nbyte</i> is greater than 0, the starting position is before the end-of-file, and the starting position is greater than or equal to the offset maximum established in the open file description associated with <i>fdes</i> .
The <code>readv()</code> function may fail if:	
EFAULT	The <i>iov</i> argument points outside the allocated address space.
EINVAL	The <i>iovcnt</i> argument was less than or equal to 0, or greater than or equal to <code>IOV_MAX</code> . (See <code>intro(2)</code> for a definition of <code>IOV_MAX</code>).
EINVAL	The sum of the <code>iov_len</code> values in the <i>iov</i> array overflowed an int.
The <code>pread()</code> function will fail and the file pointer remain unchanged if:	

readvl(2)

USAGE EFAULT *label_p* points to an illegal address.

ATTRIBUTES The `pread()` function has a transitional interface for 64-bit file offsets. See `lf64(5)`.

See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	<code>read()</code> is Async-Signal-Safe

SUMMARY OF TRUSTED SOLARIS CHANGES

`readl()`, `preadl()`, and `readvl()` return in the buffer that was referenced by *label_p*, the CMW label associated with the data that was read.

The last access time is updated only when the calling process has both mandatory read and write access to the filesystem object. There is no privilege to override this restriction.

For conduits, a sensitivity label is associated with each byte of data.

Trusted Solaris 8 4/01 Reference Manual

`intro(2)`, `chmod(2)`, `creat(2)`, `fcntl(2)`, `open(2)`

`dup(2)`, `getmsg(2)`, `ioctl(2)`, `pipe(2)`, `attributes(5)`, `streamio(7I)`, `termio(7I)`

NAME	rename – change the name of a file
SYNOPSIS	<pre>#include <stdio.h> int rename(const char *old, const char *new);</pre>
DESCRIPTION	<p>The <code>rename()</code> function changes the name of a file. The <i>old</i> argument points to the pathname of the file to be renamed. The <i>new</i> argument points to the new pathname of the file.</p> <p>If <i>old</i> and <i>new</i> both refer to the same existing file, the <code>rename()</code> function returns successfully and performs no other action.</p> <p>If <i>old</i> points to the pathname of a file that is not a directory, <i>new</i> must not point to the pathname of a directory. If the link named by <i>new</i> exists, it will be removed and <i>old</i> will be renamed to <i>new</i>. In this case, a link named <i>new</i> must remain visible to other processes throughout the renaming operation and will refer to either the file referred to by <i>new</i> or the file referred to as <i>old</i> before the operation began.</p> <p>If <i>old</i> points to the pathname of a directory, <i>new</i> must not point to the pathname of a file that is not a directory. If the directory named by <i>new</i> exists, it will be removed and <i>old</i> will be renamed to <i>new</i>. In this case, a link named <i>new</i> will exist throughout the renaming operation and will refer to either the file referred to by <i>new</i> or the file referred to as <i>old</i> before the operation began. Thus, if <i>new</i> names an existing directory, it must be an empty directory.</p> <p>The <i>new</i> pathname must not contain a path prefix that names <i>old</i>. Write access permission is required for both the directory containing <i>old</i> and the directory containing <i>new</i>. If <i>old</i> points to the pathname of a directory, write access permission is required for the directory named by <i>old</i>, and, if it exists, the directory named by <i>new</i>.</p> <p>If the directory containing <i>old</i> has the sticky bit set, at least one of the following conditions listed below must be true:</p> <ul style="list-style-type: none"> ■ The user must own <i>old</i> ■ The user must own the directory containing <i>old</i> ■ <i>old</i> must be writable by the user ■ The user must be a privileged user <p>If <i>new</i> exists, and the directory containing <i>new</i> is writable and has the sticky bit set, at least one of the following conditions must be true:</p> <ul style="list-style-type: none"> ■ the user must own <i>new</i> ■ the user must own the directory containing <i>new</i> ■ <i>new</i> must be writable by the user ■ the user must be a privileged user <p>If the link named by <i>new</i> exists, the file's link count becomes zero when it is removed, and no process has the file open, then the space occupied by the file will be freed and the file will no longer be accessible. If one or more processes have the file open when</p>

rename(2)

the last link is removed, the link will be removed before `rename()` returns, but the removal of the file contents will be postponed until all references to the file have been closed.

Upon successful completion, the `rename()` function will mark for update the `st_ctime` and `st_mtime` fields of the parent directory of each file.

A single-level directory cannot be renamed (single-level directories are always contained in multilevel directories). A multilevel directory cannot be the new containing directory. There is no privilege to bypass these restrictions.

RETURN VALUES

`rename()` returns:

- 0 On success.
- 1 On failure, and sets `errno` to indicate the error.

ERRORS

The `rename()` function will fail if:

EACCES	A component of either path prefix denies search permission; one of the directories containing <i>old</i> and <i>new</i> denies write permissions; or write permission is denied by a directory pointed to by <i>old</i> or <i>new</i> . To bypass ownership restrictions, the calling process may assert one or more of these privileges: PRIV_FILE_DAC_SEARCH, PRIV_FILE_MAC_SEARCH, PRIV_FILE_MAC_WRITE, PRIV_FILE_DAC_WRITE, and PRIV_FILE_OWNER.
EBUSY	The <i>new</i> argument is a directory and the mount point for a mounted file system.
EDQUOT	The directory where the new name entry is being placed cannot be extended because the user's quota of disk blocks on that file system has been exhausted.
EEXIST	The link named by <i>new</i> is a directory containing entries other than <code>'.'</code> (the directory itself) and <code>'..'</code> (the parent directory).
EINVAL	The <i>new</i> argument directory pathname contains a path prefix that names the <i>old</i> directory.
EISDIR	The <i>new</i> argument points to a directory but <i>old</i> points to a file that is not a directory.
ELOOP	Too many symbolic links were encountered in translating the pathname.
ENAMETOOLONG	The length of <i>old</i> or <i>new</i> exceeds <code>PATH_MAX</code> , or a pathname component is longer than <code>NAME_MAX</code> while <code>_POSIX_NO_TRUNC</code> is in effect.

		rename(2)				
	EMLINK	The file named by <i>old</i> is a directory, and the link count of the parent directory of <i>new</i> would exceed LINK_MAX.				
	ENOENT	The link named by <i>old</i> does not exist, or either <i>old</i> or <i>new</i> points to an empty string.				
	ENOSPC	The directory that would contain <i>new</i> cannot be extended.				
	ENOTDIR	A component of either path prefix is not a directory, or <i>old</i> names a directory and <i>new</i> names a nondirectory file.				
	EROFS	The requested operation requires writing in a directory on a read-only file system.				
	EXDEV	The links named by <i>old</i> and <i>new</i> are on different file systems.				
	EIO	An I/O error occurred while making or updating a directory entry.				
ATTRIBUTES	See attributes(5) for descriptions of the following attributes:					
	<table><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr><tr><td>MT-Level</td><td>Async-Signal-Safe</td></tr></table>		ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	Async-Signal-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE					
MT-Level	Async-Signal-Safe					
SUMMARY OF TRUSTED SOLARIS CHANGES	Appropriate privilege is required to override access checks.					
	A single-level directory cannot be renamed. A multilevel directory cannot be the new containing directory. There is no privilege to bypass these restrictions.					
Trusted Solaris 8 4/01 Reference Manual	chmod(2), link(2), unlink(2)					
OS-1000 Reference Manual	attributes(5)					
WARNINGS	The system can deadlock if there is a loop in the file system graph. Such a loop can occur if there is an entry in directory <i>a</i> , <i>a/name1</i> , that is a hard link to directory <i>b</i> , and an entry in directory <i>b</i> , <i>b/name2</i> , that is a hard link to directory <i>a</i> . When such a loop exists and two separate processes attempt to rename <i>a/name1</i> to <i>b/name2</i> and <i>b/name2</i> to <i>a/name1</i> , the system may deadlock attempting to lock both directories for modification. The solution is to use symbolic links instead of hard links for directories.					

rmkdir(2)

NAME	rmkdir – remove a directory												
SYNOPSIS	<pre>#include <unistd.h> int rmkdir(const char *path);</pre>												
DESCRIPTION	<p>The <code>rmkdir()</code> function removes the directory named by the path name pointed to by <i>path</i>. The directory must not have any entries other than “.” and “..”.</p> <p>If the directory’s link count becomes zero and no process has the directory open, the space occupied by the directory is freed and the directory is no longer accessible. If one or more processes have the directory open when the last link is removed, the “.” and “..” entries, if present, are removed before <code>rmkdir()</code> returns and no new entries may be created in the directory, but the directory is not removed until all references to the directory have been closed.</p> <p>Upon successful completion <code>rmkdir()</code> marks for update the <code>st_ctime</code> and <code>st_mtime</code> fields of the parent directory. A multilevel directory can be removed only when all its contained single-level directories are empty.</p>												
RETURN VALUES	<p><code>rmkdir()</code> returns:</p> <p>0 On success.</p> <p>–1 On failure, and sets <code>errno</code> to indicate the error.</p>												
ERRORS	<p>The <code>rmkdir()</code> function will fail if:</p> <table> <tr> <td>EACCES</td><td>Search permission is denied for a component of the path prefix. To override this restriction, the calling process must assert one or both of these privileges: <code>PRIV_FILE_DAC_SEARCH</code> and <code>PRIV_FILE_MAC_SEARCH</code>.</td></tr> <tr> <td>EACCES</td><td>Write permission is denied on the directory containing the directory to be removed. To bypass discretionary or mandatory write restrictions, the calling process must assert one or both of these privileges: <code>PRIV_FILE_DAC_WRITE</code> and <code>PRIV_FILE_MAC_WRITE</code>.</td></tr> <tr> <td>EACCES</td><td>If the containing directory has the <code>S_ISVTX</code> variable set, the calling process must either be the owner of the containing directory or the directory to be deleted, or must have write access to the directory to be deleted. To override this restriction, the calling process may assert one or more of these privileges: <code>PRIV_FILE_MAC_WRITE</code>, <code>PRIV_FILE_DAC_WRITE</code>, and <code>PRIV_FILE_OWNER</code>.</td></tr> <tr> <td>EBUSY</td><td>The directory to be removed is the mount point for a mounted file system.</td></tr> <tr> <td>EEXIST</td><td>The directory contains entries other than those for “.” and “..”.</td></tr> <tr> <td>EFAULT</td><td>The <i>path</i> argument points to an illegal address.</td></tr> </table>	EACCES	Search permission is denied for a component of the path prefix. To override this restriction, the calling process must assert one or both of these privileges: <code>PRIV_FILE_DAC_SEARCH</code> and <code>PRIV_FILE_MAC_SEARCH</code> .	EACCES	Write permission is denied on the directory containing the directory to be removed. To bypass discretionary or mandatory write restrictions, the calling process must assert one or both of these privileges: <code>PRIV_FILE_DAC_WRITE</code> and <code>PRIV_FILE_MAC_WRITE</code> .	EACCES	If the containing directory has the <code>S_ISVTX</code> variable set, the calling process must either be the owner of the containing directory or the directory to be deleted, or must have write access to the directory to be deleted. To override this restriction, the calling process may assert one or more of these privileges: <code>PRIV_FILE_MAC_WRITE</code> , <code>PRIV_FILE_DAC_WRITE</code> , and <code>PRIV_FILE_OWNER</code> .	EBUSY	The directory to be removed is the mount point for a mounted file system.	EEXIST	The directory contains entries other than those for “.” and “..”.	EFAULT	The <i>path</i> argument points to an illegal address.
EACCES	Search permission is denied for a component of the path prefix. To override this restriction, the calling process must assert one or both of these privileges: <code>PRIV_FILE_DAC_SEARCH</code> and <code>PRIV_FILE_MAC_SEARCH</code> .												
EACCES	Write permission is denied on the directory containing the directory to be removed. To bypass discretionary or mandatory write restrictions, the calling process must assert one or both of these privileges: <code>PRIV_FILE_DAC_WRITE</code> and <code>PRIV_FILE_MAC_WRITE</code> .												
EACCES	If the containing directory has the <code>S_ISVTX</code> variable set, the calling process must either be the owner of the containing directory or the directory to be deleted, or must have write access to the directory to be deleted. To override this restriction, the calling process may assert one or more of these privileges: <code>PRIV_FILE_MAC_WRITE</code> , <code>PRIV_FILE_DAC_WRITE</code> , and <code>PRIV_FILE_OWNER</code> .												
EBUSY	The directory to be removed is the mount point for a mounted file system.												
EEXIST	The directory contains entries other than those for “.” and “..”.												
EFAULT	The <i>path</i> argument points to an illegal address.												

rmdir(2)

EINVAL	The directory to be removed is the current directory, or the final component of <i>path</i> is “.”.
EIO	An I/O error occurred while accessing the file system.
ELOOP	Too many symbolic links were encountered in translating <i>path</i> .
ENAMETOOLONG	The length of the <i>path</i> argument exceeds PATH_MAX, or the length of a <i>path</i> component exceeds NAME_MAX while _POSIX_NO_TRUNC is in effect.
ENOENT	The named directory does not exist or is the null pathname.
ENOLINK	The <i>path</i> argument points to a remote machine, and the connection to that machine is no longer active.
ENOTDIR	A component of the path prefix is not a directory.
EROFS	The directory entry to be removed is part of a read-only file system.

ATTRIBUTES

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Async-Signal-Safe

SUMMARY OF TRUSTED SOLARIS CHANGES

Appropriate privilege is required to override access checks.

A multilevel directory can be removed only when all its contained single-level directories are empty.

Trusted Solaris 8
4/01 Reference
Manual
Solaris 8
Reference Manual

mkdir(1), rm(1), mkdir(2)

attributes(5)

secconf(2)

NAME	secconf – get security configuration information																								
SYNOPSIS	<pre>cc [flags...] file... -ltsol #include <tsol/secconf.h> long secconf(int name);</pre>																								
DESCRIPTION	<p>The <code>secconf()</code> system call provides a method for an application to determine the current value of a configurable security system limit or option.</p> <p>The <i>name</i> argument represents the system variable to be queried.</p> <table> <tr> <td>int name</td><td>_TSOL_CLEAN_WINDOWS</td></tr> <tr> <td>Variable name</td><td>tsol_clean_windows</td></tr> <tr> <td></td><td>Force cleaning of unused register windows before return from system call (SPARC architecture only).</td></tr> <tr> <td>int name</td><td>_TSOL_FLUSH_BUFFERS</td></tr> <tr> <td>Variable name</td><td>tsol_flush_buffers</td></tr> <tr> <td></td><td>Force flushing of file data blocks before inode updates.</td></tr> <tr> <td>int name</td><td>_TSOL_HIDE_UPGRADED_NAMES</td></tr> <tr> <td>Variable name</td><td>tsol_hide_upgraded_names</td></tr> <tr> <td></td><td>Hide upgrade directory entries.</td></tr> <tr> <td>int name</td><td>_TSOL_PRIVS_DEBUG</td></tr> <tr> <td>Variable name</td><td>tsol_privs_debug</td></tr> <tr> <td></td><td>Enables privilege debugging mode.</td></tr> </table>	int name	_TSOL_CLEAN_WINDOWS	Variable name	tsol_clean_windows		Force cleaning of unused register windows before return from system call (SPARC architecture only).	int name	_TSOL_FLUSH_BUFFERS	Variable name	tsol_flush_buffers		Force flushing of file data blocks before inode updates.	int name	_TSOL_HIDE_UPGRADED_NAMES	Variable name	tsol_hide_upgraded_names		Hide upgrade directory entries.	int name	_TSOL_PRIVS_DEBUG	Variable name	tsol_privs_debug		Enables privilege debugging mode.
int name	_TSOL_CLEAN_WINDOWS																								
Variable name	tsol_clean_windows																								
	Force cleaning of unused register windows before return from system call (SPARC architecture only).																								
int name	_TSOL_FLUSH_BUFFERS																								
Variable name	tsol_flush_buffers																								
	Force flushing of file data blocks before inode updates.																								
int name	_TSOL_HIDE_UPGRADED_NAMES																								
Variable name	tsol_hide_upgraded_names																								
	Hide upgrade directory entries.																								
int name	_TSOL_PRIVS_DEBUG																								
Variable name	tsol_privs_debug																								
	Enables privilege debugging mode.																								
RETURN VALUES	<p><code>secconf()</code> returns:</p> <table> <tr> <td>0</td><td>On success.</td></tr> <tr> <td>-1</td><td>When <i>name</i> is an invalid value. Also sets <code>errno</code> to indicate the error.</td></tr> <tr> <td></td><td>When <i>name</i> is not defined on the system. The value of <code>errno</code> will not be set.</td></tr> </table>	0	On success.	-1	When <i>name</i> is an invalid value. Also sets <code>errno</code> to indicate the error.		When <i>name</i> is not defined on the system. The value of <code>errno</code> will not be set.																		
0	On success.																								
-1	When <i>name</i> is an invalid value. Also sets <code>errno</code> to indicate the error.																								
	When <i>name</i> is not defined on the system. The value of <code>errno</code> will not be set.																								
ERRORS	<p>The function will return the following errors:</p> <table> <tr> <td>EINVAL</td><td>The parameter <i>name</i> is unknown.</td></tr> </table>	EINVAL	The parameter <i>name</i> is unknown.																						
EINVAL	The parameter <i>name</i> is unknown.																								
FILES	<table> <tr> <td>system(4)</td><td>System configuration information file.</td></tr> <tr> <td>pathconf(2)</td><td></td></tr> <tr> <td>sysconf(3C)</td><td></td></tr> </table>	system(4)	System configuration information file.	pathconf(2)		sysconf(3C)																			
system(4)	System configuration information file.																								
pathconf(2)																									
sysconf(3C)																									

NAME	semctl – semaphore control operations								
SYNOPSIS	<pre>#include <sys/types.h> #include <sys/ipc.h> #include <sys/sem.h> int semctl(int <i>semid</i>, int <i>semnum</i>, int <i>cmd</i>, ...);</pre>								
DESCRIPTION	<p>The <code>semctl()</code> function provides a variety of semaphore control operations as specified by <i>cmd</i>. The fourth argument is optional, depending upon the operation requested. If required, it is of type union <code>semun</code>, which must be explicitly declared by the application program.</p> <pre>union semun { int val; struct semid_ds *buf; ushort_t *array; } arg ;</pre> <p>The permission required for a semaphore operation is given as <i>{token}</i>, where <i>token</i> is the type of permission needed. The types of permission are interpreted as follows:</p> <pre>00400 READ by user 00200 ALTER by user 00040 READ by group 00020 ALTER by group 00004 READ by others 00002 ALTER by others</pre> <p>The commands described hereafter as [READ] operations all require that the calling process have discretionary read access to the data structure referenced by <i>semid</i>, or that the effective privilege set of the process include <code>PRIV_IPC_DAC_READ</code>. Likewise, the commands described as [ALTER] operations all require that the calling process have discretionary write access to the data structure referenced by <i>semid</i>, or that the effective privilege set of the process include <code>PRIV_IPC_DAC_WRITE</code>.</p> <p>If the sensitivity label of the object does not match the sensitivity label of the calling process, then the process must have these privileges asserted: <code>PRIV_IPC_MAC_READ</code> for [READ] operations; <code>PRIV_IPC_MAC_WRITE</code> for [ALTER] operations.</p> <p>See the Semaphore Operation Permissions subsection of the DEFINITIONS section of <code>intro(2)</code> for more information. The following semaphore operations as specified by <i>cmd</i> are executed with respect to the semaphore specified by <i>semid</i> and <i>semnum</i>.</p> <table><tr><td>GETVAL</td><td>Return the value of <code>semval</code> (see <code>intro(2)</code>). {READ}</td></tr><tr><td>SETVAL</td><td>Set the value of <code>semval</code> to <i>arg.val</i>. {ALTER} When this command is successfully executed, the <code>semadj</code> value corresponding to the specified semaphore in all processes is cleared.</td></tr><tr><td>GETPID</td><td>Return the value of <code>(int) sempid</code>. {READ}</td></tr><tr><td>GETNCNT</td><td>Return the value of <code>semncnt</code>. {READ}</td></tr></table>	GETVAL	Return the value of <code>semval</code> (see <code>intro(2)</code>). {READ}	SETVAL	Set the value of <code>semval</code> to <i>arg.val</i> . {ALTER} When this command is successfully executed, the <code>semadj</code> value corresponding to the specified semaphore in all processes is cleared.	GETPID	Return the value of <code>(int) sempid</code> . {READ}	GETNCNT	Return the value of <code>semncnt</code> . {READ}
GETVAL	Return the value of <code>semval</code> (see <code>intro(2)</code>). {READ}								
SETVAL	Set the value of <code>semval</code> to <i>arg.val</i> . {ALTER} When this command is successfully executed, the <code>semadj</code> value corresponding to the specified semaphore in all processes is cleared.								
GETPID	Return the value of <code>(int) sempid</code> . {READ}								
GETNCNT	Return the value of <code>semncnt</code> . {READ}								

semctl(2)

	GETZCNT	Return the value of <code>semzcnt</code> . {READ}
	The following operations return and set, respectively, every <code>semval</code> in the set of semaphores.	
	GETALL	Place <code>semvals</code> into array pointed to by <code>arg.array</code> . {READ}
	SETALL	Set <code>semvals</code> according to the array pointed to by <code>arg.array</code> . {ALTER}. When this cmd is successfully executed, the <code>semadj</code> values corresponding to each specified semaphore in all processes are cleared.
	The following operations are also available.	
	IPC_STAT	Place the current value of each member of the data structure associated with <code>semid</code> into the structure pointed to by <code>arg.buf</code> . The contents of this structure are defined in <code>intro(2)</code> . {READ}
	IPC_SET	<p>Set the value of the following members of the data structure associated with <code>semid</code> to the corresponding value found in the structure pointed to by <code>arg.buf</code>:</p> <pre>sem_perm.uid sem_perm.gid sem_perm.mode /* access permission bits only */</pre> <p>This command can be executed only by a process that either has an effective user ID equal to <code>sem_perm.cuid</code> or <code>sem_perm.uid</code> in the data structure associated with <code>semid</code>, or has the <code>PRIV_IPC_OWNER</code> privilege in its set of effective privileges. In addition, the process must either have mandatory write access to the Semaphore set or have asserted the <code>PRIV_IPC_MAC_WRITE</code> privilege.</p>
	IPC_RMID	Remove from the system the semaphore identifier specified by <code>semid</code> and destroy the set of semaphores and data structure associated with that identifier. This command can be executed only by a process that either has an effective user ID equal to <code>sem_perm.cuid</code> or <code>sem_perm.uid</code> in the data structure associated with <code>semid</code> , or has the <code>PRIV_IPC_OWNER</code> privilege asserted. In addition, the process must also have mandatory write access to the Semaphore set or have asserted the <code>PRIV_IPC_MAC_WRITE</code> privilege.
RETURN VALUES	Upon successful completion, the value returned depends on <code>cmd</code> as follows:	
	GETVAL	the value of <code>semval</code>
	GETPID	the value of <code>(int) sempid</code>
	GETNCNT	the value of <code>semncnt</code>
	GETZCNT	the value of <code>semzcnt</code>

All other successful completions return 0; otherwise, -1 is returned and errno is set to indicate the error.

ERRORS

The semctl () function will fail if:

- EACCES Operation permission is denied to the calling process (see intro(2)), and the process lacks the appropriate privilege override(s) in its set of effective privileges.
- EINVAL The *semid* argument is not a valid semaphore identifier; the *semnum* argument is less than 0 or greater than *sem_nsems* -1; or the *cmd* argument is not a valid command or is IPC_SET and *sem_perm.uid* or *sem_perm.gid* is not valid.
- EPERM *cmd* is equal to IPC_RMID or IPC_SET and the effective user of the calling process is not equal to the value of *sem_perm.cuid* or *sem_perm.uid* in the data structure associated with *semid*, and the appropriate privilege is not asserted.
- EOVERFLOW The *cmd* argument is IPC_STAT and *uid* or *gid* is too large to be stored in the structure pointed to by *arg.buf*.
- ERANGE The *cmd* argument is SETVAL or SETALL and the value to which *semval* is to be set is greater than the system imposed maximum.

SUMMARY OF
TRUSTED
SOLARIS
4/01 Reference
Manual

Appropriate privilege is required to override access checks.

ipcs(1), intro(2), semget(2), semop(2)

semget(2)

NAME	semget, semgetl – get set of semaphores
SYNOPSIS	<pre>#include <sys/types.h> #include <sys/ipc.h> #include <sys/sem.h> int semget(key_t key, int nsems, int semflg); cc [flags...] file ... -ltsol [library...] #include <sys/tsol/ipcl.h> int semgetl(key_t key, int nsems, int semflg, const bslabel_t *slabel);</pre>
DESCRIPTION	<p>A semaphore structure is identified by a unique combination of key and sensitivity label. This qualification of keys by sensitivity labels allows applications that use semaphore structures to be run at multiple process sensitivity labels without inadvertently sharing data.</p> <p>The <code>semget()</code> function returns the semaphore identifier associated with the union of key and the sensitivity label of the calling process. <code>semgetl()</code> returns the semaphore-structure identifier associated with the union of <i>key</i> and <i>slabel</i>. If the value of <i>slabel</i> does not match the sensitivity label of the calling process, then the effective privilege set of the process must include both <code>PRIV_IPC_MAC_READ</code> and <code>PRIV_IPC_MAC_WRITE</code>.</p> <p>If discretionary read/write access as specified by the low-order 9 bits of <i>semflg</i> is denied to the calling process, <code>semget()</code> and <code>semgetl()</code> require one or both of these privileges: <code>PRIV_IPC_DAC_READ</code> and <code>PRIV_IPC_DAC_WRITE</code>.</p> <p>A semaphore identifier and associated data structure and set containing <i>nsems</i> semaphores (see <code>intro(2)</code>) are created for <i>key</i> if one of the following is true:</p> <ul style="list-style-type: none">■ <i>key</i> is equal to <code>IPC_PRIVATE</code>.■ <i>key</i> does not already have a semaphore identifier associated with it, and (<i>semflg</i> & <code>IPC_CREAT</code>) is true. <p>On creation, the data structure associated with the new semaphore identifier is initialized as follows:</p> <ul style="list-style-type: none">■ <code>sem_perm.cuid</code>, <code>sem_perm.uid</code>, <code>sem_perm.cgid</code>, and <code>sem_perm.gid</code> are set equal to the effective user ID and effective group ID, respectively, of the calling process.■ The access permission bits of <code>sem_perm.mode</code> are set equal to the access permission bits of <i>semflg</i>.■ <code>sem_nsems</code> is set equal to the value of <i>nsems</i>.■ <code>sem_otime</code> is set equal to 0 and <code>sem_ctime</code> is set equal to the current time.
RETURN VALUES	Upon successful completion, a non-negative integer representing a semaphore identifier is returned. Otherwise, -1 is returned and <code>errno</code> is set to indicate the error.

ERRORS

The `semget ()` function will fail if:

EACCES	A semaphore-structure identifier exists for the union of key and sensitivity label, but operation permission [see <code>intro(2)</code>] as specified by the low-order 9 bits of <i>semflg</i> would not be granted; or the sensitivity label check did not pass, and the calling process does not have the appropriate privilege override(s) in its set of effective privileges.
EEXIST	A semaphore identifier exists for <i>key</i> but both (<i>semflg</i> &IPC_CREAT) and (<i>semflg</i> &IPC_EXCL) are both true.
EFAULT	<i>slabel</i> points to an illegal address.
EINVAL	The label to which <i>slabel</i> points is not a valid sensitivity label.
EINVAL	The <i>nsems</i> argument is either less than or equal to 0 or greater than the system-imposed limit; or a semaphore identifier exists for <i>key</i> , but the number of semaphores in the set associated with it is less than <i>nsems</i> and <i>nsems</i> is not equal to 0.
ENOENT	A semaphore identifier does not exist for <i>key</i> and (<i>semflg</i> &IPC_CREAT) is false.
ENOSPC	A semaphore identifier is to be created but the system-imposed limit on the maximum number of allowed semaphores or semaphore identifiers system-wide would be exceeded.
	A semaphore identifier is to be created but the system-imposed limit on the maximum number of allowed semaphores systemwide would be exceeded.

**SUMMARY OF
TRUSTED
SOLARIS
CHANGES**

Appropriate privilege is required to override access checks.

Sensitivity labels are used together with key to determine semaphore-group identifiers.

**Trusted Solaris 8
4/01 Reference
Manual**
**SunOS 5.8
Reference Manual**

`ipcs(1)`, `ipcrm(1)`, `intro(2)`, `semctl(2)`, `semop(2)`
`stdio(3C)`

semgetl(2)

NAME	semget, semgetl – get set of semaphores
SYNOPSIS	<pre>#include <sys/types.h> #include <sys/ipc.h> #include <sys/sem.h> int semget(key_t key, int nsems, int semflg); cc [<i>flags...</i>] <i>file</i> ... -ltsol [<i>library...</i>] #include <sys/tsol/ipcl.h> int semgetl(key_t key, int nsems, int semflg, const bslabel_t *slabel);</pre>
DESCRIPTION	<p>A semaphore structure is identified by a unique combination of key and sensitivity label. This qualification of keys by sensitivity labels allows applications that use semaphore structures to be run at multiple process sensitivity labels without inadvertently sharing data.</p> <p>The <code>semget()</code> function returns the semaphore identifier associated with the union of key and the sensitivity label of the calling process. <code>semgetl()</code> returns the semaphore-structure identifier associated with the union of <i>key</i> and <i>slabel</i>. If the value of <i>slabel</i> does not match the sensitivity label of the calling process, then the effective privilege set of the process must include both <code>PRIV_IPC_MAC_READ</code> and <code>PRIV_IPC_MAC_WRITE</code>.</p> <p>If discretionary read/write access as specified by the low-order 9 bits of <i>semflg</i> is denied to the calling process, <code>semget()</code> and <code>semgetl()</code> require one or both of these privileges: <code>PRIV_IPC_DAC_READ</code> and <code>PRIV_IPC_DAC_WRITE</code>.</p> <p>A semaphore identifier and associated data structure and set containing <i>nsems</i> semaphores (see <code>intro(2)</code>) are created for <i>key</i> if one of the following is true:</p> <ul style="list-style-type: none">■ <i>key</i> is equal to <code>IPC_PRIVATE</code>.■ <i>key</i> does not already have a semaphore identifier associated with it, and (<i>semflg</i> & <code>IPC_CREAT</code>) is true. <p>On creation, the data structure associated with the new semaphore identifier is initialized as follows:</p> <ul style="list-style-type: none">■ <code>sem_perm.cuid</code>, <code>sem_perm.uid</code>, <code>sem_perm.cgid</code>, and <code>sem_perm.gid</code> are set equal to the effective user ID and effective group ID, respectively, of the calling process.■ The access permission bits of <code>sem_perm.mode</code> are set equal to the access permission bits of <i>semflg</i>.■ <code>sem_nsems</code> is set equal to the value of <i>nsems</i>.■ <code>sem_otime</code> is set equal to 0 and <code>sem_ctime</code> is set equal to the current time.
RETURN VALUES	Upon successful completion, a non-negative integer representing a semaphore identifier is returned. Otherwise, -1 is returned and <code>errno</code> is set to indicate the error.

ERRORS

The `semget ()` function will fail if:

EACCES	A semaphore-structure identifier exists for the union of key and sensitivity label, but operation permission [see <code>intro(2)</code>] as specified by the low-order 9 bits of <code>semflg</code> would not be granted; or the sensitivity label check did not pass, and the calling process does not have the appropriate privilege override(s) in its set of effective privileges.
EEXIST	A semaphore identifier exists for <i>key</i> but both (<code>semflg&IPC_CREAT</code>) and (<code>semflg&IPC_EXCL</code>) are both true.
EFAULT	<i>slabel</i> points to an illegal address.
EINVAL	The label to which <i>slabel</i> points is not a valid sensitivity label.
EINVAL	The <i>nsems</i> argument is either less than or equal to 0 or greater than the system-imposed limit; or a semaphore identifier exists for <i>key</i> , but the number of semaphores in the set associated with it is less than <i>nsems</i> and <i>nsems</i> is not equal to 0.
ENOENT	A semaphore identifier does not exist for <i>key</i> and (<code>semflg&IPC_CREAT</code>) is false.
ENOSPC	A semaphore identifier is to be created but the system-imposed limit on the maximum number of allowed semaphores or semaphore identifiers system-wide would be exceeded.
	A semaphore identifier is to be created but the system-imposed limit on the maximum number of allowed semaphores systemwide would be exceeded.

**SUMMARY OF
TRUSTED
SOLARIS
CHANGES**

Appropriate privilege is required to override access checks.

Sensitivity labels are used together with key to determine semaphore-group identifiers.

**Trusted Solaris 8
4/01 Reference
Manual**

`ipcs(1)`, `ipcrm(1)`, `intro(2)`, `semctl(2)`, `semop(2)`
`stdio(3C)`

semop(2)

NAME	semop, semtimedop – semaphore operations
SYNOPSIS	<pre>#include <sys/types.h> #include <sys/ipc.h> #include <sys/sem.h> int semop(int <i>semid</i>, struct sembuf <i>*sops</i>, size_t <i>nsops</i>); int semtimedop(int <i>semid</i>, struct sembuf <i>*sops</i>, size_t <i>nsops</i>, const struct timespec <i>*timeout</i>);</pre>
DESCRIPTION	<p>The <code>semop()</code> function is used to perform atomically an array of semaphore operations on the set of semaphores associated with the semaphore identifier specified by <i>semid</i>. The <i>sops</i> argument is a pointer to the array of semaphore-operation structures. The <i>nsops</i> argument is the number of such structures in the array.</p> <p>Each sembuf structure contains the following members:</p> <pre>short sem_num; /* semaphore number */ short sem_op; /* semaphore operation */ short sem_flg; /* operation flags */</pre> <p>Each semaphore operation specified by <code>sem_op()</code> is performed on the corresponding semaphore specified by <i>semid</i> and <code>sem_num</code>. The permission required for a semaphore operation is given as <i>{token}</i>, where <i>token</i> is the type of permission needed. The types of permission are interpreted as follows:</p> <pre>00400 READ by user 00200 ALTER by user 00040 READ by group 00020 ALTER by group 00004 READ by others 00002 ALTER by others</pre> <p>See the <i>Semaphore Operation Permissions</i> section of <code>intro(2)</code> for more information.</p> <p>The <code>sem_op</code> member specifies one of three semaphore operations:</p> <ol style="list-style-type: none">1. The <code>sem_op</code> member is a negative integer; {ALTER} <ul style="list-style-type: none">■ If <code>semval</code> (see <code>intro(2)</code>) is greater than or equal to the absolute value of <code>sem_op</code>, the absolute value of <code>sem_op</code> is subtracted from <code>semval</code>. Also, if (<code>sem_flg&SEM_UNDO</code>) is true, the absolute value of <code>sem_op</code> is added to the calling process's <code>semadj</code> value (see <code>exit(2)</code>) for the specified semaphore.■ If <code>semval</code> is less than the absolute value of <code>sem_op</code> and (<code>sem_flg&IPC_NOWAIT</code>) is true, <code>semop()</code> returns immediately.■ If <code>semval</code> is less than the absolute value of <code>sem_op</code> and (<code>sem_flg&IPC_NOWAIT</code>) is false, <code>semop()</code> increments the <code>semncnt</code> associated with the specified semaphore and suspends execution of the calling process until one of the following conditions occur:

- The value of `semval` becomes greater than or equal to the absolute value of `sem_op`. When this occurs, the value of `semncnt` associated with the specified semaphore is decremented, the absolute value of `sem_op` is subtracted from `semval` and, if (`sem_flg&SEM_UNDO`) is true, the absolute value of `sem_op` is added to the calling process's `semadj` value for the specified semaphore.
 - The `semid` for which the calling process is awaiting action is removed from the system (see `semctl(2)`). When this occurs, `errno` is set to `EIDRM` and `-1` is returned.
 - The calling process receives a signal that is to be caught. When this occurs, the value of `semncnt` associated with the specified semaphore is decremented, and the calling process resumes execution in the manner prescribed in `signal(3C)`.
2. The `sem_op` member is a positive integer; {ALTER}
- The value of `sem_op` is added to `semval` and, if (`sem_flg&SEM_UNDO`) is true, the value of `sem_op` is subtracted from the calling process's `semadj` value for the specified semaphore.
3. The `sem_op` member is 0; {READ}
- If `semval` is 0, `semop()` returns immediately.
 - If `semval` is not equal to 0 and (`sem_flg&IPC_NOWAIT`) is true, `semop()` returns immediately.
 - If `semval` is not equal to 0 and (`sem_flg&IPC_NOWAIT`) is false, `semop()` increments the `semzcnt` associated with the specified semaphore and suspends execution of the calling process until one of the following occurs:
 - The value of `semval` becomes 0, at which time the value of `semzcnt` associated with the specified semaphore is decremented.
 - The `semid` for which the calling process is awaiting action is removed from the system. When this occurs, `errno` is set to `EIDRM` and `-1` is returned.
 - The calling process receives a signal that is to be caught. When this occurs, the value of `semzcnt` associated with the specified semaphore is decremented, and the calling process resumes execution in the manner prescribed in `signal(3C)`.

If `sem_op()` is zero {READ}, the process must have discretionary and mandatory read access to the semaphore structure to which `semid` refers. Overriding these checks requires that the effective privilege set of the process include one or both of these privileges as necessary: `PRIV_IPC_DAC_READ` and `PRIV_IPC_MAC_READ`.

If `sem_op()` is a positive or a negative number {ALTER}, the process must have discretionary and mandatory write access to the semaphore structure to which `semid` refers. Overriding these checks requires that the effective privilege set of the process include one or both of these privileges as necessary: `PRIV_IPC_DAC_WRITE` and `PRIV_IPC_MAC_WRITE`.

semop(2)

	<p>Upon successful completion, the value of <i>semid</i> for each semaphore specified in the array pointed to by <i>sops</i> is set to the process ID of the calling process.</p> <p>The <code>semtimedop()</code> function behaves as <code>semop()</code> except when it must suspend execution of the calling process to complete its operation. If <code>semtimedop()</code> must suspend the calling process after the time interval specified in <i>timeout</i> expires, or if the timeout expires while the process is suspended, <code>semtimedop()</code> returns with an error. If the <code>timespec</code> structure pointed to by <i>timeout</i> is zero-valued and <code>semtimedop()</code> needs to suspend the calling process to complete the requested operation(s), it returns immediately with an error. If <i>timeout</i> is the <code>NULL</code> pointer, the behavior of <code>semtimedop()</code> is identical to that of <code>semop()</code>.</p>																								
RETURN VALUES	<p>Upon successful completion, 0 is returned. Otherwise, -1 is returned and <code>errno</code> is set to indicate the error.</p>																								
ERRORS	<p>The <code>semop()</code> and <code>semtimedop()</code> functions will fail if:</p> <table><tr><td>E2BIG</td><td>The <i>nsops</i> argument is greater than the system-imposed maximum.</td></tr><tr><td>EACCES</td><td>Operation permission is denied to the calling process (see <code>intro(2)</code>), and the calling process does not have the appropriate privilege(s) in its set of effective privileges.</td></tr><tr><td>EAGAIN</td><td>The operation would result in suspension of the calling process but (<i>sem_flg</i> & <code>IPC_NOWAIT</code>) is true.</td></tr><tr><td>EFAULT</td><td>The <i>sops</i> argument points to an illegal address.</td></tr><tr><td>EFBIG</td><td>The value of <i>sem_num</i> is less than 0 or greater than or equal to the number of semaphores in the set associated with <i>semid</i>.</td></tr><tr><td>EIDRM</td><td>A <i>semid</i> was removed from the system.</td></tr><tr><td>EINTR</td><td>A signal was received.</td></tr><tr><td>EINVAL</td><td>The <i>semid</i> argument is not a valid semaphore identifier, or the number of individual semaphores for which the calling process requests a <code>SEM_UNDO</code> would exceed the limit.</td></tr><tr><td>ENOSPC</td><td>The limit on the number of individual processes requesting an <code>SEM_UNDO</code> would be exceeded.</td></tr><tr><td>ERANGE</td><td>An operation would cause a <i>semval</i> or a <i>semadj</i> value to overflow the system-imposed limit.</td></tr></table> <p>The <code>semtimedop()</code> function will fail if:</p> <table><tr><td>EAGAIN</td><td>The timeout expired before the requested operation could be completed.</td></tr></table> <p>The <code>semtimedop()</code> function will fail if one of the following is detected:</p> <table><tr><td>EFAULT</td><td>The <i>timeout</i> argument points to an illegal address.</td></tr></table>	E2BIG	The <i>nsops</i> argument is greater than the system-imposed maximum.	EACCES	Operation permission is denied to the calling process (see <code>intro(2)</code>), and the calling process does not have the appropriate privilege(s) in its set of effective privileges.	EAGAIN	The operation would result in suspension of the calling process but (<i>sem_flg</i> & <code>IPC_NOWAIT</code>) is true.	EFAULT	The <i>sops</i> argument points to an illegal address.	EFBIG	The value of <i>sem_num</i> is less than 0 or greater than or equal to the number of semaphores in the set associated with <i>semid</i> .	EIDRM	A <i>semid</i> was removed from the system.	EINTR	A signal was received.	EINVAL	The <i>semid</i> argument is not a valid semaphore identifier, or the number of individual semaphores for which the calling process requests a <code>SEM_UNDO</code> would exceed the limit.	ENOSPC	The limit on the number of individual processes requesting an <code>SEM_UNDO</code> would be exceeded.	ERANGE	An operation would cause a <i>semval</i> or a <i>semadj</i> value to overflow the system-imposed limit.	EAGAIN	The timeout expired before the requested operation could be completed.	EFAULT	The <i>timeout</i> argument points to an illegal address.
E2BIG	The <i>nsops</i> argument is greater than the system-imposed maximum.																								
EACCES	Operation permission is denied to the calling process (see <code>intro(2)</code>), and the calling process does not have the appropriate privilege(s) in its set of effective privileges.																								
EAGAIN	The operation would result in suspension of the calling process but (<i>sem_flg</i> & <code>IPC_NOWAIT</code>) is true.																								
EFAULT	The <i>sops</i> argument points to an illegal address.																								
EFBIG	The value of <i>sem_num</i> is less than 0 or greater than or equal to the number of semaphores in the set associated with <i>semid</i> .																								
EIDRM	A <i>semid</i> was removed from the system.																								
EINTR	A signal was received.																								
EINVAL	The <i>semid</i> argument is not a valid semaphore identifier, or the number of individual semaphores for which the calling process requests a <code>SEM_UNDO</code> would exceed the limit.																								
ENOSPC	The limit on the number of individual processes requesting an <code>SEM_UNDO</code> would be exceeded.																								
ERANGE	An operation would cause a <i>semval</i> or a <i>semadj</i> value to overflow the system-imposed limit.																								
EAGAIN	The timeout expired before the requested operation could be completed.																								
EFAULT	The <i>timeout</i> argument points to an illegal address.																								

	semop(2)
EINVAL	The <i>timeout</i> argument specified a <i>tv_sec</i> or <i>tv_nsec</i> value less than 0, or a <i>tv_nsec</i> value greater than or equal to 1000 million.
	Appropriate privilege is required to override access checks.
	ipcs(1), Intro(2), exec(2), fork(2), semctl(2), semget(2)
	signal(3C)

semtimedop(2)

NAME	semop, semtimedop – semaphore operations
SYNOPSIS	<pre>#include <sys/types.h> #include <sys/ipc.h> #include <sys/sem.h> int semop(int <i>semid</i>, struct sembuf <i>*sops</i>, size_t <i>nsops</i>); int semtimedop(int <i>semid</i>, struct sembuf <i>*sops</i>, size_t <i>nsops</i>, const struct timespec <i>*timeout</i>);</pre>
DESCRIPTION	<p>The <code>semop()</code> function is used to perform atomically an array of semaphore operations on the set of semaphores associated with the semaphore identifier specified by <i>semid</i>. The <i>sops</i> argument is a pointer to the array of semaphore-operation structures. The <i>nsops</i> argument is the number of such structures in the array.</p> <p>Each <code>sembuf</code> structure contains the following members:</p> <pre>short sem_num; /* semaphore number */ short sem_op; /* semaphore operation */ short sem_flg; /* operation flags */</pre> <p>Each semaphore operation specified by <code>sem_op()</code> is performed on the corresponding semaphore specified by <i>semid</i> and <code>sem_num</code>. The permission required for a semaphore operation is given as <i>{token}</i>, where <i>token</i> is the type of permission needed. The types of permission are interpreted as follows:</p> <pre>00400 READ by user 00200 ALTER by user 00040 READ by group 00020 ALTER by group 00004 READ by others 00002 ALTER by others</pre> <p>See the <i>Semaphore Operation Permissions</i> section of <code>intro(2)</code> for more information.</p> <p>The <code>sem_op</code> member specifies one of three semaphore operations:</p> <ol style="list-style-type: none">1. The <code>sem_op</code> member is a negative integer; {ALTER} <ul style="list-style-type: none">■ If <code>semval</code> (see <code>intro(2)</code>) is greater than or equal to the absolute value of <code>sem_op</code>, the absolute value of <code>sem_op</code> is subtracted from <code>semval</code>. Also, if (<code>sem_flg&SEM_UNDO</code>) is true, the absolute value of <code>sem_op</code> is added to the calling process's <code>semadj</code> value (see <code>exit(2)</code>) for the specified semaphore.■ If <code>semval</code> is less than the absolute value of <code>sem_op</code> and (<code>sem_flg&IPC_NOWAIT</code>) is true, <code>semop()</code> returns immediately.■ If <code>semval</code> is less than the absolute value of <code>sem_op</code> and (<code>sem_flg&IPC_NOWAIT</code>) is false, <code>semop()</code> increments the <code>semncnt</code> associated with the specified semaphore and suspends execution of the calling process until one of the following conditions occur:

- The value of `semval` becomes greater than or equal to the absolute value of `sem_op`. When this occurs, the value of `semncnt` associated with the specified semaphore is decremented, the absolute value of `sem_op` is subtracted from `semval` and, if (`sem_flg&SEM_UNDO`) is true, the absolute value of `sem_op` is added to the calling process's `semadj` value for the specified semaphore.
 - The `semid` for which the calling process is awaiting action is removed from the system (see `semctl(2)`). When this occurs, `errno` is set to `EIDRM` and `-1` is returned.
 - The calling process receives a signal that is to be caught. When this occurs, the value of `semncnt` associated with the specified semaphore is decremented, and the calling process resumes execution in the manner prescribed in `signal(3C)`.
2. The `sem_op` member is a positive integer; {ALTER}
- The value of `sem_op` is added to `semval` and, if (`sem_flg&SEM_UNDO`) is true, the value of `sem_op` is subtracted from the calling process's `semadj` value for the specified semaphore.
3. The `sem_op` member is 0; {READ}
- If `semval` is 0, `semop()` returns immediately.
 - If `semval` is not equal to 0 and (`sem_flg&IPC_NOWAIT`) is true, `semop()` returns immediately.
 - If `semval` is not equal to 0 and (`sem_flg&IPC_NOWAIT`) is false, `semop()` increments the `semzcnt` associated with the specified semaphore and suspends execution of the calling process until one of the following occurs:
 - The value of `semval` becomes 0, at which time the value of `semzcnt` associated with the specified semaphore is decremented.
 - The `semid` for which the calling process is awaiting action is removed from the system. When this occurs, `errno` is set to `EIDRM` and `-1` is returned.
 - The calling process receives a signal that is to be caught. When this occurs, the value of `semzcnt` associated with the specified semaphore is decremented, and the calling process resumes execution in the manner prescribed in `signal(3C)`.

If `sem_op()` is zero {READ}, the process must have discretionary and mandatory read access to the semaphore structure to which `semid` refers. Overriding these checks requires that the effective privilege set of the process include one or both of these privileges as necessary: `PRIV_IPC_DAC_READ` and `PRIV_IPC_MAC_READ`.

If `sem_op()` is a positive or a negative number {ALTER}, the process must have discretionary and mandatory write access to the semaphore structure to which `semid` refers. Overriding these checks requires that the effective privilege set of the process include one or both of these privileges as necessary: `PRIV_IPC_DAC_WRITE` and `PRIV_IPC_MAC_WRITE`.

semtimedop(2)

	<p>Upon successful completion, the value of <i>semid</i> for each semaphore specified in the array pointed to by <i>sops</i> is set to the process ID of the calling process.</p> <p>The <code>semtimedop()</code> function behaves as <code>semop()</code> except when it must suspend execution of the calling process to complete its operation. If <code>semtimedop()</code> must suspend the calling process after the time interval specified in <i>timeout</i> expires, or if the timeout expires while the process is suspended, <code>semtimedop()</code> returns with an error. If the <code>timespec</code> structure pointed to by <i>timeout</i> is zero-valued and <code>semtimedop()</code> needs to suspend the calling process to complete the requested operation(s), it returns immediately with an error. If <i>timeout</i> is the <code>NULL</code> pointer, the behavior of <code>semtimedop()</code> is identical to that of <code>semop()</code>.</p>																								
RETURN VALUES	<p>Upon successful completion, 0 is returned. Otherwise, -1 is returned and <code>errno</code> is set to indicate the error.</p>																								
ERRORS	<p>The <code>semop()</code> and <code>semtimedop()</code> functions will fail if:</p> <table><tr><td>E2BIG</td><td>The <i>nsops</i> argument is greater than the system-imposed maximum.</td></tr><tr><td>EACCES</td><td>Operation permission is denied to the calling process (see <code>intro(2)</code>), and the calling process does not have the appropriate privilege(s) in its set of effective privileges.</td></tr><tr><td>EAGAIN</td><td>The operation would result in suspension of the calling process but (<i>sem_flg</i> & <code>IPC_NOWAIT</code>) is true.</td></tr><tr><td>EFAULT</td><td>The <i>sops</i> argument points to an illegal address.</td></tr><tr><td>EFBIG</td><td>The value of <i>sem_num</i> is less than 0 or greater than or equal to the number of semaphores in the set associated with <i>semid</i>.</td></tr><tr><td>EIDRM</td><td>A <i>semid</i> was removed from the system.</td></tr><tr><td>EINTR</td><td>A signal was received.</td></tr><tr><td>EINVAL</td><td>The <i>semid</i> argument is not a valid semaphore identifier, or the number of individual semaphores for which the calling process requests a <code>SEM_UNDO</code> would exceed the limit.</td></tr><tr><td>ENOSPC</td><td>The limit on the number of individual processes requesting an <code>SEM_UNDO</code> would be exceeded.</td></tr><tr><td>ERANGE</td><td>An operation would cause a <i>semval</i> or a <i>semadj</i> value to overflow the system-imposed limit.</td></tr></table> <p>The <code>semtimedop()</code> function will fail if:</p> <table><tr><td>EAGAIN</td><td>The timeout expired before the requested operation could be completed.</td></tr></table> <p>The <code>semtimedop()</code> function will fail if one of the following is detected:</p> <table><tr><td>EFAULT</td><td>The <i>timeout</i> argument points to an illegal address.</td></tr></table>	E2BIG	The <i>nsops</i> argument is greater than the system-imposed maximum.	EACCES	Operation permission is denied to the calling process (see <code>intro(2)</code>), and the calling process does not have the appropriate privilege(s) in its set of effective privileges.	EAGAIN	The operation would result in suspension of the calling process but (<i>sem_flg</i> & <code>IPC_NOWAIT</code>) is true.	EFAULT	The <i>sops</i> argument points to an illegal address.	EFBIG	The value of <i>sem_num</i> is less than 0 or greater than or equal to the number of semaphores in the set associated with <i>semid</i> .	EIDRM	A <i>semid</i> was removed from the system.	EINTR	A signal was received.	EINVAL	The <i>semid</i> argument is not a valid semaphore identifier, or the number of individual semaphores for which the calling process requests a <code>SEM_UNDO</code> would exceed the limit.	ENOSPC	The limit on the number of individual processes requesting an <code>SEM_UNDO</code> would be exceeded.	ERANGE	An operation would cause a <i>semval</i> or a <i>semadj</i> value to overflow the system-imposed limit.	EAGAIN	The timeout expired before the requested operation could be completed.	EFAULT	The <i>timeout</i> argument points to an illegal address.
E2BIG	The <i>nsops</i> argument is greater than the system-imposed maximum.																								
EACCES	Operation permission is denied to the calling process (see <code>intro(2)</code>), and the calling process does not have the appropriate privilege(s) in its set of effective privileges.																								
EAGAIN	The operation would result in suspension of the calling process but (<i>sem_flg</i> & <code>IPC_NOWAIT</code>) is true.																								
EFAULT	The <i>sops</i> argument points to an illegal address.																								
EFBIG	The value of <i>sem_num</i> is less than 0 or greater than or equal to the number of semaphores in the set associated with <i>semid</i> .																								
EIDRM	A <i>semid</i> was removed from the system.																								
EINTR	A signal was received.																								
EINVAL	The <i>semid</i> argument is not a valid semaphore identifier, or the number of individual semaphores for which the calling process requests a <code>SEM_UNDO</code> would exceed the limit.																								
ENOSPC	The limit on the number of individual processes requesting an <code>SEM_UNDO</code> would be exceeded.																								
ERANGE	An operation would cause a <i>semval</i> or a <i>semadj</i> value to overflow the system-imposed limit.																								
EAGAIN	The timeout expired before the requested operation could be completed.																								
EFAULT	The <i>timeout</i> argument points to an illegal address.																								

SUMMARY OF
TRUSTED
SOLARIS
Trusted Solaris 4/01 Reference Manual

	semimedop(2)
EINVAL	The <i>timeout</i> argument specified a <i>tv_sec</i> or <i>tv_nsec</i> value less than 0, or a <i>tv_nsec</i> value greater than or equal to 1000 million.
	Appropriate privilege is required to override access checks.
	ipcs(1), Intro(2), exec(2), fork(2), semctl(2), semget(2)
	signal(3C)

setaudit(2)

NAME	getaudit, setaudit, getaudit_addr, setaudit_addr – Get and set process audit information				
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lbsm -lsocket -lnsl -lintl [<i>library</i> ...] #include <sys/param.h> #include <bsm/audit.h> int getaudit(struct auditinfo *<i>info</i>) ; int setaudit(struct auditinfo *<i>info</i>) ; int getaudit_addr(struct auditinfo_addr *<i>info</i>, int <i>length</i>) ; int setaudit_addr(struct auditinfo_addr *<i>info</i>, int <i>length</i>) ;</pre>				
DESCRIPTION	<p>getaudit () gets the audit ID, the preselection mask, the terminal ID, and the audit session ID of the current process.</p> <p>Note that getaudit () may fail and return an E2BIG errno if the address field in the terminal ID is larger than 32 bits. In this case, getaudit_addr () should be used.</p> <p>setaudit () sets the audit ID, the preselection mask, the terminal ID, and the audit session ID for the current process.</p> <p>The getaudit_addr () function returns a variable length auditinfo_addr structure that contains the audit ID, the preselection mask, the terminal ID, and the audit session ID for the current process. The terminal ID contains a size field that indicates the size of the network address.</p> <p>The setaudit_addr () function sets the audit ID, the preselection mask, the terminal ID, and the audit session ID for the current process. The values are taken from the variable length structure auditinfo_addr. The terminal ID contains a size field that indicates the size of the network address.</p> <p>The info structure used to pass the process audit information contains the following members:</p> <pre>au_id_t ai_auid; /* audit user ID */ au_mask_t ai_mask; /* preselection mask */ au_tid_t ai_termid; /* terminal ID */ au_asid_t ai_asid; /* audit session ID */</pre> <p>To execute these commands successfully, a process needs certain privileges in its set of effective privileges: for getaudit (), a process needs PRIV_SYS_AUDIT, PRIV_PROC_AUDIT_TCB, or PRIV_PROC_AUDIT_APPL; for setaudit (), PRIV_SYS_AUDIT.</p>				
RETURN VALUES	<p>getaudit () and setaudit () return:</p> <table> <tr> <td>0</td><td>On success.</td></tr> <tr> <td>-1</td><td>On failure, and set errno to indicate the error.</td></tr> </table>	0	On success.	-1	On failure, and set errno to indicate the error.
0	On success.				
-1	On failure, and set errno to indicate the error.				

		setaudit(2)
ERRORS	The getaudit() and setaudit() functions will fail if:	
	EFAULT	The <i>info</i> parameter points outside the process's allocated address space.
	EPERM	The process did not have the appropriate privilege.
USAGE	Only processes with the appropriate privileges may successfully execute these calls.	
SUMMARY OF TRUSTED SOLARIS CHANGES	<p>This functionality is active only if auditing is enabled. By default, auditing is enabled in the Trusted Solaris environment. See <i>Trusted Solaris Audit Administration Manual</i> for more information.</p> <p>As explained in DESCRIPTION, privileges are needed to run this command successfully.</p>	
Trusted Solaris 8 4/01 Reference Manual	audit(2)	

setaudit_addr(2)

NAME	getaudit, setaudit, getaudit_addr, setaudit_addr – Get and set process audit information				
SYNOPSIS	<pre>cc [<i>flag</i> ...] <i>file</i> ... -lbsm -lsocket -lnsl -lintl [<i>library</i> ...] #include <sys/param.h> #include <bsm/audit.h> int getaudit(struct auditinfo *<i>info</i>) ; int setaudit(struct auditinfo *<i>info</i>) ; int getaudit_addr(struct auditinfo_addr *<i>info</i>, int <i>length</i>) ; int setaudit_addr(struct auditinfo_addr *<i>info</i>, int <i>length</i>) ;</pre>				
DESCRIPTION	<p>getaudit() gets the audit ID, the preselection mask, the terminal ID, and the audit session ID of the current process.</p> <p>Note that getaudit() may fail and return an E2BIG errno if the address field in the terminal ID is larger than 32 bits. In this case, getaudit_addr() should be used.</p> <p>setaudit() sets the audit ID, the preselection mask, the terminal ID, and the audit session ID for the current process.</p> <p>The getaudit_addr() function returns a variable length auditinfo_addr structure that contains the audit ID, the preselection mask, the terminal ID, and the audit session ID for the current process. The terminal ID contains a size field that indicates the size of the network address.</p> <p>The setaudit_addr() function sets the audit ID, the preselection mask, the terminal ID, and the audit session ID for the current process. The values are taken from the variable length structure auditinfo_addr. The terminal ID contains a size field that indicates the size of the network address.</p> <p>The info structure used to pass the process audit information contains the following members:</p> <pre>au_id_t ai_auid; /* audit user ID */ au_mask_t ai_mask; /* preselection mask */ au_tid_t ai_termid; /* terminal ID */ au_asid_t ai_asid; /* audit session ID */</pre> <p>To execute these commands successfully, a process needs certain privileges in its set of effective privileges: for getaudit(), a process needs PRIV_SYS_AUDIT, PRIV_PROC_AUDIT_TCB, or PRIV_PROC_AUDIT_APPL; for setaudit(), PRIV_SYS_AUDIT.</p>				
RETURN VALUES	<p>getaudit() and setaudit() return:</p> <table> <tr> <td>0</td><td>On success.</td></tr> <tr> <td>-1</td><td>On failure, and set errno to indicate the error.</td></tr> </table>	0	On success.	-1	On failure, and set errno to indicate the error.
0	On success.				
-1	On failure, and set errno to indicate the error.				

setaudit_addr(2)

ERRORS The getaudit() and setaudit() functions will fail if:

EFAULT The *info* parameter points outside the process's allocated address space.

EPERM The process did not have the appropriate privilege.

USAGE Only processes with the appropriate privileges may successfully execute these calls.

SUMMARY OF TRUSTED SOLARIS CHANGES This functionality is active only if auditing is enabled. By default, auditing is enabled in the Trusted Solaris environment. See *Trusted Solaris Audit Administration Manual* for more information.

As explained in DESCRIPTION, privileges are needed to run this command successfully.

**Trusted Solaris 8
4/01 Reference
Manual**

audit(2)

setaudit(2)

NAME	getaudit, setaudit – Get and set user audit identity				
SYNOPSIS	<pre>cc [flags...] file ... -lbsm -lsocket -lnsl -lintl [library...] #include <sys/param.h> #include <bsm/audit.h> int getaudit(au_id_t *auid); int setaudit(au_id_t *auid);</pre>				
DESCRIPTION	<p>The <code>getaudit()</code> function returns the audit user ID for the current process. This value is initially set at login time and inherited by all child processes. This value does not change when the real/effective user IDs change, so it can be used to identify the logged-in user even when running a <code>setuid</code> program. The audit user ID governs audit decisions for a process.</p> <p>The <code>setaudit()</code> function sets the audit user ID for the current process.</p> <p>Only a process with the <code>PRIV_SYS_AUDIT</code> privilege asserted may successfully set its user identity. To get its identity successfully, a process must have <code>PRIV_SYS_AUDIT</code>, <code>PRIV_PROC_AUDIT_TCB</code>, or <code>PRIV_PROC_AUDIT_APPL</code> in its set of effective privileges.</p>				
RETURN VALUES	<p>Upon successful completion, the <code>getaudit()</code> function returns the audit user ID of the current process on success. Otherwise, it returns <code>-1</code> and sets <code>errno</code> to indicate the error.</p> <p>Upon successful completion the <code>setaudit()</code> function returns <code>0</code>. Otherwise, <code>-1</code> is returned and <code>errno</code> is set to indicate the error.</p>				
ERRORS	<p>The <code>getaudit()</code> and <code>setaudit()</code> functions will fail if:</p> <table><tr><td>EFAULT</td><td>The <i>auid</i> argument points to an invalid address.</td></tr><tr><td>EPERM</td><td>The process does not have the appropriate privileges.</td></tr></table>	EFAULT	The <i>auid</i> argument points to an invalid address.	EPERM	The process does not have the appropriate privileges.
EFAULT	The <i>auid</i> argument points to an invalid address.				
EPERM	The process does not have the appropriate privileges.				
SUMMARY OF TRUSTED SOLARIS CHANGES	<p>This functionality is active only if auditing is enabled. By default, auditing is enabled in the Trusted Solaris environment. See <i>Trusted Solaris Audit Administration Manual</i> for more information.</p> <p>The privileges explained in DESCRIPTION are needed to run this command successfully.</p> <p>These system calls have been superseded by <code>getaudit()</code> and <code>setaudit()</code>.</p>				
Trusted Solaris 8 4/01 Reference Manual	<code>audit(2)</code> , <code>getaudit(2)</code>				

NAME	setclearance – Set process clearance								
SYNOPSIS	<pre>cc [flags...] file ... -ltsol [library...] #include <tsol/label.h> int setclearance(bclear_t *clearance_p) ;</pre>								
DESCRIPTION	<p>setclearance() is used to set the clearance for the calling process provided it has the PRIV_PROC_SETCLR privilege in its set of effective privileges. setclearance() verifies that the information pointed to by <i>clearance_p</i> is formatted correctly, and that the resulting clearance will dominate the sensitivity label of the process.</p>								
RETURN VALUES	<p>setclearance() returns:</p> <p>0 On success.</p> <p>-1 On failure, and sets errno to indicate the error.</p>								
ERRORS	<p>setclearance() fails and does not set the process clearance if any of these conditions prevails:</p> <table> <tr> <td>EFAULT</td><td>The <i>clearance_p</i> argument points to an invalid address.</td></tr> <tr> <td>EINVAL</td><td>The <i>clearance_p</i> argument does not point to a properly formatted clearance.</td></tr> <tr> <td></td><td>The clearance pointed to by <i>clearance_p</i> does not dominate the process sensitivity label.</td></tr> <tr> <td>EPERM</td><td>The calling process does not have the necessary privilege (PRIV_PROC_SETCLR) to set the clearance.</td></tr> </table>	EFAULT	The <i>clearance_p</i> argument points to an invalid address.	EINVAL	The <i>clearance_p</i> argument does not point to a properly formatted clearance.		The clearance pointed to by <i>clearance_p</i> does not dominate the process sensitivity label.	EPERM	The calling process does not have the necessary privilege (PRIV_PROC_SETCLR) to set the clearance.
EFAULT	The <i>clearance_p</i> argument points to an invalid address.								
EINVAL	The <i>clearance_p</i> argument does not point to a properly formatted clearance.								
	The clearance pointed to by <i>clearance_p</i> does not dominate the process sensitivity label.								
EPERM	The calling process does not have the necessary privilege (PRIV_PROC_SETCLR) to set the clearance.								
Trusted Solaris 8 4/01 Reference Manual	getclearance(2)								

setcmwlabel(2)

NAME	setcmwlabel, fsetcmwlabel, lsetcmwlabel – Set CMW label of a file
SYNOPSIS	<pre>cc [flags...] file ... -ltsol [library...] #include <tsol/label.h> int setcmwlabel(const char *path, const bclabel_t *label_p, const setting_flag_t flag); int fsetcmwlabel(int fd, const bclabel_t *label_p, const setting_flag_t flag); int lsetcmwlabel(const char *path, const bclabel_t *label_p, const setting_flag_t flag);</pre>
DESCRIPTION	<p>The file that is named by <i>path</i> or referred to by <i>fd</i> has its CMW label changed as specified provided the file resides on a file system that supports the setting of labels on individual objects.</p> <p>If <i>flag</i> equals SETCL_ALL, then both parts of the file's CMW label are to be set and the following checks must be made:</p> <ul style="list-style-type: none"> ■ The sensitivity label of <i>label_p</i> must be in the sensitivity label range of the containing file system. ■ If the sensitivity label of <i>label_p</i> equals the existing sensitivity label, then neither PRIV_FILE_UPGRADE_SL nor PRIV_FILE_DOWNGRADE_SL is required. ■ If the sensitivity label of <i>label_p</i> dominates but does not equal the existing sensitivity label (an upgrade), then the calling process must have PRIV_FILE_UPGRADE_SL in its set of effective privileges. ■ If the sensitivity label of <i>label_p</i> does not dominate the existing sensitivity label (a downgrade), then the calling process must have PRIV_FILE_DOWNGRADE_SL in its set of effective privileges. ■ If the sensitivity label operation is a downgrade and the calling process is not the owner of the file, then the calling process must have PRIV_FILE_OWNER in its set of effective privileges. <p>If <i>flag</i> equals SETCL_SL, then the sensitivity label of the file's CMW label is to be set and the following checks must be made:</p> <ul style="list-style-type: none"> ■ The sensitivity label of <i>label_p</i> must be in the sensitivity label range of the containing file system. ■ If the sensitivity label of <i>label_p</i> equals the existing sensitivity label, then neither PRIV_FILE_UPGRADE_SL nor PRIV_FILE_DOWNGRADE_SL is required. ■ If the sensitivity label of <i>label_p</i> dominates but does not equal the existing sensitivity label (an upgrade), then the calling process must have PRIV_FILE_UPGRADE_SL in its set of effective privileges. ■ If the sensitivity label of <i>label_p</i> does not dominate the existing sensitivity label (a downgrade), then the calling process must have PRIV_FILE_DOWNGRADE_SL in

its set of effective privileges.

- If the operation is a sensitivity label downgrade and the calling process is not the owner of the file, then the calling process must have `PRIV_FILE_OWNER` in its set of effective privileges.

There are several checks that are applicable if the sensitivity label is being changed:

- The calling process must have discretionary write access to the file.
- If there is an open file descriptor reference to the file, then the calling process must have `PRIV_PROC_TRANQUIL` in its set of effective privileges.

`setcmwlabel()` and `lsetcmwlabel()` function identically except when the final component is a symbolic link. If the final component is a symbolic link, `lsetcmwlabel()` sets the CMW label of the symbolic link, but `setcmwlabel()` sets the CMW label of the object referred to by the symbolic link.

NOTES If the sensitivity label is being set, then the calling process is responsible for verifying that sensitivity label is within the accreditation range of the system.

RETURN VALUES `setcmwlabel()`, `fsetcmwlabel()`, and `lsetcmwlabel()` return:

- 0 On success.
- 1 On failure, and set `errno` to indicate the error.

ERRORS `setcmwlabel()` and `lsetcmwlabel()` fail and the file is unchanged if any of these conditions prevails:

- | | |
|---------------|---|
| EACCES | Search permission is denied for a component of the path prefix of <i>path</i> .

The calling process does not have mandatory write access to the final component of <i>path</i> because the sensitivity label of the final component of <i>path</i> does not dominate the sensitivity label of the calling process and the calling process does not have <code>PRIV_FILE_MAC_WRITE</code> in its set of effective privileges.

The calling process does not have discretionary write access to the final component of <i>path</i> . |
| EBUSY | There is an open file descriptor reference to the final component of <i>path</i> and the calling process does not have <code>PRIV_PROC_TRANQUIL</code> in its set of effective privileges. |
| EFAULT | <i>path</i> or <i>label_p</i> points outside the allocated address space of the process. |
| EINVAL | <i>path</i> does not reside on a file system that supports the setting of labels on individual objects. |

setcmwlabel(2)

	The sensitivity label of <i>label_p</i> is not in the sensitivity label range of the containing file system.
EIO	An I/O error occurred while reading from or writing to the file system.
ELOOP	Too many symbolic links were encountered in translating <i>path</i> .
ENAMETOOLONG	The length of the path argument exceeds PATH_MAX. A pathname component is longer than NAME_MAX [see sysconf(3C)] while _POSIX_NO_TRUNC is in effect. See pathconf(2).
ENOENT	The file referred to by <i>path</i> does not exist.
ENOTDIR	A component of the path prefix of <i>path</i> is not a directory.
EPERM	The calling process does not have mandatory write access to the final component of <i>path</i> because the sensitivity label of the final component of <i>path</i> is outside the clearance of the calling process and the calling process does not have PRIV_FILE_MAC_WRITE in its set of effective privileges. A calling process that is not the owner of the file attempted to downgrade the sensitivity label associated with the final component of <i>path</i> but did not have PRIV_FILE_OWNER in its set of effective privileges. The calling process attempted to upgrade the sensitivity label associated with the final component of <i>path</i> but did not have PRIV_FILE_UPGRADE_SL in its set of effective privileges. The calling process attempted to downgrade the sensitivity label associated with the final component of <i>path</i> but did not have PRIV_FILE_DOWNGRADE_SL in its set of effective privileges.
EROFS	The file referred to by <i>path</i> resides on a read-only file system.
fsetcmwlabel()	fails if any of these conditions prevails:
EBADF	<i>fd</i> does not refer to a valid descriptor.
EBUSY	There is an open file descriptor reference to the object referred to by the descriptor and the calling process does not have PRIV_PROC_TRANQUIL in its set of effective privileges.
EFAULT	<i>label_p</i> points outside the allocated address space of the process.
EINVAL	<i>fd</i> refers to a socket, not a file. <i>fd</i> does not refer to a file on a file system that supports the setting of labels on individual objects.

		setcmwlabel(2)
		The sensitivity label of <i>label_p</i> is not in the sensitivity label range of the containing file system.
EIO		An I/O error occurred while reading from or writing to the file system.
		The calling process is not the owner of the file, attempted to downgrade the sensitivity label associated with the file, but did not have PRIV_FILE_OWNER in its set of effective privileges.
		The calling process attempted to upgrade the sensitivity label associated with the file but did not have PRIV_FILE_UPGRADE_SL in its set of effective privileges.
		The calling process attempted to downgrade the sensitivity label associated with the file but did not have PRIV_FILE_DOWNGRADE_SL in its set of effective privileges.
EPERM		The calling process does not have mandatory write access to the object referred to by <i>fd</i> because the sensitivity label of the object referred to by <i>fd</i> is outside the clearance of the calling process and the calling process does not have PRIV_FILE_MAC_WRITE in its set of effective privileges.
		A calling process that is not the owner of the file attempted to downgrade the sensitivity label associated with the object referred to by <i>fd</i> but did not have PRIV_FILE_OWNER in its set of effective privileges.
		The calling process attempted to upgrade the sensitivity label associated with the object referred to by <i>fd</i> but did not have PRIV_FILE_UPGRADE_SL in its set of effective privileges.
		The calling process attempted to downgrade the sensitivity label associated with the object referred to by <i>fd</i> but did not have PRIV_FILE_DOWNGRADE_SL in its set of effective privileges.
EROFS		The file referred to by <i>fd</i> resides on a read-only file system.
	getcmwfsrange(2), getcmwlabel(2)	

setcmwplabel(2)

NAME	setcmwplabel – set process CMW label
SYNOPSIS	<pre>cc [flag...] file... -ltsol [library...] #include <tsol/label.h> int setcmwplabel(bclabel_t *label_p, setting_flag_t flag);</pre>
DESCRIPTION	<p>setcmwplabel() sets the sensitivity label or the CMW label for the process making the call. The <i>flag</i> argument identifies which label to set:</p> <p>SETCL_ALL Set the entire CMW label of the process.</p> <p>SETCL_SL Set only the sensitivity label.</p> <p>setcmwplabel() verifies that the CMW label to which <i>label_p</i> points is formatted correctly and that the resulting label would satisfy the requirement that the clearance must dominate the sensitivity label of the process.</p> <p>When <i>flag</i> limits the setting to a single portion of the CMW label, setcmwplabel() ignores the other value in <i>label_p</i>. If the specified value for sensitivity label does not match current value of the process, the set of effective privileges of the calling process must include PRIV_PROC_SETSL.</p>
RETURN VALUES	<p>setcmwplabel() returns:</p> <p>0 On success.</p> <p>-1 On failure, and sets errno to indicate the error.</p>
ERRORS	<p>setcmwplabel() fails and does not set the process CMW label if any of these conditions is true:</p> <p>EBUSY The process is being accessed through the /proc filesystem, which can happen when the process is either being traced or debugged, and the caller of setcmwplabel() lacks the PRIV_PROC_TRANQUIL privilege.</p> <p> The <i>label_p</i> argument points to an invalid address.</p> <p>EFAULT The <i>label_p</i> argument points to an invalid address.</p> <p>EINVAL The <i>label_p</i> argument points to an improperly formatted label.</p> <p> The <i>label_p</i> argument and the <i>flag</i> argument would cause the process sensitivity label not to be dominated by the clearance.</p> <p>EPERM The calling process lacks the PRIV_PROC_SETSL privilege necessary to set the sensitivity label specified by <i>flag</i>.</p>
Trusted Solaris 8 4/01 Reference Manual	getcmwplabel(2)

NAME	setuid, setegid, seteuid, setgid – set user and group IDs
SYNOPSIS	<pre> #include <sys/types.h> #include <unistd.h> int setuid(uid_t uid) ; int setegid(gid_t egid) ; int seteuid(uid_t euid) ; int setgid(gid_t gid) ; </pre>
DESCRIPTION	<p>The <code>setuid()</code> function sets the real user ID, effective user ID, and saved user ID of the calling process. The <code>setgid()</code> function sets the real group ID, effective group ID, and saved group ID of the calling process. The <code>setegid()</code> and <code>seteuid()</code> functions set the effective group and user IDs respectively for the calling process. See <code>intro(2)</code> for more information on real, effective, and saved user and group IDs.</p> <p>Privilege-unaware programs change their UIDs either to gain or give up rights associated with the new UID. To simulate that action in a privilege-based system rather than a UID-based system, the effective and saved privilege sets are modified across <code>setuid</code> calls according to the following algorithm:</p> <pre> if ((old_uid == start_uid) && (new_uid != start_uid)) { saved_privileges = effective_privileges; effective_privileges = empty; } else if ((old_uid != start_uid) && (new_uid == start_uid)) { effective_privileges = saved_privileges; } </pre> <p>Since set UID programs may not be aware of privileges, their privilege bracketing (their use of UID changes to obtain or give up rights) is tracked in the privilege sets. When a set UID program changes from its saved UID ID to the calling user ID, it gives up its privilege. When it changes back to the saved UID ID, it regains privilege.</p> <p>At login time, the real user ID, effective user ID, and saved user ID of the login process are set to the login ID of the user responsible for the creation of the process. The same is true for the real, effective, and saved group IDs; they are set to the group ID of the user responsible for the creation of the process.</p> <p>When a process calls one of the <code>exec</code> family of functions (see <code>exec(2)</code>) to execute a file (program), the user and/or group identifiers associated with the process can change. If the file executed is a set-user-ID file, the effective and saved user IDs of the process are set to the owner of the file executed. If the file executed is a set-group-ID file, the effective and saved group IDs of the process are set to the group of the file executed. If the file executed is not a set-user-ID or set-group-ID file, the effective user ID, saved user ID, effective group ID, and saved group ID are not changed.</p> <p>If the process calling <code>setuid()</code> has the <code>PRIV_PROC_SETID</code> privilege, the real, effective, and saved user IDs are set to the <code>uid</code> parameter.</p>

setegid(2)

If the process calling `setuid()` does not have the `PRIV_PROC_SETID` privilege, but *uid* is either the real user ID or the saved user ID of the calling process, the effective user ID is set to *uid*.

If the new user ID differs from the initial user ID under which this program began execution, the saved privilege set is replaced by the effective privilege set, and the effective privilege set is cleared.

If the process calling `setgid()` has the `PRIV_PROC_SETID` privilege, the real, effective, and saved group IDs are set to the *gid* parameter.

If the process calling `setgid()` does not have the `PRIV_PROC_SETID` privilege, but *gid* is either the real group ID or the saved group ID of the calling process, the effective group ID is set to *gid*.

RETURN VALUES

`setuid()` returns:

- 0 On success.
- 1 On failure, and sets `errno` to indicate the error.

ERRORS

The `setuid()` and `setgid()` functions will fail if:

- `EINVAL` The value of *uid* or *gid* is out of range.
- `EPERM` For `setuid()` and `seteuid()`, the calling process does not have `PRIV_PROC_SETID` in its effective set of privileges, and the *uid* parameter does not match either the real or saved user IDs.

For `setgid()` and `setegid()`, the calling process does not have `PRIV_PROC_SETID` in its effective set of privileges, and the *gid* parameter does not match either the real or the saved group ID.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	<code>setuid()</code> and <code>setgid()</code> are Async-Signal-Safe

SUMMARY OF TRUSTED SOLARIS CHANGES Trusted Solaris 8 4/01 Reference Manual SunOS 5.9 Reference Manual

The Trusted Solaris environment replaces the checks of super-user in the Solaris environment with a check for `PRIV_PROC_SETID`.

`intro(2)`, `exec(2)`, `getgroups(2)`
`getuid(2)`, `attributes(5)`, `stat(5)`

NAME	setuid, setegid, seteuid, setgid – set user and group IDs
SYNOPSIS	<pre> #include <sys/types.h> #include <unistd.h> int setuid(uid_t uid) ; int setegid(gid_t egid) ; int seteuid(uid_t euid) ; int setgid(gid_t gid) ; </pre>
DESCRIPTION	<p>The <code>setuid()</code> function sets the real user ID, effective user ID, and saved user ID of the calling process. The <code>setgid()</code> function sets the real group ID, effective group ID, and saved group ID of the calling process. The <code>setegid()</code> and <code>seteuid()</code> functions set the effective group and user IDs respectively for the calling process. See <code>intro(2)</code> for more information on real, effective, and saved user and group IDs.</p> <p>Privilege-unaware programs change their UIDs either to gain or give up rights associated with the new UID. To simulate that action in a privilege-based system rather than a UID-based system, the effective and saved privilege sets are modified across <code>setuid</code> calls according to the following algorithm:</p> <pre> if ((old_uid == start_uid) && (new_uid != start_uid)) { saved_privileges = effective_privileges; effective_privileges = empty; } else if ((old_uid != start_uid) && (new_uid == start_uid)) { effective_privileges = saved_privileges; } </pre> <p>Since set UID programs may not be aware of privileges, their privilege bracketing (their use of UID changes to obtain or give up rights) is tracked in the privilege sets. When a set UID program changes from its saved UID ID to the calling user ID, it gives up its privilege. When it changes back to the saved UID ID, it regains privilege.</p> <p>At login time, the real user ID, effective user ID, and saved user ID of the login process are set to the login ID of the user responsible for the creation of the process. The same is true for the real, effective, and saved group IDs; they are set to the group ID of the user responsible for the creation of the process.</p> <p>When a process calls one of the <code>exec</code> family of functions (see <code>exec(2)</code>) to execute a file (program), the user and/or group identifiers associated with the process can change. If the file executed is a set-user-ID file, the effective and saved user IDs of the process are set to the owner of the file executed. If the file executed is a set-group-ID file, the effective and saved group IDs of the process are set to the group of the file executed. If the file executed is not a set-user-ID or set-group-ID file, the effective user ID, saved user ID, effective group ID, and saved group ID are not changed.</p> <p>If the process calling <code>setuid()</code> has the <code>PRIV_PROC_SETID</code> privilege, the real, effective, and saved user IDs are set to the <code>uid</code> parameter.</p>

seteuid(2)

If the process calling `setuid()` does not have the `PRIV_PROC_SETID` privilege, but *uid* is either the real user ID or the saved user ID of the calling process, the effective user ID is set to *uid*.

If the new user ID differs from the initial user ID under which this program began execution, the saved privilege set is replaced by the effective privilege set, and the effective privilege set is cleared.

If the process calling `setgid()` has the `PRIV_PROC_SETID` privilege, the real, effective, and saved group IDs are set to the *gid* parameter.

If the process calling `setgid()` does not have the `PRIV_PROC_SETID` privilege, but *gid* is either the real group ID or the saved group ID of the calling process, the effective group ID is set to *gid*.

RETURN VALUES

`setuid()` returns:

- 0 On success.
- 1 On failure, and sets `errno` to indicate the error.

ERRORS

The `setuid()` and `setgid()` functions will fail if:

- `EINVAL` The value of *uid* or *gid* is out of range.
- `EPERM` For `setuid()` and `seteuid()`, the calling process does not have `PRIV_PROC_SETID` in its effective set of privileges, and the *uid* parameter does not match either the real or saved user IDs.

For `setgid()` and `setegid()`, the calling process does not have `PRIV_PROC_SETID` in its effective set of privileges, and the *gid* parameter does not match either the real or the saved group ID.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	<code>setuid()</code> and <code>setgid()</code> are Async-Signal-Safe

SUMMARY OF TRUSTED SOLARIS CHANGES Trusted Solaris 8 4/01 Reference Manual SunOS 5.8 Reference Manual

The Trusted Solaris environment replaces the checks of super-user in the Solaris environment with a check for `PRIV_PROC_SETID`.

`intro(2)`, `exec(2)`, `getgroups(2)`
`getuid(2)`, `attributes(5)`, `stat(5)`

NAME	getfattrflag, fsetfattrflag, fgetfattrflag, setfattrflag, mldgetfattrflag, mldsetfattrflag – set/get the security attribute flags of a file
SYNOPSIS	<pre>cc [flags...] file... -ltso1 #include <tsol/secflgs.h> int getfattrflag(const char *path, secflgs_t *flags); int setfattrflag(const char *path, secflgs_t which, secflgs_t flags); int fgetfattrflag(int fildes, secflgs_t *flags); int fsetfattrflag(int fildes, secflgs_t which, secflgs_t flags); int mldgetfattrflag(const char *path, secflgs_t *flags); int mldsetfattrflag(const char *path, secflgs_t which, secflgs_t flags);</pre>
DESCRIPTION	<p>setfattrflag(), fsetfattrflag(), and mldsetfattrflag() set the security flags of the file whose name is given by <i>path</i> or referred to by the open file descriptor <i>fildes</i>. The bit pattern contained in <i>which</i> is used to indicate which flags are being affected. The corresponding bits in <i>flags</i> are set to 1 or 0 to indicate whether the affected flags are being set or unset respectively.</p> <p>getfattrflag(), fgetfattrflag(), and mldgetfattrflag() get the security flags of the file whose name is given by <i>path</i> or referred to by the open file descriptor <i>fildes</i> and store it in the location pointed to by <i>flags</i>.</p> <p>Attribute bits are interpreted as follows:</p> <p>FAF_MLD Directory has MLD semantics.</p> <p>FAF_PUBLIC Filesystem object is a public object.</p> <p>FAF_SLD Directory is an SLD.</p> <p>Attribute flags are constructed by OR'ing the attribute flag bits.</p> <p>FAF_MLD is the only flag that may be modified without privilege if the directory is empty, the effective user ID of the process matches the directory owner, and the process has mandatory as well as discretionary write access. The FAF_MLD flag, once set, cannot be unset. Additionally, the FAF_MLD flag may only be set via the mldsetfattrflag interface. The FAF_PUBLIC flag can only be read or modified by a process possessing the PRIV_FILE_AUDIT privilege. A process attempting to read the FAF_PUBLIC flag without the PRIV_FILE_AUDIT privilege in effect will not fail. However the value of FAF_PUBLIC will be returned as unset. The FAF_SLD flag can never be set. The ability to read any flag is dependant upon the process having mandatory and discretionary read access to the file. The ability to set any flag is dependant upon the process having mandatory and discretionary write access to the file.</p>

setfattrflag(2)

If *path* is a symbolic link, the target's attribute flags are affected rather than the link's. If *path* is a multilevel directory, `getfattrflag()` and `setfattrflag()` will affect the underlying single-level directory beneath (unless *path* is adorned). `mldgetfattrflag()` and `mldsetfattrflag()` do not translate multi-level directories to underlying single-level directories. `fgetfattrflag()` and `fsetfattrflag()` affect only the file referred to by *fildev*.

RETURN VALUES

These functions return:

- 0 On success.
- 1 On failure, and set `errno` to indicate the error.

ERRORS

`getfattrflag()` and `mldgetfattrflag()` will fail if one or more of the following are true:

- EACCES Search permission is denied on a component of the path prefix of *path*. To override this restriction, the calling process may assert the `PRIV_FILE_DAC_SEARCH` privilege and/or the `PRIV_FILE_MAC_SEARCH` privilege.
 - EACCES Read permission is denied the final component of *path*. To override this restriction, the calling process may assert the `PRIV_FILE_MAC_READ` privilege.
 - EFAULT *path* points to an illegal address.
 - EINTR A signal was caught during execution of the function.
 - EIO An I/O error occurred while reading from the file system.
 - ELOOP Too many symbolic links were encountered in translating *path*.
 - EMULTIHOP Components of *path* require hopping to multiple remote machines and file system type does not allow it.
 - ENAMETOOLONG The length of the *path* argument exceeds `PATH_MAX`, or the length of a *path* component exceeds `NAME_MAX` while `_POSIX_NO_TRUNC` is in effect.
 - ENOENT Either a component of the path prefix, or the file referred to by *path* does not exist or is a null pathname.
 - ENOLINK *fildev* points to a remote machine and the link to that machine is no longer active.
 - ENOTDIR A component of the prefix of *path* is not a directory.
- `fgetfattrflag()` fails and the file mode is unchanged if:
- EACCES Read permission is denied on *fildev*. To override this restriction, the calling process may assert the `PRIV_FILE_MAC_READ` privilege.
 - EBADF *fildev* is not an open file descriptor.

EIO	An I/O error occurred while reading from the file system.
EINTR	A signal was caught during execution of the <code>fgetfattrflag()</code> function.
<code>setfattrflag()</code> and <code>mlsetfattrflag()</code> will fail and the file mode is unchanged if one or more of the following are true:	
EACCES	Search permission is denied on a component of the path prefix of <i>path</i> . To override this restriction, the calling process may assert the <code>PRIV_FILE_DAC_SEARCH</code> privilege and/or the <code>PRIV_FILE_MAC_SEARCH</code> privilege.
EACCES	Write permission is denied <i>path</i> . To override this restriction, the calling process may assert the <code>PRIV_FILE_MAC_WRITE</code> privilege.
EACCES	The calling process does not own <i>files</i> . To override this restriction, the calling process may assert the <code>PRIV_FILE_OWNER</code> privilege.
EFAULT	<i>path</i> points to an illegal address.
EINTR	A signal was caught during execution of the function.
EINVAL	<i>path</i> is not a valid pathname. When setting <code>FAF_MLD</code> , <i>path</i> must refer to an empty directory.
EIO	An I/O error occurred while writing to the file system.
ELOOP	Too many symbolic links were encountered in translating <i>path</i> .
EMULTIHOP	Components of <i>path</i> require hopping to multiple remote machines and filesystem type does not allow it.
ENAMETOOLONG	The length of the <i>path</i> argument exceeds <code>PATH_MAX</code> , or the length of a <i>path</i> component exceeds <code>NAME_MAX</code> while <code>_POSIX_NO_TRUNC</code> is in effect.
ENOENT	Either a component of the path prefix, or the file referred to by <i>path</i> does not exist or is a null pathname.
ENOLINK	<i>path</i> points to a remote machine and the link to that machine is no longer active.
ENOTDIR	A component of the prefix of <i>path</i> is not a directory.
EPERM	The effective user ID does not match the owner of the file and the process does not possess the privilege <code>PRIV_FILE_OWNER</code> .
EPERM	The process does not possess the privilege <code>PRIV_FILE_AUDIT</code> and is attempting to set the <code>FAF_PUBLIC</code> flag.
EROFS	The file referred to by <i>path</i> resides on a read-only file system.
<code>fsetfattrflag()</code> fails and the file mode is unchanged if:	

setfattrflag(2)

EACCES	The calling process does not own <i>filde</i> s. To override this restriction, the calling process may assert the PRIV_FILE_OWNER privilege.
EACCES	Write access is denied on <i>filde</i> s. To override this restriction, the calling process may assert the PRIV_FILE_MAC_WRITE privilege.
EINVAL	<i>filde</i> s is not a valid pathname. When setting FAF_MLD, <i>filde</i> s must refer to an empty directory.
EBADF	<i>filde</i> s is not an open file descriptor.
EIO	An I/O error occurred while writing to the file system.
EINTR	A signal was caught during execution of the fsetfattrflag() function.
EPERM	The process does not possess the privilege PRIV_FILE_AUDIT and is attempting to set the FAF_PUBLIC flag.
EROFS	The file referred to by <i>filde</i> s resides on a read-only file system.

**Trusted Solaris 8
4/01 Reference
Manual**

setfattrflag(1), getfattrflag(1)

Trusted Solaris Developer's Guide

	setfpriv(2)										
NAME	getfpriv, fgetfpriv, setfpriv, fsetfpriv – return or set a privilege set associated with a file										
SYNOPSIS	<pre>cc [flags...] file... -ltsol int getfpriv(char *path, priv_ftype_t type, priv_set_t *priv_set); int setfpriv(char *path, priv_op_t op, priv_ftype_t type, priv_set_t *priv_set); int fgetfpriv(int fd, priv_ftype_t type, priv_set_t *priv_set); int fsetfpriv(int fd, priv_op_t op, priv_ftype_t type, priv_set_t *priv_set);</pre>										
DESCRIPTION	<p>Set or get privileges of the file that is named by <i>path</i> or referred to by <i>fd</i>. <i>fgetfpriv()</i> and <i>fsetfpriv()</i> function exactly like <i>getfpriv()</i> and <i>setfpriv()</i> respectively, except that they require an open reference to a file as their argument.</p> <p><i>getfpriv()</i> copies the privilege set indicated by <i>type</i> and associated with the named file into the address specified by <i>priv_set</i>. Values for <i>type</i> are:</p> <table> <tr> <td>PRIV_FORCED</td><td>The forced privilege set.</td></tr> <tr> <td>PRIV_ALLOWED</td><td>The allowed privilege set.</td></tr> </table> <p>MAC read permission is required for the named file unless the privilege PRIV_FILE_MAC_READ is effective.</p> <p><i>setfpriv()</i> sets/modifies the privilege set (the target set) indicated by <i>type</i> and associated with the named file. Modification occurs according to the value of <i>op</i> and the privilege set specified by <i>priv_set</i> (the specified set). Values for <i>op</i> are:</p> <table> <tr> <td>PRIV_ON</td><td>Each privilege asserted in the specified set is asserted in the target set.</td></tr> <tr> <td>PRIV_OFF</td><td>Each privilege asserted in the specified set is cleared in the target set.</td></tr> <tr> <td>PRIV_SET</td><td>The target set is set exactly equal to the specified set.</td></tr> </table> <p>Values for <i>type</i> are the same as those used for <i>getfpriv()</i>.</p> <p>In all cases, the privilege PRIV_FILE_SETPRIV must be effective. In addition, only the owner of a file may change its privilege sets, unless the privilege PRIV_FILE_OWNER is effective.</p> <p>The invoking process must have MAC write permission for the named file (unless the privilege PRIV_FILE_MAC_WRITE is effective). DAC write access is not required.</p> <p>It is an error to attempt to assert a forced privilege if the corresponding allowed privilege is not present. For this reason, it is recommended that the allowed privilege set be modified first whenever both privilege sets are to be modified.</p>	PRIV_FORCED	The forced privilege set.	PRIV_ALLOWED	The allowed privilege set.	PRIV_ON	Each privilege asserted in the specified set is asserted in the target set.	PRIV_OFF	Each privilege asserted in the specified set is cleared in the target set.	PRIV_SET	The target set is set exactly equal to the specified set.
PRIV_FORCED	The forced privilege set.										
PRIV_ALLOWED	The allowed privilege set.										
PRIV_ON	Each privilege asserted in the specified set is asserted in the target set.										
PRIV_OFF	Each privilege asserted in the specified set is cleared in the target set.										
PRIV_SET	The target set is set exactly equal to the specified set.										

setfpriv(2)

If the target set is the allowed set, all privileges cleared from the target set are also automatically cleared from the forced set.

Normally MAC read permission is required or the privilege `PRIV_FILE_MAC_READ` must be effective for `getfpriv()` to complete its operation successfully unless the named file is a pty pseudo-terminal. If the named file is a pseudo-terminal (`/dev/ptyp*` or `/dev/ttyp*`) and the label of the process invoking `getfpriv()` does not dominate the label of the named file and the privilege `PRIV_FILE_MAC_READ` is not effective then `getfpriv()` returns success but sets the privilege fields of *priv_set* to zero.

RETURN VALUES

These routines return:

- 0 On success.
- 1 On failure, and set `errno` to indicate the error.

ERRORS

These routines fail and the target set is not modified if:

- `EINVAL` An illegal or undefined value is supplied for *size* or *type*.
- `EFAULT` *priv_set* refers to an invalid address.

Additionally, `getfpriv()` and `setfpriv()` fail if:

- `EACCES` Search permission is denied a component of *path*. To override this restriction, the calling process may assert the `PRIV_FILE_DAC_SEARCH` privilege and/or the `PRIV_FILE_MAC_SEARCH` privilege.

`getfpriv()` and `fgetfpriv()` fail if:

- `EACCES` MAC read permission is denied for the named file, and privilege `PRIV_FILE_MAC_READ` is not effective.
- `ENOENT` A component of the specified path does not exist.
- `ENOTDIR` A component of the specified path prefix is not a directory.
- `ENAMETOOLONG` The length of the path argument exceeds `PATH_MAX`, or a pathname component is longer than `NAME_MAX` while `_POSIX_NO_TRUNC` is in effect.

`setfpriv()` and `fsetfpriv()` fail and the target set is not modified if:

- `EACCES` MAC write permission is denied for the named file, privilege `PRIV_FILE_MAC_WRITE` is not effective, and the user's clearance dominates the sensitivity label of the file.
- `EINVAL` (1) The named file resides on a file system that does not support privileges (that is, a file system other than NFS, TMPFS) or (2) an illegal or undefined value is supplied for *op*. Also if privilege `PRIV_FILE_MAC_WRITE` is not effective.

		setfpriv(2)
EPERM	MAC write permission is denied for the named file, and the user's clearance does not dominate the label of the named file, or (2) PRIV_FILE_SETPRIV is not effective, or (3) the effective uid does not match the owner of the named file and privilege PRIV_FILE_OWNER is not effective.	
EROFS	The named file resides on a read-only file system.	
	getppriv(2), setppriv(2), priv_macros(5)	

setgid(2)

NAME	setuid, setegid, seteuid, setgid – set user and group IDs
SYNOPSIS	<pre>#include <sys/types.h> #include <unistd.h> int setuid(uid_t uid) ; int setegid(gid_t egid) ; int seteuid(uid_t euid) ; int setgid(gid_t gid) ;</pre>
DESCRIPTION	<p>The <code>setuid()</code> function sets the real user ID, effective user ID, and saved user ID of the calling process. The <code>setgid()</code> function sets the real group ID, effective group ID, and saved group ID of the calling process. The <code>setegid()</code> and <code>seteuid()</code> functions set the effective group and user IDs respectively for the calling process. See <code>intro(2)</code> for more information on real, effective, and saved user and group IDs.</p> <p>Privilege-unaware programs change their UIDs either to gain or give up rights associated with the new UID. To simulate that action in a privilege-based system rather than a UID-based system, the effective and saved privilege sets are modified across <code>setuid</code> calls according to the following algorithm:</p> <pre>if ((old_uid == start_uid) && (new_uid != start_uid)) { saved_privileges = effective_privileges; effective_privileges = empty; } else if ((old_uid != start_uid) && (new_uid == start_uid)) { effective_privileges = saved_privileges; }</pre> <p>Since set UID programs may not be aware of privileges, their privilege bracketing (their use of UID changes to obtain or give up rights) is tracked in the privilege sets. When a set UID program changes from its saved UID ID to the calling user ID, it gives up its privilege. When it changes back to the saved UID ID, it regains privilege.</p> <p>At login time, the real user ID, effective user ID, and saved user ID of the login process are set to the login ID of the user responsible for the creation of the process. The same is true for the real, effective, and saved group IDs; they are set to the group ID of the user responsible for the creation of the process.</p> <p>When a process calls one of the <code>exec</code> family of functions (see <code>exec(2)</code>) to execute a file (program), the user and/or group identifiers associated with the process can change. If the file executed is a set-user-ID file, the effective and saved user IDs of the process are set to the owner of the file executed. If the file executed is a set-group-ID file, the effective and saved group IDs of the process are set to the group of the file executed. If the file executed is not a set-user-ID or set-group-ID file, the effective user ID, saved user ID, effective group ID, and saved group ID are not changed.</p> <p>If the process calling <code>setuid()</code> has the <code>PRIV_PROC_SETID</code> privilege, the real, effective, and saved user IDs are set to the <code>uid</code> parameter.</p>

	setgid(2)				
	<p>If the process calling <code>setuid()</code> does not have the <code>PRIV_PROC_SETID</code> privilege, but <i>uid</i> is either the real user ID or the saved user ID of the calling process, the effective user ID is set to <i>uid</i>.</p> <p>If the new user ID differs from the initial user ID under which this program began execution, the saved privilege set is replaced by the effective privilege set, and the effective privilege set is cleared.</p> <p>If the process calling <code>setgid()</code> has the <code>PRIV_PROC_SETID</code> privilege, the real, effective, and saved group IDs are set to the <i>gid</i> parameter.</p> <p>If the process calling <code>setgid()</code> does not have the <code>PRIV_PROC_SETID</code> privilege, but <i>gid</i> is either the real group ID or the saved group ID of the calling process, the effective group ID is set to <i>gid</i>.</p>				
RETURN VALUES	<p><code>setuid()</code> returns:</p> <p>0 On success.</p> <p>-1 On failure, and sets <code>errno</code> to indicate the error.</p>				
ERRORS	<p>The <code>setuid()</code> and <code>setgid()</code> functions will fail if:</p> <table> <tr> <td>EINVAL</td><td>The value of <i>uid</i> or <i>gid</i> is out of range.</td></tr> <tr> <td>EPERM</td><td> <p>For <code>setuid()</code> and <code>seteuid()</code>, the calling process does not have <code>PRIV_PROC_SETID</code> in its effective set of privileges, and the <i>uid</i> parameter does not match either the real or saved user IDs.</p> <p>For <code>setgid()</code> and <code>setegid()</code>, the calling process does not have <code>PRIV_PROC_SETID</code> in its effective set of privileges, and the <i>gid</i> parameter does not match either the real or the saved group ID.</p> </td></tr> </table>	EINVAL	The value of <i>uid</i> or <i>gid</i> is out of range.	EPERM	<p>For <code>setuid()</code> and <code>seteuid()</code>, the calling process does not have <code>PRIV_PROC_SETID</code> in its effective set of privileges, and the <i>uid</i> parameter does not match either the real or saved user IDs.</p> <p>For <code>setgid()</code> and <code>setegid()</code>, the calling process does not have <code>PRIV_PROC_SETID</code> in its effective set of privileges, and the <i>gid</i> parameter does not match either the real or the saved group ID.</p>
EINVAL	The value of <i>uid</i> or <i>gid</i> is out of range.				
EPERM	<p>For <code>setuid()</code> and <code>seteuid()</code>, the calling process does not have <code>PRIV_PROC_SETID</code> in its effective set of privileges, and the <i>uid</i> parameter does not match either the real or saved user IDs.</p> <p>For <code>setgid()</code> and <code>setegid()</code>, the calling process does not have <code>PRIV_PROC_SETID</code> in its effective set of privileges, and the <i>gid</i> parameter does not match either the real or the saved group ID.</p>				
ATTRIBUTES	<p>See <code>attributes(5)</code> for descriptions of the following attributes:</p> <table> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> <tr> <td>MT-Level</td><td><code>setuid()</code> and <code>setgid()</code> are Async-Signal-Safe</td></tr> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	<code>setuid()</code> and <code>setgid()</code> are Async-Signal-Safe
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
MT-Level	<code>setuid()</code> and <code>setgid()</code> are Async-Signal-Safe				
SUMMARY OF TRUSTED SOLARIS CHANGES Trusted Solaris 8 4/01 Reference Manual	<p>The Trusted Solaris environment replaces the checks of super-user in the Solaris environment with a check for <code>PRIV_PROC_SETID</code>.</p> <p><code>intro(2)</code>, <code>exec(2)</code>, <code>getgroups(2)</code></p> <p><code>getuid(2)</code>, <code>attributes(5)</code>, <code>stat(5)</code></p>				

setgroups(2)

NAME	getgroups, setgroups – Get or set supplementary group access list IDs								
SYNOPSIS	<pre>#include <unistd.h> int getgroups(int <i>gidsetsize</i>, gid_t *<i>grouplist</i>); int setgroups(int <i>ngroups</i>, const gid_t *<i>grouplist</i>);</pre>								
DESCRIPTION	<p>The <code>getgroups()</code> function gets the current supplemental group access list of the calling process and stores the result in the array of group IDs specified by <i>grouplist</i>. This array has <i>gidsetsize</i> entries and must be large enough to contain the entire list. This list cannot be larger than <code>NGROUPS_MAX</code>. If <i>gidsetsize</i> equals 0, <code>getgroups()</code> will return the number of groups to which the calling process belongs without modifying the array pointed to by <i>grouplist</i>.</p> <p>The <code>setgroups()</code> function sets the supplementary group access list of the calling process from the array of group IDs specified by <i>grouplist</i>. The number of entries is specified by <i>ngroups</i> and can not be greater than <code>NGROUPS_MAX</code>. The calling process must have <code>PRIV_PROC_SETID</code> in its set of effective privileges to set new groups. If <code>PRIV_PROC_SETID</code> is not in the effective privilege set, the operation fails and sets <code>errno</code> to <code>EPERM</code>.</p>								
RETURN VALUES	Upon successful completion, <code>getgroups()</code> returns the number of supplementary group IDs set for the calling process and <code>setgroups()</code> returns 0. Otherwise, <code>-1</code> is returned and <code>errno</code> is set to indicate the error.								
ERRORS	<p>The <code>getgroups()</code> and <code>setgroups()</code> functions will fail if:</p> <table><tr><td>EFAULT</td><td>A referenced part of the array pointed to by <i>grouplist</i> is an illegal address.</td></tr></table> <p>The <code>getgroups()</code> function will fail if:</p> <table><tr><td>EINVAL</td><td>The value of <i>gidsetsize</i> is non-zero and less than the number of supplementary group IDs set for the calling process.</td></tr></table> <p>The <code>setgroups()</code> function will fail if:</p> <table><tr><td>EINVAL</td><td>The value of <i>ngroups</i> is greater than <code>NGROUPS_MAX</code>.</td></tr><tr><td>EPERM</td><td>The calling process does not have the <code>PRIV_PROC_SETID</code> privilege.</td></tr></table>	EFAULT	A referenced part of the array pointed to by <i>grouplist</i> is an illegal address.	EINVAL	The value of <i>gidsetsize</i> is non-zero and less than the number of supplementary group IDs set for the calling process.	EINVAL	The value of <i>ngroups</i> is greater than <code>NGROUPS_MAX</code> .	EPERM	The calling process does not have the <code>PRIV_PROC_SETID</code> privilege.
EFAULT	A referenced part of the array pointed to by <i>grouplist</i> is an illegal address.								
EINVAL	The value of <i>gidsetsize</i> is non-zero and less than the number of supplementary group IDs set for the calling process.								
EINVAL	The value of <i>ngroups</i> is greater than <code>NGROUPS_MAX</code> .								
EPERM	The calling process does not have the <code>PRIV_PROC_SETID</code> privilege.								
ATTRIBUTES	<p>See <code>attributes(5)</code> for descriptions of the following attributes:</p> <table><thead><tr><th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr></thead><tbody><tr><td>MT-Level</td><td>Async-Signal-Safe</td></tr></tbody></table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	MT-Level	Async-Signal-Safe				
ATTRIBUTE TYPE	ATTRIBUTE VALUE								
MT-Level	Async-Signal-Safe								
SUMMARY OF TRUSTED SOLARIS CHANGES	To set new groups, the calling process must have <code>PRIV_PROC_SETID</code> in its set of effective privileges.								

setgroups(2)

chown(2), setuid(2)

groups(1), getuid(2), getgrnam(3C), initgroups(3C), attributes(5)

setpatrr(2)

NAME	getpatrr, setpatrr – get/set process attribute flags																
SYNOPSIS	<pre>cc [flags...] file... -ltsol #include <tsol/patrr.h> int getpatrr(patrr_type_t type, patrr_flag_t *value); int setpatrr(patrr_type_t type, patrr_flag_t value);</pre>																
DESCRIPTION	<p>Process attribute flags are a set of flags that describe additional attributes that the process has. Each flag in the set is separately addressable although all flags share the <code>getpatrr()</code> and the <code>setpatrr()</code> system call interfaces. Likewise, each flag in the set has its own protection policy although all flags use the same protection mechanism. In the set are seven types of flags, specified in <code><tsol/patrr.h></code>, and addressed by the <i>type</i> argument. These are the values for <i>type</i>:</p> <table> <tr> <td>PAF_TRUSTED_PATH</td><td>Trusted path flag</td></tr> <tr> <td>PAF_PRIV_DBG</td><td>Privilege debugging flag</td></tr> <tr> <td>PAF_TOKMAPPER</td><td>Network token mapping process flag</td></tr> <tr> <td>PAF_LABEL_VIEW</td><td>Label view flags</td></tr> <tr> <td>PAF_LABEL_XLATE</td><td>Label translation flags</td></tr> <tr> <td>PAF_DISKLESS_BOOT</td><td>Part of diskless boot flag</td></tr> <tr> <td>PAF_SELAGNT</td><td>Part of selection agent flag</td></tr> <tr> <td>PAF_PRINT_SYSTEM</td><td>Part of trusted printing system flag</td></tr> </table> <p>A description of each type of process attribute flag follows.</p> <p>Trusted path flag This one-bit flag marks a trusted path process. This flag can be viewed and cleared, but cannot be set. In other words, the call to <code>setpatrr(PAF_TRUSTED_PATH, PAF_TP_ON)</code> will always fail. A process inherits the trusted path flag from its parent process. The <code>init</code> process receives the trusted path flag from the system. A user session creator, such as <code>login</code>, clears this flag before starting a user session.</p> <pre>setpatrr(PAF_TRUSTED_PATH, PAF_TP_OFF)</pre> <p>Privilege debugging flag This one-bit flag indicates that the process is in privilege-debugging mode—a process-operation mode in which privilege requirement is logged but not enforced. This flag can be viewed or cleared, but cannot be set except by a trusted path process.</p> <p>Network token mapping process flag This one-bit flag, when set, identifies the process as the network token mapping process. The network token mapping process is exempt from network token mapping. This flag can be viewed and cleared, but cannot be set except by a trusted path process.</p>	PAF_TRUSTED_PATH	Trusted path flag	PAF_PRIV_DBG	Privilege debugging flag	PAF_TOKMAPPER	Network token mapping process flag	PAF_LABEL_VIEW	Label view flags	PAF_LABEL_XLATE	Label translation flags	PAF_DISKLESS_BOOT	Part of diskless boot flag	PAF_SELAGNT	Part of selection agent flag	PAF_PRINT_SYSTEM	Part of trusted printing system flag
PAF_TRUSTED_PATH	Trusted path flag																
PAF_PRIV_DBG	Privilege debugging flag																
PAF_TOKMAPPER	Network token mapping process flag																
PAF_LABEL_VIEW	Label view flags																
PAF_LABEL_XLATE	Label translation flags																
PAF_DISKLESS_BOOT	Part of diskless boot flag																
PAF_SELAGNT	Part of selection agent flag																
PAF_PRINT_SYSTEM	Part of trusted printing system flag																

	setpattr(2)
Label view flags	These two-bit flags support per-process label translation. These flags are viewable and modifiable without restriction.
Label translation flags	These fifteen-bit flags support the GFI <code>FLAGS=</code> option in the <code>label_encodings(4)</code> file. Only a trusted path process can view or modify these flags.
Part of diskless boot flag	This one-bit flag identifies the process as taking part in diskless booting. This flag can be viewed and cleared, but cannot be set except by a trusted path process.
Part of selection agent flag	This one-bit flag identifies the process as part of the “cut and paste” selection agent. This flag can be viewed and cleared, but cannot be set except by a trusted path process.
Part of trusted printing system flag	<p>This one-bit flag identifies the process as a member of the Trusted Printing System. This flag can be viewed and cleared, but cannot be set except by a trusted path process.</p> <p>In short, these flag-related protection policies apply. Any process may view or clear any process attribute flag except the label translation flags; viewing or clearing the label translation flags requires that a process have the trusted path attribute. Any process may set label view flags; setting other flags requires that the setting process have the trusted path attribute.</p> <p><code>getpattr()</code> copies the <i>type</i> process flag of the calling process into the <i>pattr_flag_t</i> variable addressed by <i>value</i>. Only the lower <i>n</i> bits are copied, where <i>n</i> is the width of the flag. The higher bits are cleared.</p> <p><code>setpattr()</code> copies the lower <i>n</i> bits of <i>value</i> to the <i>type</i> process flag of the calling process, where <i>n</i> is the width of the selected process flag.</p>
RETURN VALUES	<p><code>getpattr()</code> and <code>setpattr()</code> return:</p> <p>0 On success.</p> <p>−1 On failure, and sets <code>errno</code> to indicate the error.</p>
ERRORS	<p><code>getpattr()</code> may fail for one of these reasons:</p> <p><code>EFAULT</code> The <i>value</i> argument points to a bad address.</p> <p><code>EINVAL</code> The <i>type</i> argument is not one of the listed type constants.</p> <p><code>EACCES</code> The calling process is not a trusted path process as required to view the <i>type</i> flag.</p> <p><code>setpattr()</code> may fail for one of these reasons:</p> <p><code>EFAULT</code> The <i>value</i> argument points to a bad address.</p> <p><code>EINVAL</code> The <i>type</i> argument is not one of the listed type constants.</p> <p><code>EACCES</code> The calling process is not a trusted path process as required to modify the <i>type</i> flag.</p>

setpattr(2)

**Trusted Solaris 8
4/01 Reference
Manual**

pattr(1)

NAME	getppriv, setppriv – Return or assign a privilege set associated with the invoking process														
SYNOPSIS	<pre>cc [flags...] file... -ltsol #include <tsol/priv.h> int getppriv(priv_ptype_ttype, priv_set_t*pset); int setppriv(priv_op_top, priv_ptype_ttype, priv_set_t*pset);</pre>														
DESCRIPTION	<p>getppriv() copies the <i>type</i> privilege set of the invoking process into the <i>pset</i> address. <i>type</i> may have one of four values, specified in <tsol/priv.h>:</p> <table> <tr> <td>PRIV_EFFECTIVE</td><td>The effective privilege set</td></tr> <tr> <td>PRIV_INHERITABLE</td><td>The inheritable privilege set</td></tr> <tr> <td>PRIV_PERMITTED</td><td>The permitted privilege set</td></tr> <tr> <td>PRIV_SAVED</td><td>The saved privilege set</td></tr> </table> <p>setppriv() assigns or modifies the <i>type</i> privilege set (the target set) of the invoking process. Modification occurs according to the values of <i>op</i> and of the <i>pset</i> privilege set (the source set). <i>op</i> values are specified in <tsol/priv.h>:</p> <table> <tr> <td>PRIV_ON</td><td>Each privilege asserted in the source set is asserted in the target set.</td></tr> <tr> <td>PRIV_OFF</td><td>Each privilege asserted in the source set is cleared in the target set.</td></tr> <tr> <td>PRIV_SET</td><td>The target set is made exactly equal to the source set.</td></tr> </table> <p>Values for <i>type</i> are the same as those for <i>type</i> in getppriv(), exclusive of PRIV_SAVED.</p> <p>If the target set is the permitted set, all privileges cleared from the target set are also cleared from the effective set. Any attempted assignment of a privilege cleared in the permitted set is always an error. Attempting to clear a privilege that is already cleared is not an error.</p>	PRIV_EFFECTIVE	The effective privilege set	PRIV_INHERITABLE	The inheritable privilege set	PRIV_PERMITTED	The permitted privilege set	PRIV_SAVED	The saved privilege set	PRIV_ON	Each privilege asserted in the source set is asserted in the target set.	PRIV_OFF	Each privilege asserted in the source set is cleared in the target set.	PRIV_SET	The target set is made exactly equal to the source set.
PRIV_EFFECTIVE	The effective privilege set														
PRIV_INHERITABLE	The inheritable privilege set														
PRIV_PERMITTED	The permitted privilege set														
PRIV_SAVED	The saved privilege set														
PRIV_ON	Each privilege asserted in the source set is asserted in the target set.														
PRIV_OFF	Each privilege asserted in the source set is cleared in the target set.														
PRIV_SET	The target set is made exactly equal to the source set.														
RETURN VALUES	<p>getppriv() and setppriv() return:</p> <table> <tr> <td>0</td><td>On success.</td></tr> <tr> <td>-1</td><td>On failure, and set errno to indicate the error.</td></tr> </table>	0	On success.	-1	On failure, and set errno to indicate the error.										
0	On success.														
-1	On failure, and set errno to indicate the error.														
ERRORS	<p>getppriv() fails if either of these conditions prevails:</p> <table> <tr> <td>EINVAL</td><td>An illegal or undefined value was supplied for <i>type</i>.</td></tr> <tr> <td>EFAULT</td><td><i>pset</i> refers to an invalid address.</td></tr> </table> <p>setppriv() fails and the target set is not modified if any of these conditions prevails:</p> <table> <tr> <td>EINVAL</td><td>An illegal or undefined value is supplied for <i>type</i> or <i>op</i>.</td></tr> </table>	EINVAL	An illegal or undefined value was supplied for <i>type</i> .	EFAULT	<i>pset</i> refers to an invalid address.	EINVAL	An illegal or undefined value is supplied for <i>type</i> or <i>op</i> .								
EINVAL	An illegal or undefined value was supplied for <i>type</i> .														
EFAULT	<i>pset</i> refers to an invalid address.														
EINVAL	An illegal or undefined value is supplied for <i>type</i> or <i>op</i> .														

setppriv(2)

EFAULT	set refers to an invalid address.
EINVAL	In a process privilege set, an attempt is made to assert a privilege that is cleared in the permitted set of the process.

getfpriv(2), setfpriv(2), priv_to_str(3TSOL), priv_set_to_str(3TSOL), str_to_priv(3TSOL), str_to_priv_set(3TSOL), priv_macros(5)

**Trusted Solaris 8
4/01 Reference
Manual**

NAME	setregid – Set real and effective group IDs				
SYNOPSIS	<pre>#include <unistd.h> int setregid(gid_t rgid, gid_t egid);</pre>				
DESCRIPTION	<p>The <code>setregid()</code> function is used to set the real and effective group IDs of the calling process. If <i>rgid</i> is <code>-1</code>, the real group ID is not changed; if <i>egid</i> is <code>-1</code>, the effective group ID is not changed. The real and effective group IDs may be set to different values in the same call.</p> <p>If the calling process has the <code>PRIV_PROC_SETID</code> privilege, the real GID and the effective GID can be set to any legal value.</p> <p>If the calling process does not have the <code>PRIV_PROC_SETID</code> privilege, either the real GID can be set to the saved setGID from <code>execve(2)</code>, or the effective GID can either be set to the saved setGID or the real GID. Note: if a setGID process sets its effective GID to its real GID, it can still set its effective GID back to the saved setGID.</p> <p>In either case, if the real group ID is being changed (that is, if <i>rgid</i> is not <code>-1</code>), or the effective group ID is being changed to a value not equal to the real group ID, the saved set-group-ID is set equal to the new effective group ID.</p>				
RETURN VALUES	<p><code>setregid()</code> returns:</p> <p>0 On success.</p> <p>-1 On failure and sets <code>errno</code> to indicate the error.</p>				
ERRORS	<p>The <code>setregid()</code> function will fail if:</p> <table> <tr> <td><code>EINVAL</code></td><td>The value of <i>rgid</i> or <i>egid</i> is less than 0 or greater than <code>UID_MAX</code> (defined in <code><limits.h></code>).</td></tr> <tr> <td><code>EPERM</code></td><td>The calling process does not have the <code>PRIV_PROC_SETID</code> privilege and a change other than changing the real GID to the saved setGID, or changing the effective GID to the real GID or the saved GID, was specified.</td></tr> </table>	<code>EINVAL</code>	The value of <i>rgid</i> or <i>egid</i> is less than 0 or greater than <code>UID_MAX</code> (defined in <code><limits.h></code>).	<code>EPERM</code>	The calling process does not have the <code>PRIV_PROC_SETID</code> privilege and a change other than changing the real GID to the saved setGID, or changing the effective GID to the real GID or the saved GID, was specified.
<code>EINVAL</code>	The value of <i>rgid</i> or <i>egid</i> is less than 0 or greater than <code>UID_MAX</code> (defined in <code><limits.h></code>).				
<code>EPERM</code>	The calling process does not have the <code>PRIV_PROC_SETID</code> privilege and a change other than changing the real GID to the saved setGID, or changing the effective GID to the real GID or the saved GID, was specified.				
USAGE	If a set-group-ID process sets its effective group ID to its real group ID, it can still set its effective group ID back to the saved set-group- ID.				
SUMMARY OF TRUSTED SOLARIS CHANGES	<p>The Trusted Solaris environment replaces the checks of super-user in the Solaris environment with privilege checks.</p> <p><code>execve(2)</code>, <code>setreuid(2)</code>, <code>setuid(2)</code></p> <p><code>getgid(2)</code></p>				

setreuid(2)

NAME	setreuid – Set real and effective user IDs				
SYNOPSIS	<pre>#include <unistd.h> int setreuid(uid_t <i>ruid</i>, uid_t <i>euid</i>);</pre>				
DESCRIPTION	<p>The <code>setreuid()</code> function is used to set the real and effective user IDs of the calling process. If <i>ruid</i> is <code>-1</code>, the real user ID is not changed; if <i>euid</i> is <code>-1</code>, the effective user ID is not changed. The real and effective user IDs may be set to different values in the same call.</p> <p>If the calling process has the <code>PRIV_PROC_SETID</code> privilege, the real user ID and the effective user ID can be set to any legal value.</p> <p>If the calling process does not have the <code>PRIV_PROC_SETID</code> privilege, either the real user ID can be set to the effective user ID, or the effective user ID can either be set to the saved set-user ID from <code>execve(2)</code> or the real user ID. Note: if a set-UID process sets its effective user ID to its real user ID, it can still set its effective user ID back to the saved set-user ID.</p> <p>In either case, if the real user ID is being changed (that is, if <i>ruid</i> is not <code>-1</code>), or the effective user ID is being changed to a value not equal to the real user ID, the saved set-user ID is set equal to the new effective user ID.</p>				
RETURN VALUES	<p><code>setreuid()</code> returns:</p> <p>0 On success.</p> <p>-1 On failure and sets <code>errno</code> to indicate the error.</p>				
ERRORS	<p>The <code>setreuid()</code> function will fail if:</p> <table><tr><td>EINVAL</td><td>The value of <i>ruid</i> or <i>euid</i> is less than 0 or greater than <code>UID_MAX</code> (defined in <code><limits.h></code>).</td></tr><tr><td>EPERM</td><td>The calling process does not have the <code>PRIV_PROC_SETID</code> privilege and a change other than changing the real user ID to the effective user ID, or changing the effective user ID to the real user ID or the saved set-user ID, was specified.</td></tr></table>	EINVAL	The value of <i>ruid</i> or <i>euid</i> is less than 0 or greater than <code>UID_MAX</code> (defined in <code><limits.h></code>).	EPERM	The calling process does not have the <code>PRIV_PROC_SETID</code> privilege and a change other than changing the real user ID to the effective user ID, or changing the effective user ID to the real user ID or the saved set-user ID, was specified.
EINVAL	The value of <i>ruid</i> or <i>euid</i> is less than 0 or greater than <code>UID_MAX</code> (defined in <code><limits.h></code>).				
EPERM	The calling process does not have the <code>PRIV_PROC_SETID</code> privilege and a change other than changing the real user ID to the effective user ID, or changing the effective user ID to the real user ID or the saved set-user ID, was specified.				
USAGE	If a set-user-ID process sets its effective user ID to its real user ID, it can still set its effective user ID back to the saved set-user ID.				
SUMMARY OF TRUSTED SOLARIS CHANGES Trusted Solaris 8 4/01 Reference Manual	<p>The Trusted Solaris environment replaces the checks of super-user in the Solaris environment with privilege checks.</p> <p><code>execve(2)</code>, <code>setregid(2)</code>, <code>setuid(2)</code></p> <p><code>getuid(2)</code></p>				

NAME	getrlimit, setrlimit – Control maximum system resource consumption								
SYNOPSIS	<pre>#include <sys/resource.h> int getrlimit(int <i>resource</i>, struct rlimit <i>*rlp</i>); int setrlimit(int <i>resource</i>, const struct rlimit <i>*rlp</i>);</pre>								
DESCRIPTION	<p>Limits on the consumption of a variety of system resources by a process and each process it creates may be obtained with the <code>getrlimit()</code> and set with <code>setrlimit()</code> functions.</p> <p>Each call to either <code>getrlimit()</code> or <code>setrlimit()</code> identifies a specific resource to be operated upon as well as a resource limit. A resource limit is a pair of values: one specifying the current (soft) limit, the other a maximum (hard) limit. Soft limits may be changed by a process to any value that is less than or equal to the hard limit. A process may (irreversibly) lower its hard limit to any value that is greater than or equal to the soft limit. Only a process that has the <code>PRIV_SYS_CONFIG</code> privilege can raise a hard limit. Both hard and soft limits can be changed in a single call to <code>setrlimit()</code> subject to the constraints described above. Limits may have an “infinite” value of <code>RLIM_INFINITY</code>. The <i>rlp</i> argument is a pointer to <code>struct rlimit</code> that includes the following members:</p> <pre>rlim_t rlim_cur; /* current (soft) limit */ rlim_t rlim_max; /* hard limit */</pre> <p>The type <code>rlim_t</code> is an arithmetic data type to which objects of type <code>int</code>, <code>size_t</code>, and <code>off_t</code> can be cast without loss of information.</p> <p>The possible resources, their descriptions, and the actions taken when the current limit is exceeded are summarized as follows:</p> <table> <tr> <td><code>RLIMIT_CORE</code></td><td>The maximum size of a core file in bytes that may be created by a process. A limit of 0 will prevent the creation of a core file. The writing of a core file will terminate at this size.</td></tr> <tr> <td><code>RLIMIT_CPU</code></td><td>The maximum amount of CPU time in seconds used by a process. This is a soft limit only. The <code>SIGXCPU</code> signal is sent to the process. If the process is holding or ignoring <code>SIGXCPU</code>, the behavior is scheduling class defined.</td></tr> <tr> <td><code>RLIMIT_DATA</code></td><td>The maximum size of a process’s heap in bytes. The <code>brk(2)</code> function will fail with <code>errno</code> set to <code>ENOMEM</code>.</td></tr> <tr> <td><code>RLIMIT_FSIZE</code></td><td>The maximum size of a file in bytes that may be created by a process. A limit of 0 will prevent the creation of a file. The <code>SIGXFSZ</code> signal is sent to the process. If the process is holding or ignoring <code>SIGXFSZ</code>, continued attempts to increase the size of a file beyond the limit will fail with <code>errno</code> set to <code>EFBIG</code>.</td></tr> </table>	<code>RLIMIT_CORE</code>	The maximum size of a core file in bytes that may be created by a process. A limit of 0 will prevent the creation of a core file. The writing of a core file will terminate at this size.	<code>RLIMIT_CPU</code>	The maximum amount of CPU time in seconds used by a process. This is a soft limit only. The <code>SIGXCPU</code> signal is sent to the process. If the process is holding or ignoring <code>SIGXCPU</code> , the behavior is scheduling class defined.	<code>RLIMIT_DATA</code>	The maximum size of a process’s heap in bytes. The <code>brk(2)</code> function will fail with <code>errno</code> set to <code>ENOMEM</code> .	<code>RLIMIT_FSIZE</code>	The maximum size of a file in bytes that may be created by a process. A limit of 0 will prevent the creation of a file. The <code>SIGXFSZ</code> signal is sent to the process. If the process is holding or ignoring <code>SIGXFSZ</code> , continued attempts to increase the size of a file beyond the limit will fail with <code>errno</code> set to <code>EFBIG</code> .
<code>RLIMIT_CORE</code>	The maximum size of a core file in bytes that may be created by a process. A limit of 0 will prevent the creation of a core file. The writing of a core file will terminate at this size.								
<code>RLIMIT_CPU</code>	The maximum amount of CPU time in seconds used by a process. This is a soft limit only. The <code>SIGXCPU</code> signal is sent to the process. If the process is holding or ignoring <code>SIGXCPU</code> , the behavior is scheduling class defined.								
<code>RLIMIT_DATA</code>	The maximum size of a process’s heap in bytes. The <code>brk(2)</code> function will fail with <code>errno</code> set to <code>ENOMEM</code> .								
<code>RLIMIT_FSIZE</code>	The maximum size of a file in bytes that may be created by a process. A limit of 0 will prevent the creation of a file. The <code>SIGXFSZ</code> signal is sent to the process. If the process is holding or ignoring <code>SIGXFSZ</code> , continued attempts to increase the size of a file beyond the limit will fail with <code>errno</code> set to <code>EFBIG</code> .								

setrlimit(2)

RLIMIT_NOFILE	One more than the maximum value that the system may assign to a newly created descriptor. This limit constrains the number of file descriptors that a process may create.
RLIMIT_STACK	<p>The maximum size of a process's stack in bytes. The system will not automatically grow the stack beyond this limit.</p> <p>Within a process, <code>setrlimit()</code> will increase the limit on the size of your stack, but will not move current memory segments to allow for that growth. To guarantee that the process stack can grow to the limit, the limit must be altered prior to the execution of the process in which the new stack size is to be used.</p> <p>Within a multithreaded process, <code>setrlimit()</code> has no impact on the stack size limit for the calling thread if the calling thread is not the main thread. A call to <code>setrlimit()</code> for <code>RLIMIT_STACK</code> impacts only the main thread's stack, and should be made only from the main thread, if at all.</p> <p>The <code>SIGSEGV</code> signal is sent to the process. If the process is holding or ignoring <code>SIGSEGV</code>, or is catching <code>SIGSEGV</code> and has not made arrangements to use an alternate stack (see <code>sigaltstack(2)</code>), the disposition of <code>SIGSEGV</code> will be set to <code>SIG_DFL</code> before it is sent.</p>
RLIMIT_VMEM	The maximum size of a process's mapped address space in bytes. If this limit is exceeded, the <code>brk(2)</code> and <code>mmap(2)</code> functions will fail with <code>errno</code> set to <code>ENOMEM</code> . In addition, the automatic stack growth will fail with the effects outlined above.
RLIMIT_AS	This is the maximum size of a process's total available memory, in bytes. If this limit is exceeded, the <code>brk(2)</code> , <code>malloc(3C)</code> , <code>mmap(2)</code> and <code>sbrk(2)</code> functions will fail with <code>errno</code> set to <code>ENOMEM</code> . In addition, the automatic stack growth will fail with the effects outlined above.

Because limit information is stored in the per-process information, the shell builtin `ulimit` command must directly execute this system call if it is to affect all future processes created by the shell.

The value of the current limit of the following resources affect these implementation defined parameters:

Limit	Implementation Defined Constant
RLIMIT_FSIZE	FCHR_MAX
RLIMIT_NOFILE	OPEN_MAX

When using the `getrlimit()` function, if a resource limit can be represented correctly in an object of type `rlim_t`, then its representation is returned; otherwise, if the value of the resource limit is equal to that of the corresponding saved hard limit, the value returned is `RLIM_SAVED_MAX`; otherwise the value returned is `RLIM_SAVED_CUR`.

When using the `setrlimit()` function, if the requested new limit is `RLIM_INFINITY`, the new limit will be "no limit"; otherwise if the requested new limit is `RLIM_SAVED_MAX`, the new limit will be the corresponding saved hard limit; otherwise, if the requested new limit is `RLIM_SAVED_CUR`, the new limit will be the corresponding saved soft limit; otherwise, the new limit will be the requested value. In addition, if the corresponding saved limit can be represented correctly in an object of type `rlim_t`, then it will be overwritten with the new limit.

The result of setting a limit to `RLIM_SAVED_MAX` or `RLIM_SAVED_CUR` is unspecified unless a previous call to `getrlimit()` returned that value as the soft or hard limit for the corresponding resource limit.

A limit whose value is greater than `RLIM_INFINITY` is permitted.

The `exec` family of functions also cause resource limits to be saved. See `exec(2)`.

RETURN VALUES

`getrlimit()` and `setrlimit()` return:

- 0 On success.
- 1 On failure, and set `errno` to indicate the error.

ERRORS

The `getrlimit()` and `setrlimit()` functions will fail if:

- `EFAULT` The *rlp* argument points to an illegal address.
- `EINVAL` An invalid *resource* was specified; or in a `setrlimit()` call, the new `rlim_cur` exceeds the new `rlim_max`.
- `EPERM` The limit specified to `setrlimit()` would have raised the maximum limit value, and the calling process does not have the `PRIV_SYS_CONFIG` privilege.

The `setrlimit()` function may fail if:

- `EINVAL` The limit specified cannot be lowered because current usage is already higher than the limit.

USAGE

The `getrlimit()` and `setrlimit()` functions have transitional interfaces for 64-bit file offsets. See `lf64(5)`.

SUMMARY OF TRUSTED SOLARIS CHANGES Trusted Solaris 8 4/01 Reference Manual

The calling process must have the `PRIV_SYS_CONFIG` privilege in order to increase a hard resource limit.

`open(2)`

setrlimit(2)

**SunOS 5.8
Reference Manual**

brk(2), sigaltstack(2), malloc(3C), signal(3C), signal(5)

NAME	setuid, setegid, seteuid, setgid – set user and group IDs
SYNOPSIS	<pre> #include <sys/types.h> #include <unistd.h> int setuid(uid_t uid) ; int setegid(gid_t egid) ; int seteuid(uid_t euid) ; int setgid(gid_t gid) ; </pre>
DESCRIPTION	<p>The <code>setuid()</code> function sets the real user ID, effective user ID, and saved user ID of the calling process. The <code>setgid()</code> function sets the real group ID, effective group ID, and saved group ID of the calling process. The <code>setegid()</code> and <code>seteuid()</code> functions set the effective group and user IDs respectively for the calling process. See <code>intro(2)</code> for more information on real, effective, and saved user and group IDs.</p> <p>Privilege-unaware programs change their UIDs either to gain or give up rights associated with the new UID. To simulate that action in a privilege-based system rather than a UID-based system, the effective and saved privilege sets are modified across <code>setuid</code> calls according to the following algorithm:</p> <pre> if ((old_uid == start_uid) && (new_uid != start_uid)) { saved_privileges = effective_privileges; effective_privileges = empty; } else if ((old_uid != start_uid) && (new_uid == start_uid)) { effective_privileges = saved_privileges; } </pre> <p>Since set UID programs may not be aware of privileges, their privilege bracketing (their use of UID changes to obtain or give up rights) is tracked in the privilege sets. When a set UID program changes from its saved UID ID to the calling user ID, it gives up its privilege. When it changes back to the saved UID ID, it regains privilege.</p> <p>At login time, the real user ID, effective user ID, and saved user ID of the login process are set to the login ID of the user responsible for the creation of the process. The same is true for the real, effective, and saved group IDs; they are set to the group ID of the user responsible for the creation of the process.</p> <p>When a process calls one of the <code>exec</code> family of functions (see <code>exec(2)</code>) to execute a file (program), the user and/or group identifiers associated with the process can change. If the file executed is a set-user-ID file, the effective and saved user IDs of the process are set to the owner of the file executed. If the file executed is a set-group-ID file, the effective and saved group IDs of the process are set to the group of the file executed. If the file executed is not a set-user-ID or set-group-ID file, the effective user ID, saved user ID, effective group ID, and saved group ID are not changed.</p> <p>If the process calling <code>setuid()</code> has the <code>PRIV_PROC_SETID</code> privilege, the real, effective, and saved user IDs are set to the <code>uid</code> parameter.</p>

setuid(2)

If the process calling `setuid()` does not have the `PRIV_PROC_SETID` privilege, but *uid* is either the real user ID or the saved user ID of the calling process, the effective user ID is set to *uid*.

If the new user ID differs from the initial user ID under which this program began execution, the saved privilege set is replaced by the effective privilege set, and the effective privilege set is cleared.

If the process calling `setgid()` has the `PRIV_PROC_SETID` privilege, the real, effective, and saved group IDs are set to the *gid* parameter.

If the process calling `setgid()` does not have the `PRIV_PROC_SETID` privilege, but *gid* is either the real group ID or the saved group ID of the calling process, the effective group ID is set to *gid*.

RETURN VALUES

`setuid()` returns:

- 0 On success.
- 1 On failure, and sets `errno` to indicate the error.

ERRORS

The `setuid()` and `setgid()` functions will fail if:

- `EINVAL` The value of *uid* or *gid* is out of range.
- `EPERM` For `setuid()` and `seteuid()`, the calling process does not have `PRIV_PROC_SETID` in its effective set of privileges, and the *uid* parameter does not match either the real or saved user IDs.

For `setgid()` and `setegid()`, the calling process does not have `PRIV_PROC_SETID` in its effective set of privileges, and the *gid* parameter does not match either the real or the saved group ID.

ATTRIBUTES

See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	<code>setuid()</code> and <code>setgid()</code> are Async-Signal-Safe

SUMMARY OF TRUSTED SOLARIS CHANGES Trusted Solaris 8 4/01 Reference Manual SunOS 5.8 Reference Manual

The Trusted Solaris environment replaces the checks of super-user in the Solaris environment with a check for `PRIV_PROC_SETID`.

`intro(2)`, `exec(2)`, `getgroups(2)`
`getuid(2)`, `attributes(5)`, `stat(5)`

NAME	shmop, shmat, shmdt – shared memory operations
SYNOPSIS	<pre>#include <sys/types.h> #include <sys/shm.h> void *shmat(int shmid, const void *shmaddr, int shmflg);</pre>
Default	int shmdt(char *shmaddr);
Standard-conforming	int shmdt(const void *shmaddr);
DESCRIPTION	<p>The <code>shmat()</code> function attaches the shared memory segment associated with the shared memory identifier specified by <code>shmid</code> to the data segment of the calling process.</p> <p>The permission required for a shared memory control operation is given as <code>{token}</code>, where <code>token</code> is the type of permission needed. The types of permission are interpreted as follows:</p> <pre>00400 READ by user 00200 WRITE by user 00040 READ by group 00020 WRITE by group 00004 READ by others 00002 WRITE by others</pre> <p>See the <i>Shared Memory Operation Permissions</i> section of <code>intro(2)</code> for more information.</p> <p>A process attempting to map a shared-memory segment as read-only (<code>shmflg&SHM_RDONLY</code>) must either have discretionary and mandatory read access to the shared-memory object or have the necessary privileges in its set of effective privileges: <code>PRIV_IPC_DAC_READ</code> and <code>PRIV_IPC_MAC_READ</code>. Otherwise, mapping the shared-memory segment for reading and writing requires that the process have discretionary and mandatory read access and discretionary and mandatory write access to the shared memory object, or that the effective privilege set of the process include these privileges as necessary: <code>PRIV_IPC_DAC_READ</code>, <code>PRIV_IPC_MAC_READ</code>, <code>PRIV_IPC_DAC_WRITE</code>, and <code>PRIV_IPC_MAC_WRITE</code>.</p> <p>When (<code>shmflg&SHM_SHARE_MMU</code>) is true, virtual memory resources in addition to shared memory itself are shared among processes that use the same shared memory.</p> <p>The shared memory segment is attached to the data segment of the calling process at the address specified based on one of the following criteria:</p> <ul style="list-style-type: none"> ■ If <code>shmaddr</code> is equal to <code>(void *) 0</code>, the segment is attached to the first available address as selected by the system. ■ If <code>shmaddr</code> is equal to <code>(void *) 0</code> and (<code>shmflg&SHM_SHARE_MMU</code>) is true, then the segment is attached to the first available suitably aligned address. When (<code>shmflg&SHM_SHARE_MMU</code>) is set, however, the permission given by <code>shmget()</code> determines whether the segment is attached for reading or reading and writing. ■ If <code>shmaddr</code> is not equal to <code>(void *) 0</code> and (<code>shmflg&SHM_RND</code>) is true, the segment is attached to the address given by <code>(shmaddr - (shmaddr modulus SHMLBA))</code>.

shmat(2)

- If *shmaddr* is not equal to (void *) 0 and (*shmflg*&SHM_RND) is false, the segment is attached to the address given by *shmaddr*.

The segment is attached for reading if (*shmflg*&SHM_RDONLY) is true {READ}, otherwise it is attached for reading and writing {READ/WRITE}.

The *shmdt* () function detaches from the calling process's data segment the shared memory segment located at the address specified by *shmaddr*. If the application is standard-conforming [see standards(5)], the *shmaddr* argument is of type const void *. Otherwise it is of type char *.

Shared memory segments must be explicitly removed after the last reference to them has been removed.

RETURN VALUES

Upon successful completion, *shmat* () returns the data segment start address of the attached shared memory segment; *shmdt* () returns 0. Otherwise, -1 is returned, the shared memory segment is not attached, and *errno* is set to indicate the error.

ERRORS

The *shmat* () function will fail if:

- | | |
|--------|--|
| EACCES | Operation permission is denied to the calling process [see intro(2)], and the calling process does not have the appropriate privilege(s) in its set of effective privileges. |
| EINVAL | The <i>shm</i> id argument is not a valid shared memory identifier. |
| EINVAL | The <i>shmaddr</i> argument is not equal to 0, and the value of (<i>shmaddr</i> - (<i>shmaddr</i> modulus SHMLBA)) is an illegal address. |
| EINVAL | The <i>shmaddr</i> argument is not equal to 0, is an illegal address, and (<i>shmflg</i> &SHM_RND) is false. |
| EINVAL | The <i>shmaddr</i> argument is not equal to 0, is not properly aligned, and (<i>shmflg</i> &SHM_SHARE_MMU) is true. |
| EINVAL | SHM_SHARE_MMU is not supported in certain architectures. |
| EMFILE | The number of shared memory segments attached to the calling process would exceed the system-imposed limit. |
| ENOMEM | The available data space is not large enough to accommodate the shared memory segment. |
| EPERM | The LOCK and UNLOCK operation does not have the appropriate privilege in its set of effective privileges. |

The *shmdt* () function will fail if:

- | | |
|--------|---|
| EINVAL | The <i>shmaddr</i> argument is not the data segment start address of a shared memory segment. |
|--------|---|

SUMMARY OF TRUSTED SOLARIS CHANGES

Appropriate privilege is required to override access checks.

intro(2), exec(2), fork(2), shmctl(2), shmget(2)
exit(2), xpg4(5)

shmat(2)

shmctl(2)

NAME	shmctl – Shared memory control operations						
SYNOPSIS	<pre>#include <sys/types.h> #include <sys/ipc.h> #include <sys/shm.h> int shmctl(int shmid, int cmd, struct shmid_ds *buf);</pre>						
DESCRIPTION	<p>The <code>shmctl()</code> function provides a variety of shared memory control operations as specified by <i>cmd</i>. The permission required for a shared memory control operation is given as <i>{token}</i>, where <i>token</i> is the type of permission needed. The types of permission are interpreted as follows:</p> <pre>00400 READ by user 00200 WRITE by user 00040 READ by group 00020 WRITE by group 00004 READ by others 00002 WRITE by others</pre> <p>See the <i>Shared Memory Operation Permissions</i> section of <code>intro(2)</code> for more information.</p> <p>The following operations require the specified tokens:</p> <table> <tr> <td>IPC_STAT</td><td> <p>Place the current value of each member of the data structure associated with <i>shmid</i> into the structure pointed to by <i>buf</i>. The contents of this structure are defined in <code>intro(2)</code> {READ}</p> <p>The calling process must either have mandatory read access to the shared-memory segment or have asserted the <code>PRIV_IPC_MAC_READ</code> privilege, and either have discretionary read access to the data structure or have <code>PRIV_IPC_DAC_READ</code> in its set of effective privileges.</p> </td></tr> <tr> <td>IPC_SET</td><td> <p>Set the value of the following members of the data structure associated with <i>shmid</i> to the corresponding value found in the structure pointed to by <i>buf</i>:</p> <pre>shm_perm.uid shm_perm.gid shm_perm.mode /* access permission bits only */</pre> <p>This command can be executed only by a process that either has an effective user ID equal to <code>shm_perm.cuid</code> or <code>shm_perm.uid</code> in the data structure associated with <i>shmid</i>, or has <code>PRIV_IPC_OWNER</code> in its set of effective privileges. In addition, the process must either have mandatory write access to the semaphore set or have asserted the <code>PRIV_IPC_MAC_WRITE</code> privilege.</p> </td></tr> <tr> <td>IPC_RMID</td><td> <p>Remove from the system the shared-memory identifier specified by <i>shmid</i> and destroy the shared-memory segment and data structure associated with the identifier. This command can be executed only by a process that either has an effective user ID</p> </td></tr> </table>	IPC_STAT	<p>Place the current value of each member of the data structure associated with <i>shmid</i> into the structure pointed to by <i>buf</i>. The contents of this structure are defined in <code>intro(2)</code> {READ}</p> <p>The calling process must either have mandatory read access to the shared-memory segment or have asserted the <code>PRIV_IPC_MAC_READ</code> privilege, and either have discretionary read access to the data structure or have <code>PRIV_IPC_DAC_READ</code> in its set of effective privileges.</p>	IPC_SET	<p>Set the value of the following members of the data structure associated with <i>shmid</i> to the corresponding value found in the structure pointed to by <i>buf</i>:</p> <pre>shm_perm.uid shm_perm.gid shm_perm.mode /* access permission bits only */</pre> <p>This command can be executed only by a process that either has an effective user ID equal to <code>shm_perm.cuid</code> or <code>shm_perm.uid</code> in the data structure associated with <i>shmid</i>, or has <code>PRIV_IPC_OWNER</code> in its set of effective privileges. In addition, the process must either have mandatory write access to the semaphore set or have asserted the <code>PRIV_IPC_MAC_WRITE</code> privilege.</p>	IPC_RMID	<p>Remove from the system the shared-memory identifier specified by <i>shmid</i> and destroy the shared-memory segment and data structure associated with the identifier. This command can be executed only by a process that either has an effective user ID</p>
IPC_STAT	<p>Place the current value of each member of the data structure associated with <i>shmid</i> into the structure pointed to by <i>buf</i>. The contents of this structure are defined in <code>intro(2)</code> {READ}</p> <p>The calling process must either have mandatory read access to the shared-memory segment or have asserted the <code>PRIV_IPC_MAC_READ</code> privilege, and either have discretionary read access to the data structure or have <code>PRIV_IPC_DAC_READ</code> in its set of effective privileges.</p>						
IPC_SET	<p>Set the value of the following members of the data structure associated with <i>shmid</i> to the corresponding value found in the structure pointed to by <i>buf</i>:</p> <pre>shm_perm.uid shm_perm.gid shm_perm.mode /* access permission bits only */</pre> <p>This command can be executed only by a process that either has an effective user ID equal to <code>shm_perm.cuid</code> or <code>shm_perm.uid</code> in the data structure associated with <i>shmid</i>, or has <code>PRIV_IPC_OWNER</code> in its set of effective privileges. In addition, the process must either have mandatory write access to the semaphore set or have asserted the <code>PRIV_IPC_MAC_WRITE</code> privilege.</p>						
IPC_RMID	<p>Remove from the system the shared-memory identifier specified by <i>shmid</i> and destroy the shared-memory segment and data structure associated with the identifier. This command can be executed only by a process that either has an effective user ID</p>						

	equal to <code>shm_perm.cuid</code> or <code>shm_perm.uid</code> in the data structure associated with <i>shmid</i> , or has <code>PRIV_IPC_OWNER</code> in its set of effective privileges. In addition, the process must either have mandatory write access to the shared memory segment or have asserted the <code>PRIV_IPC_MAC_WRITE</code> privilege.
SHM_LOCK	Lock the shared-memory segment specified by <i>shmid</i> in memory. This command can be executed only by a process that has discretionary and mandatory read access (or the appropriate privilege override) and also has <code>PRIV_SYS_CONFIG</code> in its effective privilege set.
SHM_UNLOCK	Unlock the shared-memory segment specified by <i>shmid</i> . This command can be executed only by a process that has discretionary and mandatory read access (or the appropriate privilege override) and also has <code>PRIV_SYS_CONFIG</code> in its effective privilege set.
	Shared memory segments must be explicitly removed after the last reference to them has been removed.
RETURN VALUES	<p><code>shmctl()</code> returns:</p> <p>0 On success.</p> <p>-1 On failure, and sets <code>errno</code> to indicate the error.</p>
ERRORS	<p>The <code>shmctl()</code> function will fail if:</p> <p>EACCES <i>cmd</i> is equal to <code>IPC_STAT</code>. {READ} operation permission is denied to the calling process, and the calling process does not have the appropriate privilege(s) in its set of effective privileges.</p> <p>EFAULT The <i>buf</i> argument points to an illegal address.</p> <p>EINVAL The <i>shmid</i> argument is not a valid shared memory identifier; or the <i>cmd</i> argument is not a valid command or is <code>IPC_SET</code> and <code>shm_perm.uid</code> or <code>shm_perm.gid</code> is not valid.</p> <p>ENOMEM The <i>cmd</i> argument is equal to <code>SHM_LOCK</code> and there is not enough memory.</p> <p>EOVERFLOW The <i>cmd</i> argument is <code>IPC_STAT</code> and <i>uid</i> or <i>gid</i> is too large to be stored in the structure pointed to by <i>buf</i>.</p> <p>EPERM <i>cmd</i> is equal to <code>IPC_RMID</code> or <code>IPC_SET</code>. The effective user ID of the calling process does not match the value of <code>shm_perm.cuid</code> or <code>shm_perm.uid</code> in the data structure associated with <i>shmid</i>; or the mandatory access check failed; and the calling process does not have the appropriate privilege overrides(s) in its set of effective privileges.</p> <p><i>cmd</i> is equal to <code>SHM_LOCK</code> or <code>SHM_UNLOCK</code> and <code>PRIV_SYS_CONFIG</code> is not in the effective privilege set of the</p>

shmctl(2)

process.

**SUMMARY OF
TRUSTED
SOLARIS
Trusted Solaris 3
4/01 Reference
Manual**

Appropriate privilege is required to override access checks.

ipcs(1), intro(2), shmget(2), shmop(2)

NAME	shmop, shmat, shmdt – shared memory operations
SYNOPSIS	<pre>#include <sys/types.h> #include <sys/shm.h> void *shmat(int shmid, const void *shmaddr, int shmflg);</pre>
Default	int shmdt(char *shmaddr);
Standard-conforming	int shmdt(const void *shmaddr);
DESCRIPTION	<p>The shmat() function attaches the shared memory segment associated with the shared memory identifier specified by <i>shmid</i> to the data segment of the calling process.</p> <p>The permission required for a shared memory control operation is given as <i>{token}</i>, where <i>token</i> is the type of permission needed. The types of permission are interpreted as follows:</p> <pre>00400 READ by user 00200 WRITE by user 00040 READ by group 00020 WRITE by group 00004 READ by others 00002 WRITE by others</pre> <p>See the <i>Shared Memory Operation Permissions</i> section of intro(2) for more information.</p> <p>A process attempting to map a shared-memory segment as read-only (<i>shmflg</i>&SHM_RDONLY) must either have discretionary and mandatory read access to the shared-memory object or have the necessary privileges in its set of effective privileges: PRIV_IPC_DAC_READ and PRIV_IPC_MAC_READ. Otherwise, mapping the shared-memory segment for reading and writing requires that the process have discretionary and mandatory read access and discretionary and mandatory write access to the shared memory object, or that the effective privilege set of the process include these privileges as necessary: PRIV_IPC_DAC_READ, PRIV_IPC_MAC_READ, PRIV_IPC_DAC_WRITE, and PRIV_IPC_MAC_WRITE.</p> <p>When (<i>shmflg</i>&SHM_SHARE_MMU) is true, virtual memory resources in addition to shared memory itself are shared among processes that use the same shared memory.</p> <p>The shared memory segment is attached to the data segment of the calling process at the address specified based on one of the following criteria:</p> <ul style="list-style-type: none"> ■ If <i>shmaddr</i> is equal to (void *) 0, the segment is attached to the first available address as selected by the system. ■ If <i>shmaddr</i> is equal to (void *) 0 and (<i>shmflg</i>&SHM_SHARE_MMU) is true, then the segment is attached to the first available suitably aligned address. When (<i>shmflg</i>&SHM_SHARE_MMU) is set, however, the permission given by shmget() determines whether the segment is attached for reading or reading and writing. ■ If <i>shmaddr</i> is not equal to (void *) 0 and (<i>shmflg</i>&SHM_RND) is true, the segment is attached to the address given by (<i>shmaddr</i> - (<i>shmaddr</i> modulus SHMLBA)).

shmdt(2)

	<ul style="list-style-type: none"> ■ If <i>shmaddr</i> is not equal to (void *) 0 and (<i>shmflg</i>&SHM_RND) is false, the segment is attached to the address given by <i>shmaddr</i>. <p>The segment is attached for reading if (<i>shmflg</i>&SHM_RDONLY) is true {READ}, otherwise it is attached for reading and writing {READ/WRITE}.</p> <p>The <i>shmdt</i> () function detaches from the calling process's data segment the shared memory segment located at the address specified by <i>shmaddr</i>. If the application is standard-conforming [see standards(5)], the <i>shmaddr</i> argument is of type const void *. Otherwise it is of type char *.</p> <p>Shared memory segments must be explicitly removed after the last reference to them has been removed.</p>																				
RETURN VALUES	<p>Upon successful completion, <i>shmat</i> () returns the data segment start address of the attached shared memory segment; <i>shmdt</i> () returns 0. Otherwise, -1 is returned, the shared memory segment is not attached, and <i>errno</i> is set to indicate the error.</p>																				
ERRORS	<p>The <i>shmat</i> () function will fail if:</p> <table> <tr> <td>EACCES</td><td>Operation permission is denied to the calling process [see intro(2)], and the calling process does not have the appropriate privilege(s) in its set of effective privileges.</td></tr> <tr> <td>EINVAL</td><td>The <i>shm</i>id argument is not a valid shared memory identifier.</td></tr> <tr> <td>EINVAL</td><td>The <i>shmaddr</i> argument is not equal to 0, and the value of (<i>shmaddr</i> - (<i>shmaddr</i> modulus SHMLBA)) is an illegal address.</td></tr> <tr> <td>EINVAL</td><td>The <i>shmaddr</i> argument is not equal to 0, is an illegal address, and (<i>shmflg</i>&SHM_RND) is false.</td></tr> <tr> <td>EINVAL</td><td>The <i>shmaddr</i> argument is not equal to 0, is not properly aligned, and (<i>shmflg</i>&SHM_SHARE_MMU) is true.</td></tr> <tr> <td>EINVAL</td><td>SHM_SHARE_MMU is not supported in certain architectures.</td></tr> <tr> <td>EMFILE</td><td>The number of shared memory segments attached to the calling process would exceed the system-imposed limit.</td></tr> <tr> <td>ENOMEM</td><td>The available data space is not large enough to accommodate the shared memory segment.</td></tr> <tr> <td>EPERM</td><td>The LOCK and UNLOCK operation does not have the appropriate privilege in its set of effective privileges.</td></tr> </table> <p>The <i>shmdt</i> () function will fail if:</p> <table> <tr> <td>EINVAL</td><td>The <i>shmaddr</i> argument is not the data segment start address of a shared memory segment.</td></tr> </table>	EACCES	Operation permission is denied to the calling process [see intro(2)], and the calling process does not have the appropriate privilege(s) in its set of effective privileges.	EINVAL	The <i>shm</i> id argument is not a valid shared memory identifier.	EINVAL	The <i>shmaddr</i> argument is not equal to 0, and the value of (<i>shmaddr</i> - (<i>shmaddr</i> modulus SHMLBA)) is an illegal address.	EINVAL	The <i>shmaddr</i> argument is not equal to 0, is an illegal address, and (<i>shmflg</i> &SHM_RND) is false.	EINVAL	The <i>shmaddr</i> argument is not equal to 0, is not properly aligned, and (<i>shmflg</i> &SHM_SHARE_MMU) is true.	EINVAL	SHM_SHARE_MMU is not supported in certain architectures.	EMFILE	The number of shared memory segments attached to the calling process would exceed the system-imposed limit.	ENOMEM	The available data space is not large enough to accommodate the shared memory segment.	EPERM	The LOCK and UNLOCK operation does not have the appropriate privilege in its set of effective privileges.	EINVAL	The <i>shmaddr</i> argument is not the data segment start address of a shared memory segment.
EACCES	Operation permission is denied to the calling process [see intro(2)], and the calling process does not have the appropriate privilege(s) in its set of effective privileges.																				
EINVAL	The <i>shm</i> id argument is not a valid shared memory identifier.																				
EINVAL	The <i>shmaddr</i> argument is not equal to 0, and the value of (<i>shmaddr</i> - (<i>shmaddr</i> modulus SHMLBA)) is an illegal address.																				
EINVAL	The <i>shmaddr</i> argument is not equal to 0, is an illegal address, and (<i>shmflg</i> &SHM_RND) is false.																				
EINVAL	The <i>shmaddr</i> argument is not equal to 0, is not properly aligned, and (<i>shmflg</i> &SHM_SHARE_MMU) is true.																				
EINVAL	SHM_SHARE_MMU is not supported in certain architectures.																				
EMFILE	The number of shared memory segments attached to the calling process would exceed the system-imposed limit.																				
ENOMEM	The available data space is not large enough to accommodate the shared memory segment.																				
EPERM	The LOCK and UNLOCK operation does not have the appropriate privilege in its set of effective privileges.																				
EINVAL	The <i>shmaddr</i> argument is not the data segment start address of a shared memory segment.																				
SUMMARY OF TRUSTED SOLARIS CHANGES	<p>Appropriate privilege is required to override access checks.</p>																				

intro(2), exec(2), fork(2), shmctl(2), shmget(2)
exit(2), xpg4(5)

shmdt(2)

shmget(2)

NAME	shmget, shmgetl – Get shared memory segment identifier
SYNOPSIS	<pre>cc [<i>flags...</i>] <i>file</i> ... -ltsol [<i>library...</i>] #include <sys/types.h> #include <sys/ipc.h> #include <sys/shm.h> int shmget(key_t <i>key</i>, size_t <i>size</i>, int <i>shmflg</i>); #include <sys/tsol/ipcl.h> int shmgetl(key_t <i>key</i>, size_t <i>size</i>, int <i>shmflg</i>, const bslabel_t *<i>slabel</i>);</pre>
DESCRIPTION	<p>A shared-memory segment is identified by a unique combination of key and sensitivity label. This qualification of keys by sensitivity labels allows applications that use shared-memory segments to be run at multiple process sensitivity labels without inadvertently sharing data. <code>shmget()</code> returns the shared-memory identifier associated with the union of <i>key</i> and the sensitivity label of the calling process.</p> <p><code>shmgetl()</code> returns the shared-memory identifier associated with the union of <i>key</i> and <i>slabel</i>. If the value of <i>slabel</i> does not match the sensitivity label of the calling process, then the effective privilege set of the process must include the necessary privileges: <code>PRIV_IPC_MAC_READ</code> and <code>PRIV_IPC_MAC_WRITE</code>.</p> <p>If discretionary read/write access is denied to the calling process as specified by the low-order 9 bits of <i>shmflg</i>, both <code>shmget()</code> and <code>shmgetl()</code> require one or both of these privileges: <code>PRIV_IPC_DAC_READ</code> and <code>PRIV_IPC_DAC_WRITE</code>.</p> <p>The <code>shmget()</code> function returns the shared memory identifier associated with <i>key</i>.</p> <p>A shared memory identifier and associated data structure and shared memory segment of at least <i>size</i> bytes (see <code>intro(3)</code>) are created for <i>key</i> if one of the following are true:</p> <ul style="list-style-type: none">■ The <i>key</i> argument is equal to <code>IPC_PRIVATE</code>.■ The <i>key</i> argument does not already have a shared memory identifier associated with it, and (<i>shmflg</i> & <code>IPC_CREAT</code>) is true. <p>Upon creation, the data structure associated with the new shared memory identifier is initialized as follows:</p> <ul style="list-style-type: none">■ The values of <code>shm_perm.cuid</code>, <code>shm_perm.uid</code>, <code>shm_perm.cgid</code>, and <code>shm_perm.gid</code> are set equal to the effective user ID and effective group ID, respectively, of the calling process.■ The access permission bits of <code>shm_perm.mode</code> are set equal to the access permission bits of <i>shmflg</i>. <code>shm_segsz</code> is set equal to the value of <i>size</i>.■ The values of <code>shm_lpid</code>, <code>shm_nattch</code>, <code>shm_atime</code>, and <code>shm_dtime</code> are set equal to 0.

RETURN VALUES

- The `shm_ctime` is set equal to the current time.

Shared memory segments must be explicitly removed after the last reference to them has been removed.

Upon successful completion, a non-negative integer representing a shared memory identifier is returned. Otherwise, `-1` is returned and `errno` is set to indicate the error.

ERRORS

The `shmget()` function will fail if:

EACCES	A shared memory identifier exists for the union of key and sensitivity label, but operation permission (see <code>intro(3)</code>) as specified by the low-order 9 bits of <code>shmflg</code> would not be granted. The calling process, which failed the check for discretionary or mandatory access, does not have the appropriate privilege override(s) in its set of effective privileges.
EEXIST	A shared memory identifier exists for <i>key</i> but both (<code>shmflg&IPC_CREATE</code>) and (<code>shmflg&IPC_EXCL</code>) are true.
EFAULT	<i>slabel</i> points to an illegal address.
EINVAL	The <i>size</i> argument is less than the system-imposed minimum or greater than the system-imposed maximum.
EINVAL	A shared memory identifier exists for <i>key</i> but the size of the segment associated with it is less than <i>size</i> and <i>size</i> is not equal to 0. The label pointed to by <i>slabel</i> is not a valid sensitivity label.
ENOENT	A shared memory identifier does not exist for the union of key and sensitivity label, and (<code>shmflg&IPC_CREATE</code>) is false.
ENOMEM	A shared memory identifier and associated shared memory segment are to be created but the amount of available memory is not sufficient to fill the request.
ENOSPC	A shared memory identifier is to be created but the system-imposed limit on the maximum number of allowed shared memory identifiers system-wide would be exceeded.

SUMMARY OF
TRUSTED
SOLARIS
CHANGES

Trusted Solaris 8
4/01 Reference
Manual

Appropriate privilege is required to override access checks.

Sensitivity labels are used together with *key* to determine shared-memory identifiers.

`shmctl(2)`, `shmop(2)`, `intro(3)`

`ftok(3C)`

shmgetl(2)

NAME	shmget, shmgetl – Get shared memory segment identifier
SYNOPSIS	<pre>cc [flags...] file ... -ltsol [library...] #include <sys/types.h> #include <sys/ipc.h> #include <sys/shm.h> int shmget(key_t key, size_t size, int shmflg); #include <sys/tsol/ipcl.h> int shmgetl(key_t key, size_t size, int shmflg, const bslabel_t *slabel);</pre>
DESCRIPTION	<p>A shared-memory segment is identified by a unique combination of key and sensitivity label. This qualification of keys by sensitivity labels allows applications that use shared-memory segments to be run at multiple process sensitivity labels without inadvertently sharing data. <code>shmget()</code> returns the shared-memory identifier associated with the union of <i>key</i> and the sensitivity label of the calling process.</p> <p><code>shmgetl()</code> returns the shared-memory identifier associated with the union of <i>key</i> and <i>slabel</i>. If the value of <i>slabel</i> does not match the sensitivity label of the calling process, then the effective privilege set of the process must include the necessary privileges: <code>PRIV_IPC_MAC_READ</code> and <code>PRIV_IPC_MAC_WRITE</code>.</p> <p>If discretionary read/write access is denied to the calling process as specified by the low-order 9 bits of <i>shmflg</i>, both <code>shmget()</code> and <code>shmgetl()</code> require one or both of these privileges: <code>PRIV_IPC_DAC_READ</code> and <code>PRIV_IPC_DAC_WRITE</code>.</p> <p>The <code>shmget()</code> function returns the shared memory identifier associated with <i>key</i>.</p> <p>A shared memory identifier and associated data structure and shared memory segment of at least <i>size</i> bytes (see <code>intro(3)</code>) are created for <i>key</i> if one of the following are true:</p> <ul style="list-style-type: none">■ The <i>key</i> argument is equal to <code>IPC_PRIVATE</code>.■ The <i>key</i> argument does not already have a shared memory identifier associated with it, and $(shmflg \& IPC_CREAT)$ is true. <p>Upon creation, the data structure associated with the new shared memory identifier is initialized as follows:</p> <ul style="list-style-type: none">■ The values of <code>shm_perm.cuid</code>, <code>shm_perm.uid</code>, <code>shm_perm.cgid</code>, and <code>shm_perm.gid</code> are set equal to the effective user ID and effective group ID, respectively, of the calling process.■ The access permission bits of <code>shm_perm.mode</code> are set equal to the access permission bits of <i>shmflg</i>. <code>shm_segsz</code> is set equal to the value of <i>size</i>.■ The values of <code>shm_lpid</code>, <code>shm_nattch</code>, <code>shm_atime</code>, and <code>shm_dtime</code> are set equal to 0.

RETURN VALUES

- The `shm_ctime` is set equal to the current time.

Shared memory segments must be explicitly removed after the last reference to them has been removed.

Upon successful completion, a non-negative integer representing a shared memory identifier is returned. Otherwise, `-1` is returned and `errno` is set to indicate the error.

ERRORS

The `shmget()` function will fail if:

EACCES	A shared memory identifier exists for the union of key and sensitivity label, but operation permission (see <code>intro(3)</code>) as specified by the low-order 9 bits of <code>shmflg</code> would not be granted. The calling process, which failed the check for discretionary or mandatory access, does not have the appropriate privilege override(s) in its set of effective privileges.
EEXIST	A shared memory identifier exists for <i>key</i> but both (<code>shmflg&IPC_CREATE</code>) and (<code>shmflg&IPC_EXCL</code>) are true.
EFAULT	<i>slabel</i> points to an illegal address.
EINVAL	The <i>size</i> argument is less than the system-imposed minimum or greater than the system-imposed maximum.
EINVAL	A shared memory identifier exists for <i>key</i> but the size of the segment associated with it is less than <i>size</i> and <i>size</i> is not equal to 0. The label pointed to by <i>slabel</i> is not a valid sensitivity label.
ENOENT	A shared memory identifier does not exist for the union of key and sensitivity label, and (<code>shmflg&IPC_CREATE</code>) is false.
ENOMEM	A shared memory identifier and associated shared memory segment are to be created but the amount of available memory is not sufficient to fill the request.
ENOSPC	A shared memory identifier is to be created but the system-imposed limit on the maximum number of allowed shared memory identifiers system-wide would be exceeded.

SUMMARY OF
TRUSTED
SOLARIS
CHANGES

Trusted Solaris 8
4/01 Reference
Manual

Appropriate privilege is required to override access checks.

Sensitivity labels are used together with *key* to determine shared-memory identifiers.

`shmctl(2)`, `shmop(2)`, `intro(3)`

`ftok(3C)`

shmop(2)

NAME	shmop, shmat, shmdt – shared memory operations
SYNOPSIS	<pre>#include <sys/types.h> #include <sys/shm.h> void *shmat(int shmid, const void *shmaddr, int shmflg); int shmdt(char *shmaddr);</pre>
Default	int shmdt(const void *shmaddr);
Standard-conforming	int shmdt(const void *shmaddr);
DESCRIPTION	<p>The shmat() function attaches the shared memory segment associated with the shared memory identifier specified by <i>shmid</i> to the data segment of the calling process.</p> <p>The permission required for a shared memory control operation is given as <i>{token}</i>, where <i>token</i> is the type of permission needed. The types of permission are interpreted as follows:</p> <pre>00400 READ by user 00200 WRITE by user 00040 READ by group 00020 WRITE by group 00004 READ by others 00002 WRITE by others</pre> <p>See the <i>Shared Memory Operation Permissions</i> section of intro(2) for more information.</p> <p>A process attempting to map a shared-memory segment as read-only (<i>shmflg</i>&SHM_RDONLY) must either have discretionary and mandatory read access to the shared-memory object or have the necessary privileges in its set of effective privileges: PRIV_IPC_DAC_READ and PRIV_IPC_MAC_READ. Otherwise, mapping the shared-memory segment for reading and writing requires that the process have discretionary and mandatory read access and discretionary and mandatory write access to the shared memory object, or that the effective privilege set of the process include these privileges as necessary: PRIV_IPC_DAC_READ, PRIV_IPC_MAC_READ, PRIV_IPC_DAC_WRITE, and PRIV_IPC_MAC_WRITE.</p> <p>When (<i>shmflg</i>&SHM_SHARE_MMU) is true, virtual memory resources in addition to shared memory itself are shared among processes that use the same shared memory.</p> <p>The shared memory segment is attached to the data segment of the calling process at the address specified based on one of the following criteria:</p> <ul style="list-style-type: none"> ■ If <i>shmaddr</i> is equal to (void *) 0, the segment is attached to the first available address as selected by the system. ■ If <i>shmaddr</i> is equal to (void *) 0 and (<i>shmflg</i>&SHM_SHARE_MMU) is true, then the segment is attached to the first available suitably aligned address. When (<i>shmflg</i>&SHM_SHARE_MMU) is set, however, the permission given by shmget() determines whether the segment is attached for reading or reading and writing. ■ If <i>shmaddr</i> is not equal to (void *) 0 and (<i>shmflg</i>&SHM_RND) is true, the segment is attached to the address given by (<i>shmaddr</i> - (<i>shmaddr</i> modulus SHMLBA)).

- If *shmaddr* is not equal to (void *) 0 and (*shmflg*&SHM_RND) is false, the segment is attached to the address given by *shmaddr*.

The segment is attached for reading if (*shmflg*&SHM_RDONLY) is true {READ}, otherwise it is attached for reading and writing {READ/WRITE}.

The *shmdt* () function detaches from the calling process's data segment the shared memory segment located at the address specified by *shmaddr*. If the application is standard-conforming [see standards(5)], the *shmaddr* argument is of type const void *. Otherwise it is of type char *.

Shared memory segments must be explicitly removed after the last reference to them has been removed.

RETURN VALUES

Upon successful completion, *shmat* () returns the data segment start address of the attached shared memory segment; *shmdt* () returns 0. Otherwise, -1 is returned, the shared memory segment is not attached, and *errno* is set to indicate the error.

ERRORS

The *shmat* () function will fail if:

EACCES	Operation permission is denied to the calling process [see intro(2)], and the calling process does not have the appropriate privilege(s) in its set of effective privileges.
EINVAL	The <i>shmid</i> argument is not a valid shared memory identifier.
EINVAL	The <i>shmaddr</i> argument is not equal to 0, and the value of (<i>shmaddr</i> - (<i>shmaddr</i> modulus SHMLBA)) is an illegal address.
EINVAL	The <i>shmaddr</i> argument is not equal to 0, is an illegal address, and (<i>shmflg</i> &SHM_RND) is false.
EINVAL	The <i>shmaddr</i> argument is not equal to 0, is not properly aligned, and (<i>shmflg</i> &SHM_SHARE_MMU) is true.
EINVAL	SHM_SHARE_MMU is not supported in certain architectures.
EMFILE	The number of shared memory segments attached to the calling process would exceed the system-imposed limit.
ENOMEM	The available data space is not large enough to accommodate the shared memory segment.
EPERM	The LOCK and UNLOCK operation does not have the appropriate privilege in its set of effective privileges.

The *shmdt* () function will fail if:

EINVAL	The <i>shmaddr</i> argument is not the data segment start address of a shared memory segment.
--------	---

SUMMARY OF TRUSTED SOLARIS CHANGES

Appropriate privilege is required to override access checks.

shmop(2)

Trusted Solaris 8
4/01 Reference
Manual
SunOS 5.8
Reference Manual

intro(2), exec(2), fork(2), shmctl(2), shmget(2)

exit(2), xpg4(5)

NAME	sigsend, sigsendset – Send a signal to a process or a group of processes
SYNOPSIS	<pre>#include <signal.h> int sigsend(idtype_t <i>idtype</i>, id_t <i>id</i>, int <i>sig</i>); int sigsendset(procset_t *<i>psp</i>, int <i>sig</i>);</pre>
DESCRIPTION	<p>The <code>sigsend()</code> function sends a signal to the process or group of processes specified by <i>id</i> and <i>idtype</i>. The signal to be sent is specified by <i>sig</i> and is either 0 or one of the values listed in <code>signal(3HEAD)</code>. If <i>sig</i> is 0 (the null signal), error checking is performed but no signal is actually sent. This value can be used to check the validity of <i>id</i> and <i>idtype</i>.</p> <p>The sending process must have MAC write access to the receiving processes. The real or effective user ID of the sending process must match the real or saved user ID of the receiving process, unless the sending process has the <code>PRIV_PROC_OWNER</code> privilege, or <i>sig</i> is <code>SIGCONT</code> and the sending process has the same session ID as the receiving process.</p> <p>If <i>idtype</i> is <code>P_PID</code>, <i>sig</i> is sent to the process with process ID <i>id</i>.</p> <p>If <i>idtype</i> is <code>P_PGID</code>, <i>sig</i> is sent to all process with process group ID <i>id</i>.</p> <p>If <i>idtype</i> is <code>P_SID</code>, <i>sig</i> is sent to all process with session ID <i>id</i>.</p> <p>If <i>idtype</i> is <code>P_UID</code>, <i>sig</i> is sent to any process with effective user ID <i>id</i>.</p> <p>If <i>idtype</i> is <code>P_GID</code>, <i>sig</i> is sent to any process with effective group ID <i>id</i>.</p> <p>If <i>idtype</i> is <code>P_CID</code>, <i>sig</i> is sent to any process with scheduler class ID <i>id</i> (see <code>prctl(2)</code>).</p> <p>If <i>idtype</i> is <code>P_ALL</code>, <i>sig</i> is sent to all processes and <i>id</i> is ignored.</p> <p>If <i>id</i> is <code>P_MYID</code>, the value of <i>id</i> is taken from the calling process.</p> <p>The process with a process ID of 0 is always excluded. The process with a process ID of 1 is excluded unless <i>idtype</i> is equal to <code>P_PID</code>.</p> <p>The <code>sigsendset()</code> function provides an alternate interface for sending signals to sets of processes. This function sends signals to the set of processes specified by <i>psp</i>. <i>psp</i> is a pointer to a structure of type <code>procset_t</code>, defined in <code><sys/procset.h></code>, which includes the following members:</p> <pre>idop_t p_op; idtype_t p_lidtype; id_t p_lid; idtype_t p_ridtype; id_t p_rid;</pre> <p>The <code>p_lidtype</code> and <code>p_lid</code> members specify the ID type and ID of one (“left”) set of processes; the <code>p_ridtype</code> and <code>p_rid</code> members specify the ID type and ID of a</p>

sigsend(2)

second (“right”) set of processes. ID types and IDs are specified just as for the *idtype* and *id* arguments to `sigsend()`. The `p_op` member specifies the operation to be performed on the two sets of processes to get the set of processes the function is to apply to. The valid values for `p_op` and the processes they specify are:

POP_DIFF	Set difference: processes in left set and not in right set.
POP_AND	Set intersection: processes in both left and right sets.
POP_OR	Set union: processes in either left or right set or both.
POP_XOR	Set exclusive-or: processes in left or right set but not in both.

RETURN VALUES Upon successful completion, 0 is returned. Otherwise, -1 is returned and `errno` is set to indicate the error.

ERRORS The `sigsend()` and `sigsendset()` functions will fail if:

EINVAL	The <i>sig</i> argument is not a valid signal number, or the <i>idtype</i> argument is not a valid <i>idtype</i> field.
EINVAL	The <i>sig</i> argument is SIGKILL, <i>idtype</i> is P_PID and <i>id</i> is 1 (proc1).
EPERM	The calling process does not have the PRIV_PROC_OWNER privilege, and its real or effective user ID does not match the real or effective user ID of the receiving process, and the calling process is not sending SIGCONT to a process that shares the same session.
ESRCH	No process can be found corresponding to that specified by <i>id</i> and <i>idtype</i> . Or, the sending process does not have MAC write access to the specified process.

The `sigsendset()` function will fail if:

EFAULT	The <i>psp</i> argument points to an illegal address.
--------	---

SUMMARY OF TRUSTED SOLARIS CHANGES
Trusted Solaris 8 4/01 Reference Manual
SunOS 5.10 Reference Manual

The sending process is required to have MAC write access to the target processes. The PRIV_PROC_MAC_WRITE and PRIV_PROC_OWNER privileges are recognized.

getpid(2), kill(2), priocntl(2)
kill(1), signal(3C), signal(3HEAD)

NAME	sigsend, sigsendset – Send a signal to a process or a group of processes
SYNOPSIS	<pre>#include <signal.h> int sigsend(idtype_t <i>idtype</i>, id_t <i>id</i>, int <i>sig</i>); int sigsendset(procset_t *<i>psp</i>, int <i>sig</i>);</pre>
DESCRIPTION	<p>The <code>sigsend()</code> function sends a signal to the process or group of processes specified by <i>id</i> and <i>idtype</i>. The signal to be sent is specified by <i>sig</i> and is either 0 or one of the values listed in <code>signal(3HEAD)</code>. If <i>sig</i> is 0 (the null signal), error checking is performed but no signal is actually sent. This value can be used to check the validity of <i>id</i> and <i>idtype</i>.</p> <p>The sending process must have MAC write access to the receiving processes. The real or effective user ID of the sending process must match the real or saved user ID of the receiving process, unless the sending process has the <code>PRIV_PROC_OWNER</code> privilege, or <i>sig</i> is <code>SIGCONT</code> and the sending process has the same session ID as the receiving process.</p> <p>If <i>idtype</i> is <code>P_PID</code>, <i>sig</i> is sent to the process with process ID <i>id</i>.</p> <p>If <i>idtype</i> is <code>P_PGID</code>, <i>sig</i> is sent to all process with process group ID <i>id</i>.</p> <p>If <i>idtype</i> is <code>P_SID</code>, <i>sig</i> is sent to all process with session ID <i>id</i>.</p> <p>If <i>idtype</i> is <code>P_UID</code>, <i>sig</i> is sent to any process with effective user ID <i>id</i>.</p> <p>If <i>idtype</i> is <code>P_GID</code>, <i>sig</i> is sent to any process with effective group ID <i>id</i>.</p> <p>If <i>idtype</i> is <code>P_CID</code>, <i>sig</i> is sent to any process with scheduler class ID <i>id</i> (see <code>prctl(2)</code>).</p> <p>If <i>idtype</i> is <code>P_ALL</code>, <i>sig</i> is sent to all processes and <i>id</i> is ignored.</p> <p>If <i>id</i> is <code>P_MYID</code>, the value of <i>id</i> is taken from the calling process.</p> <p>The process with a process ID of 0 is always excluded. The process with a process ID of 1 is excluded unless <i>idtype</i> is equal to <code>P_PID</code>.</p> <p>The <code>sigsendset()</code> function provides an alternate interface for sending signals to sets of processes. This function sends signals to the set of processes specified by <i>psp</i>. <i>psp</i> is a pointer to a structure of type <code>procset_t</code>, defined in <code><sys/procset.h></code>, which includes the following members:</p> <pre>idop_t p_op; idtype_t p_lidtype; id_t p_lid; idtype_t p_ridtype; id_t p_rid;</pre> <p>The <code>p_lidtype</code> and <code>p_lid</code> members specify the ID type and ID of one (“left”) set of processes; the <code>p_ridtype</code> and <code>p_rid</code> members specify the ID type and ID of a</p>

sigsendset(2)

second (“right”) set of processes. ID types and IDs are specified just as for the *idtype* and *id* arguments to `sigsend()`. The `p_op` member specifies the operation to be performed on the two sets of processes to get the set of processes the function is to apply to. The valid values for `p_op` and the processes they specify are:

POP_DIFF	Set difference: processes in left set and not in right set.
POP_AND	Set intersection: processes in both left and right sets.
POP_OR	Set union: processes in either left or right set or both.
POP_XOR	Set exclusive-or: processes in left or right set but not in both.

RETURN VALUES Upon successful completion, 0 is returned. Otherwise, -1 is returned and `errno` is set to indicate the error.

ERRORS The `sigsend()` and `sigsendset()` functions will fail if:

EINVAL	The <i>sig</i> argument is not a valid signal number, or the <i>idtype</i> argument is not a valid <i>idtype</i> field.
EINVAL	The <i>sig</i> argument is SIGKILL, <i>idtype</i> is P_PID and <i>id</i> is 1 (proc1).
EPERM	The calling process does not have the PRIV_PROC_OWNER privilege, and its real or effective user ID does not match the real or effective user ID of the receiving process, and the calling process is not sending SIGCONT to a process that shares the same session.
ESRCH	No process can be found corresponding to that specified by <i>id</i> and <i>idtype</i> . Or, the sending process does not have MAC write access to the specified process.

The `sigsendset()` function will fail if:

EFAULT	The <i>psp</i> argument points to an illegal address.
--------	---

SUMMARY OF TRUSTED SOLARIS CHANGES
Trusted Solaris 8 4/01 Reference Manual
SunOS 5.10 Reference Manual

The sending process is required to have MAC write access to the target processes. The PRIV_PROC_MAC_WRITE and PRIV_PROC_OWNER privileges are recognized.

getpid(2), kill(2), priocntl(2)
kill(1), signal(3C), signal(3HEAD)

NAME	stat, lstat, fstat – get file status				
SYNOPSIS	<pre>#include <sys/types.h> #include <sys/stat.h> int stat(const char *path, struct stat *buf); int lstat(const char *path, struct stat *buf); int fstat(int fd, struct stat *buf);</pre>				
DESCRIPTION	<p>The <code>stat()</code> function obtains information about the file pointed to by <i>path</i>. Read, write, or execute permission of the named file is not required, but all directories listed in the path name leading to the file must be searchable.</p> <p>The <code>lstat()</code> function obtains file attributes similar to <code>stat()</code>, except when the named file is a symbolic link; in that case <code>lstat()</code> returns information about the link, while <code>stat()</code> returns information about the file the link references.</p> <p>The <code>fstat()</code> function obtains information about an open file known by the file descriptor <i>fd</i>, obtained from a successful <code>open(2)</code>, <code>creat(2)</code>, <code>dup(2)</code>, <code>fcntl(2)</code>, or <code>pipe(2)</code> function.</p> <p>The <i>buf</i> argument is a pointer to a <code>stat</code> structure into which information is placed concerning the file. A <code>stat</code> structure includes the following members:</p> <pre>mode_t st_mode; /* File mode (see mknod(2)) */ ino_t st_ino; /* Inode number */ dev_t st_dev; /* ID of device containing */ dev_t st_rdev; /* a directory entry for this file */ dev_t st_rdev; /* ID of device */ /* This entry is defined only for */ /* char special or block special files */ nlink_t st_nlink; /* Number of links */ uid_t st_uid; /* User ID of the file's owner */ gid_t st_gid; /* Group ID of the file's group */ off_t st_size; /* File size in bytes */ time_t st_atime; /* Time of last access */ time_t st_mtime; /* Time of last data modification */ time_t st_ctime; /* Time of last file status change */ /* Times measured in seconds since */ /* 00:00:00 UTC, Jan. 1, 1970 */ long st_blksize; /* Preferred I/O block size */ blkcnt_t st_blocks; /* Number of 512 byte blocks allocated*/</pre> <p>Descriptions of structure members are as follows:</p> <table> <tr> <td><code>st_mode</code></td><td>The mode of the file as described in <code>mknod(2)</code>. In addition to the modes described in <code>mknod()</code>, the mode of a file may also be <code>S_IFLNK</code> if the file is a symbolic link. <code>S_IFLNK</code> may only be returned by <code>lstat()</code>.</td></tr> <tr> <td><code>st_ino</code></td><td>This field uniquely identifies the file in a given file system. The pair <code>st_ino</code> and <code>st_dev</code> uniquely identifies regular files.</td></tr> </table>	<code>st_mode</code>	The mode of the file as described in <code>mknod(2)</code> . In addition to the modes described in <code>mknod()</code> , the mode of a file may also be <code>S_IFLNK</code> if the file is a symbolic link. <code>S_IFLNK</code> may only be returned by <code>lstat()</code> .	<code>st_ino</code>	This field uniquely identifies the file in a given file system. The pair <code>st_ino</code> and <code>st_dev</code> uniquely identifies regular files.
<code>st_mode</code>	The mode of the file as described in <code>mknod(2)</code> . In addition to the modes described in <code>mknod()</code> , the mode of a file may also be <code>S_IFLNK</code> if the file is a symbolic link. <code>S_IFLNK</code> may only be returned by <code>lstat()</code> .				
<code>st_ino</code>	This field uniquely identifies the file in a given file system. The pair <code>st_ino</code> and <code>st_dev</code> uniquely identifies regular files.				

stat(2)

	st_dev	This field uniquely identifies the file system that contains the file. Its value may be used as input to the <code>ustat()</code> function to determine more information about this file system. No other meaning is associated with this value.
	st_rdev	This field should be used only by administrative commands. It is valid only for block special or character special files and only has meaning on the system where the file was configured.
	st_nlink	This field should be used only by administrative commands.
	st_uid	The user ID of the file's owner.
	st_gid	The group ID of the file's group.
	st_size	For regular files, this is the address of the end of the file. For block special or character special, this is not defined. See also <code>pipe(2)</code> .
	st_atime	Time when file data was last accessed. Changed by the following functions: <code>creat()</code> , <code>mknod()</code> , <code>pipe()</code> , <code>utime(2)</code> , and <code>read(2)</code> .
	st_mtime	Time when data was last modified. Changed by the following functions: <code>creat()</code> , <code>mknod()</code> , <code>pipe()</code> , <code>utime()</code> , and <code>write(2)</code> .
	st_ctime	Time when file status was last changed. Changed by the following functions: <code>chmod()</code> , <code>chown()</code> , <code>creat()</code> , <code>link(2)</code> , <code>mknod()</code> , <code>pipe()</code> , <code>unlink(2)</code> , <code>utime()</code> , and <code>write()</code> .
	st_blksize	A hint as to the "best" unit size for I/O operations. This field is not defined for block special or character special files.
	st_blocks	The total number of physical blocks of size 512 bytes actually allocated on disk. This field is not defined for block special or character special files.
	<p><code>stat()</code>, <code>lstat()</code>, and <code>fstat()</code> require mandatory read access to the final component of <i>path</i>. If the file descriptor is open only for writing, <code>fstat()</code> requires mandatory read access to the object to which the descriptor refers. To override these restrictions, the calling process may assert the <code>PRIV_FILE_MAC_READ</code> privilege in its set of effective privileges.</p> <p>If the calling process does not have mandatory read access, <code>stat()</code>, <code>lstat()</code>, and <code>fstat()</code> return fixed values for some elements of the <code>stat</code> structure.</p>	
RETURN VALUES	Upon successful completion, 0 is returned. Otherwise, -1 is returned and <code>errno</code> is set to indicate the error.	
ERRORS	The <code>stat()</code> , <code>fstat()</code> , and <code>lstat()</code> functions will fail if:	
	EOVERFLOW	The file size in bytes or the number of blocks allocated to the file or the file serial number cannot be represented correctly in the structure pointed to by <i>buf</i> .

The `stat()` and `lstat()` functions will fail if:

EACCES	Search permission is denied for a component of the path prefix. To override this restriction, the calling process may assert one or both of these privileges: <code>PRIV_FILE_DAC_SEARCH</code> and <code>PRIV_FILE_MAC_SEARCH</code> .
EFAULT	The <i>buf</i> or <i>path</i> argument points to an illegal address.
EINTR	A signal was caught during the execution of the <code>stat()</code> or <code>lstat()</code> function.
ELOOP	Too many symbolic links were encountered in translating <i>path</i> .
ENAMETOOLONG	The length of the <i>path</i> argument exceeds <code>PATH_MAX</code> , or the length of a <i>path</i> component exceeds <code>NAME_MAX</code> while <code>_POSIX_NO_TRUNC</code> is in effect.
ENOENT	The named file does not exist or is the null pathname.
ENOLINK	The <i>path</i> argument points to a remote machine and the link to that machine is no longer active.
ENOTDIR	A component of the path prefix is not a directory.
EOVERFLOW	A component is too large to store in the structure pointed to by <i>buf</i> .

The `fstat()` function will fail if:

EBADF	The <i>fildev</i> argument is not a valid open file descriptor.
EFAULT	The <i>buf</i> argument points to an illegal address.
EINTR	A signal was caught during the execution of the <code>fstat()</code> function.
ENOLINK	The <i>fildev</i> argument points to a remote machine and the link to that machine is no longer active.
EOVERFLOW	A component is too large to store in the structure pointed to by <i>buf</i> .

USAGE The `stat()`, `fstat()`, and `lstat()` functions have transitional interfaces for 64-bit file offsets. See `lf64(5)`.

ATTRIBUTES See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	<code>stat()</code> and <code>fstat()</code> are Async-Signal-Safe

SUMMARY OF TRUSTED SOLARIS CHANGES

`stat()`, `lstat()`, and `fstat()` require mandatory read access to the final component of *path*. If the file descriptor is open only for writing, `fstat()` requires

stat(2)

mandatory read access to the object to which the descriptor refers. To override these restrictions, the calling process may assert the `PRIV_FILE_MAC_READ` privilege in its set of effective privileges.

To override access restrictions, the calling process of `stat()` or `lstat()` may also assert one or both of these privileges: `PRIV_FILE_DAC_SEARCH` and `PRIV_FILE_MAC_SEARCH`.

Certain uses of this interface may present a covert channel. If a covert channel is exploited, the execution of the process may be delayed. To bypass this delay, the process may assert the `PRIV_PROC_NODELAY` privilege.

**Trusted Solaris 8
4/01 Reference
Manual**

`chmod(2)`, `chown(2)`, `creat(2)`, `fcntl(2)`, `link(2)`, `mknod(2)`, `open(2)`, `read(2)`, `unlink(2)`, `write(2)`

**SunOS 5.8
Reference Manual
NOTES**

`dup(2)`, `pipe(2)`, `time(2)`, `utime(2)`, `fattach(3C)`, `stat(3HEAD)`, `attributes(5)`

If you use `chmod(2)` to change the file group owner permissions on a file with ACL entries, both the file group owner permissions and the ACL mask are changed to the new permissions. Be aware that the new ACL mask permissions may change the effective permissions for additional users and groups who have ACL entries on the file.

NAME	statvfs, fstatvfs – get file system information
SYNOPSIS	<pre>#include <sys/types.h> #include <sys/statvfs.h> int statvfs(const char *path, struct statvfs *buf); int fstatvfs(int fildes, struct statvfs *buf);</pre>
DESCRIPTION	<p>The <code>statvfs()</code> function returns a “generic superblock” describing a file system; it can be used to acquire information about mounted file systems. The <code>buf</code> argument is a pointer to a structure (described below) that is filled by the function.</p> <p>The <code>path</code> argument should name a file that resides on that file system. The file system type is known to the operating system. Read, write, or execute permission for the named file is not required, but all directories listed in the path name leading to the file must be searchable.</p> <p>The <code>statvfs</code> structure pointed to by <code>buf</code> includes the following members:</p> <pre>u_long f_bsize; /* preferred file system block size */ u_long f_frsize; /* fundamental filesystem block (size if supported) */ fsblkcnt_t f_blocks; /* total # of blocks on file system in units of f_frsize */ fsblkcnt_t f_bfree; /* total # of free blocks */ fsblkcnt_t f_bavail; /* # of free blocks avail to non-super-user */ fsfilcnt_t f_files; /* total # of file nodes (inodes) */ fsfilcnt_t f_ffree; /* total # of free file nodes */ fsfilcnt_t f_favail; /* # of inodes avail to non-super-user */ u_long f_fsid; /* file system id (dev for now) */ char f_basetype[FSTYPESZ]; /* target fs type name, null-terminated */ u_long f_flag; /* bit mask of flags */ u_long f_namemax; /* maximum file name length */ char f_fstr[32]; /* file system specific string */ u_long f_filler[16]; /* reserved for future expansion */</pre> <p>The <code>f_basetype</code> member contains a null-terminated FSType name of the mounted target.</p> <p>The following values can be returned in the <code>f_flag</code> field:</p> <pre>ST_RDONLY 0x01 /* read-only file system */ ST_NOSUID 0x02 /* does not support setuid/setgid semantics */ ST_NOTRUNC 0x04 /* does not truncate file names longer than NAME_MAX */</pre> <p>The <code>fstatvfs()</code> function is similar to <code>statvfs()</code>, except that the file named by <code>path</code> in <code>statvfs()</code> is instead identified by an open file descriptor <code>fildes</code> obtained from a successful <code>open(2)</code>, <code>creat(2)</code>, <code>dup(2)</code>, <code>fcntl(2)</code>, or <code>pipe(2)</code> function call.</p>

statvfs(2)

RETURN VALUES

statvfs() returns:

- 0 On success.
- 1 On failure, and sets errno to indicate the error.

ERRORS

The statvfs() and fstatvfs() functions will fail if:

E_OVERFLOW One of the values to be returned cannot be represented correctly in the structure pointed to by *buf*.

The statvfs() function will fail if:

EACCES Search permission is denied on a component of the path prefix. To override this restriction, the calling process may assert one or both of these privileges: PRIV_FILE_DAC_SEARCH and PRIV_FILE_MAC_SEARCH

The calling process does not have mandatory read access to *path_name*. To override this restriction, the calling process may assert the PRIV_FILE_MAC_READ privilege.

EFAULT The *path* or *buf* argument points to an illegal address.

EINTR A signal was caught during the execution of the statvfs() function.

EIO An I/O error occurred while reading the file system.

ELOOP Too many symbolic links were encountered in translating *path*.

ENAMETOOLONG The length of a *path* component exceeds NAME_MAX characters, or the length of *path* exceeds PATH_MAX characters.

ENOENT Either a component of the path prefix or the file referred to by *path* does not exist.

ENOLINK The *path* argument points to a remote machine and the link to that machine is no longer active.

ENOTDIR A component of the path prefix of *path* is not a directory.

The fstatvfs() function will fail if:

EACCES The descriptor is open only for writing and the calling process does not have mandatory read access to the object to which the descriptor refers. To override this restriction, the calling process may assert the PRIV_FILE_MAC_READ privilege.

EBADF The *fd* argument is not an open file descriptor.

EFAULT The *buf* argument points to an illegal address.

EINTR A signal was caught during the execution of the fstatvfs() function.

SUNARIS
Trusted Solaris 4/01 Reference Manual
SunOS 5.8
Reference Manual

statvfs(2)

EIO An I/O error occurred while reading the file system.

USAGE The statvfs() and fstatvfs() functions have transitional interfaces for 64-bit file offsets. See lf64(5).

SUMMARY OF TRUSTED Appropriate privilege is required to override access checks.

CHANGES chmod(2), chown(2), creat(2), fcntl(2), link(2), mknod(2), open(2), read(2),
 unlink(2), write(2)

 dup(2), pipe(2), time(2), utime(2)

BUGS The values returned for f_files, f_ffree, and f_favail may not be valid for NFS mounted file systems.

stime(2)

NAME	stime – Set system time and date				
SYNOPSIS	<pre>#include <unistd.h> int stime(const time_t *tp);</pre>				
DESCRIPTION	The <code>stime()</code> function sets the system's idea of the time and date. The <i>tp</i> argument points to the value of time as measured in seconds from 00:00:00 UTC January 1, 1970. The calling process must have the <code>PRIV_SYS_CONFIG</code> privilege in order to use this system call.				
RETURN VALUES	<p><code>stime()</code> returns:</p> <p>0 On success.</p> <p>-1 On failure, and sets <code>errno</code> to indicate the error.</p>				
ERRORS	<p>The <code>stime()</code> function will fail if:</p> <table><tr><td>EINVAL</td><td>The <i>tp</i> argument points to an invalid (negative) time value.</td></tr><tr><td>EPERM</td><td>The calling process does not have the <code>PRIV_SYS_CONFIG</code> privilege.</td></tr></table>	EINVAL	The <i>tp</i> argument points to an invalid (negative) time value.	EPERM	The calling process does not have the <code>PRIV_SYS_CONFIG</code> privilege.
EINVAL	The <i>tp</i> argument points to an invalid (negative) time value.				
EPERM	The calling process does not have the <code>PRIV_SYS_CONFIG</code> privilege.				
SUMMARY OF TRUSTED SOLARIS CHANGES SUNOS 5.8 Reference Manual	<p>The calling process must have the <code>PRIV_SYS_CONFIG</code> privilege in order to use this system call.</p> <p><code>time(2)</code></p>				

NAME	swapctl – Manage swap space
SYNOPSIS	<pre>#include <sys/stat.h> #include <sys/swap.h> int swapctl(int cmd, void *arg);</pre>
DESCRIPTION	<p>The <code>swapctl()</code> function adds, deletes, or returns information about swap resources. <i>cmd</i> specifies one of the following options contained in <code><sys/swap.h></code>:</p> <pre>SC_ADD /* add a resource for swapping */ SC_LIST /* list the resources for swapping */ SC_REMOVE /* remove a resource for swapping */ SC_GETNSWP /* return number of swap resources */</pre> <p>When <code>SC_ADD</code> or <code>SC_REMOVE</code> is specified, <i>arg</i> is a pointer to a <code>swapres</code> structure containing the following members:</p> <pre>char *sr_name; /* pathname of resource */ off_t sr_start; /* offset to start of swap area */ off_t sr_length; /* length of swap area */</pre> <p>The <code>sr_start</code> and <code>sr_length</code> members are specified in 512-byte blocks. A swap resource can only be removed by specifying the same values for the <code>sr_start</code> and <code>sr_length</code> members as were specified when it was added. Swap resources need not be removed in the order in which they were added.</p> <p>When <code>SC_LIST</code> is specified, <i>arg</i> is a pointer to a <code>swaptable</code> structure containing the following members:</p> <pre>int swt_n; /* number of swapents following */ struct swapent swt_ent[]; /* array of swt_n swapents */</pre> <p>A <code>swapent</code> structure contains the following members:</p> <pre>char *ste_path; /* name of the swap file */ off_t ste_start; /* starting block for swapping */ off_t ste_length; /* length of swap area */ long ste_pages; /* number of pages for swapping */ long ste_free; /* number of ste_pages free */ long ste_flags; /* ST_INDEL bit set if swap file */ /* is now being deleted */</pre> <p>The <code>SC_LIST</code> function causes <code>swapctl()</code> to return at most <code>swt_n</code> entries. The return value of <code>swapctl()</code> is the number actually returned. The <code>ST_INDEL</code> bit is turned on in <code>ste_flags</code> if the swap file is in the process of being deleted.</p> <p>When <code>SC_GETNSWP</code> is specified, <code>swapctl()</code> returns as its value the number of swap resources in use. <i>arg</i> is ignored for this operation.</p> <p>The <code>SC_ADD</code> and <code>SC_REMOVE</code> functions will fail if calling process does not have appropriate privileges.</p>

swapctl(2)

RETURN VALUES	Upon successful completion, <code>swapctl()</code> returns a value of 0 for <code>SC_ADD</code> or <code>SC_REMOVE</code> , the number of <code>struct swapent</code> entries actually returned for <code>SC_LIST</code> , or the number of swap resources in use for <code>SC_GETNSWP</code> . Upon failure, <code>swapctl()</code> returns a value of -1 and sets <code>errno</code> to indicate an error.
ERRORS	Under the following conditions, the function <code>swapctl()</code> fails and sets <code>errno</code> to:
<code>EEXIST</code>	Part of the range specified by <code>sr_start</code> and <code>sr_length</code> is already being used for swapping on the specified resource (<code>SC_ADD</code>).
<code>EFAULT</code>	Either <code>arg</code> , <code>sr_name</code> , or <code>ste_path</code> points to an illegal address.
<code>EINVAL</code>	The specified function value is not valid, the path specified is not a swap resource (<code>SC_REMOVE</code>), part of the range specified by <code>sr_start</code> and <code>sr_length</code> lies outside the resource specified (<code>SC_ADD</code>), or the specified swap area is less than one page (<code>SC_ADD</code>).
<code>EISDIR</code>	The path specified for <code>SC_ADD</code> is a directory.
<code>ELOOP</code>	Too many symbolic links were encountered in translating the pathname provided to <code>SC_ADD</code> or <code>SC_REMOVE</code> .
<code>ENAMETOOLONG</code>	The length of a component of the path specified for <code>SC_ADD</code> or <code>SC_REMOVE</code> exceeds <code>NAME_MAX</code> characters or the length of the path exceeds <code>PATH_MAX</code> characters and <code>_POSIX_NO_TRUNC</code> is in effect.
<code>ENOENT</code>	The pathname specified for <code>SC_ADD</code> or <code>SC_REMOVE</code> does not exist.
<code>ENOMEM</code>	An insufficient number of <code>struct swapent</code> structures were provided to <code>SC_LIST</code> , or there were insufficient system storage resources available during an <code>SC_ADD</code> or <code>SC_REMOVE</code> , or the system would not have enough swap space after an <code>SC_REMOVE</code> .
<code>ENOSYS</code>	The pathname specified for <code>SC_ADD</code> or <code>SC_REMOVE</code> is not a file or block special device.
<code>ENOTDIR</code>	Pathname provided to <code>SC_ADD</code> or <code>SC_REMOVE</code> contained a component in the path prefix that was not a directory.
<code>EPERM</code>	The effective user of the calling process is not super-user. To override this restriction, the calling process must assert the <code>PRIV_SYS_MOUNT</code> privilege.
<code>EROFS</code>	The pathname specified for <code>SC_ADD</code> is a read-only file system.

Additionally, the `swapctl()` function will fail for 32-bit interfaces if:

EOVERFLOW The amount of swap space configured on the machine is too large to be represented by a 32-bit quantity.

EXAMPLES

EXAMPLE 1 The usage of the `SC_GETNSWP` and `SC_LIST` commands.

The following example demonstrates the usage of the `SC_GETNSWP` and `SC_LIST` commands.

```
#include <sys/stat.h>
#include <sys/swap.h>
#include <stdio.h>

#define MAXSTRSIZE 80

main(argc, argv)
    int      argc;
    char     *argv[];
{
    swaptbl_t *s;
    int      i, n, num;
    char     *strtab; /* string table for path names */

again:
    if ((num = swapctl(SC_GETNSWP, 0)) == -1) {
        perror("swapctl: GETNSWP");
        exit(1);
    }
    if (num == 0) {
        fprintf(stderr, "No Swap Devices Configured\n");
        exit(2);
    }
    /* allocate swaptable for num+1 entries */
    if ((s = (swaptbl_t *)
        malloc(num * sizeof(swaptent_t) +
              sizeof(struct swaptable))) ==
        (void *) 0) {
        fprintf(stderr, "Malloc Failed\n");
        exit(3);
    }
    /* allocate num+1 string holders */
    if ((strtab = (char *)
        malloc((num + 1) * MAXSTRSIZE)) == (void *) 0) {
        fprintf(stderr, "Malloc Failed\n");
        exit(3);
    }
    /* initialize string pointers */
    for (i = 0; i < (num + 1); i++) {
        s->swt_ent[i].ste_path = strtab + (i * MAXSTRSIZE);
    }

    s->swt_n = num + 1;
    if ((n = swapctl(SC_LIST, s)) < 0) {
        perror("swapctl");
        exit(1);
    }
}
```

swapctl(2)

EXAMPLE 1 The usage of the SC_GETNSWP and SC_LIST commands. (Continued)

```
    }
    if (n > num) {          /* more were added */
        free(s);
        free(strtab);
        goto again;
    }
    for (i = 0; i < n; i++)
        printf("%s %ld\n",
            s->swt_ent[i].ste_path, s->swt_ent[i].ste_pages);
}
```

**SUMMARY OF
TRUSTED
SOLARIS
CHANGES**

For a successful call, the calling process must assert the PRIV_SYS_MOUNT privilege.

NAME	symlink – make a symbolic link to a file	
SYNOPSIS	<pre>#include <unistd.h> int symlink(const char *name1, const char *name2);</pre>	
DESCRIPTION	<p>The <code>symlink()</code> function creates a symbolic link <i>name2</i> to the file <i>name1</i>. Either name may be an arbitrary pathname, the files need not be on the same file system, and <i>name1</i> may be nonexistent.</p> <p>The file to which the symbolic link points is used when an <code>open(2)</code> operation is performed on the link. A <code>stat()</code> operation performed on a symbolic link returns the linked-to file, while an <code>lstat()</code> operation returns information about the link itself. See <code>stat(2)</code>. Unexpected results may occur when a symbolic link is made to a directory. To avoid confusion in applications, the <code>readlink(2)</code> call can be used to read the contents of a symbolic link.</p> <p>The containing directory cannot be a multilevel directory. There is no privilege to bypass this restriction.</p> <p>The link is created with its sensitivity label set to the sensitivity label of the calling process, and its user ID set to the effective user ID of the calling process. If the file system was not mounted with the BSD file-creation semantics flag and the set-gid bit of the parent directory is clear, the new link's group ID is set to the group ID of the directory in which the link is created. The new link's permission bits are set to 0777. Even when the containing directory has a default access control list (ACL), no ACL is set on the new link.</p>	
RETURN VALUES	<p><code>symlink()</code> returns:</p> <p>0 On success.</p> <p>–1 On failure, sets <code>errno</code> to indicate the error, and the symbolic link is not made.</p>	
ERRORS	EACCES	<p>Search permission is denied for a component of the path prefix of <i>name2</i>. To override this restriction, the calling process may assert one or both of these privileges: <code>PRIV_FILE_DAC_SEARCH</code> and <code>PRIV_FILE_MAC_SEARCH</code>.</p> <p>Write permission is denied to the containing directory of <i>name2</i>. To override this restriction, the calling process may assert one or both of these privileges: <code>PRIV_FILE_DAC_WRITE</code> and <code>PRIV_FILE_MAC_WRITE</code>.</p>
	EDQUOT	<p>The directory where the entry for the new symbolic link is being placed cannot be extended because the user's quota of disk blocks on that file system has been exhausted; the new symbolic link cannot be created because the user's quota of disk blocks on that file system has been exhausted; or the user's quota of inodes on the file system where the file is being created has been exhausted.</p>

symlink(2)

EEXIST	The file referred to by <i>name2</i> already exists.
EFAULT	The <i>name1</i> or <i>name2</i> argument points to an illegal address.
EIO	An I/O error occurs while reading from or writing to the file system.
ELOOP	Too many symbolic links are encountered in translating <i>name2</i> .
ENAMETOOLONG	The length of the <i>name2</i> argument exceeds <code>PATH_MAX</code> , or the length of a <i>name2</i> component exceeds <code>NAME_MAX</code> while <code>_POSIX_NO_TRUNC</code> is in effect.
ENOENT	A component of the path prefix of <i>name2</i> does not exist.
ENOSPC	The directory in which the entry for the new symbolic link is being placed cannot be extended because no space is left on the file system containing the directory; the new symbolic link cannot be created because no space is left on the file system which will contain the link; or there are no free inodes on the file system on which the file is being created.
ENOSYS	The file system does not support symbolic links
ENOTDIR	A component of the path prefix of <i>name2</i> is not a directory.
EROFS	The file <i>name2</i> would reside on a read-only file system.

SUMMARY OF TRUSTED SOLARIS CHANGES

Trusted Solaris 8
4/01 Reference
Manual
SunOS 5.8
Reference Manual

Appropriate privilege is required to override access checks.

The containing directory cannot be a multilevel directory.

link(2), open(2), readlink(2), stat(2), unlink(2)

cp(1)

NAME	sysinfo – Get and set system information strings										
SYNOPSIS	<pre>#include <sys/systeminfo.h> long sysinfo(int <i>command</i>, char *<i>buf</i>, long <i>count</i>);</pre>										
DESCRIPTION	<p>The <code>sysinfo()</code> function copies information relating to the operating system on which the process is executing into the buffer pointed to by <i>buf</i>. It can also set certain information where appropriate <i>commands</i> are available. The <i>count</i> parameter indicates the size of the buffer.</p> <p>The POSIX P1003.1 interface (see <code>standards(5)</code>) <code>sysconf(3C)</code> provides a similar class of configuration information, but returns an integer rather than a string.</p> <p>The values for <i>command</i> are as follows:</p> <table> <tr> <td>SI_SYSNAME</td><td>Copy into the array pointed to by <i>buf</i> the string that would be returned by <code>uname(2)</code> in the <i>sysname</i> field. This is the name of the implementation of the operating system, for example, SunOS or UTS.</td></tr> <tr> <td>SI_HOSTNAME</td><td>Copy into the array pointed to by <i>buf</i> a string that names the present host machine. This is the string that would be returned by <code>uname(2)</code> in the <i>nodename</i> field. This hostname or nodename is often the name the machine is known by locally. The <i>hostname</i> is the name of this machine as a node in some network. Different networks may have different names for the node, but presenting the nodename to the appropriate network directory or name-to-address mapping service should produce a transport end point address. The name may not be fully qualified. Internet host names may be up to 256 bytes in length (plus the terminating null).</td></tr> <tr> <td>SI_SET_HOSTNAME</td><td>Copy the null-terminated contents of the array pointed to by <i>buf</i> into the string maintained by the kernel whose value will be returned by succeeding calls to <code>sysinfo()</code> with the command <code>SI_HOSTNAME</code>. This command requires that the calling process have the <code>PRIV_SYS_NET_CONFIG</code> privilege.</td></tr> <tr> <td>SI_RELEASE</td><td>Copy into the array pointed to by <i>buf</i> the string that would be returned by <code>uname(2)</code> in the <i>release</i> field. Typical values might be 5.2 or 4.1.</td></tr> <tr> <td>SI_VERSION</td><td>Copy into the array pointed to by <i>buf</i> the string that would be returned by <code>uname(2)</code> in the <i>version</i> field. The syntax and semantics of this string are defined by the system provider.</td></tr> </table>	SI_SYSNAME	Copy into the array pointed to by <i>buf</i> the string that would be returned by <code>uname(2)</code> in the <i>sysname</i> field. This is the name of the implementation of the operating system, for example, SunOS or UTS.	SI_HOSTNAME	Copy into the array pointed to by <i>buf</i> a string that names the present host machine. This is the string that would be returned by <code>uname(2)</code> in the <i>nodename</i> field. This hostname or nodename is often the name the machine is known by locally. The <i>hostname</i> is the name of this machine as a node in some network. Different networks may have different names for the node, but presenting the nodename to the appropriate network directory or name-to-address mapping service should produce a transport end point address. The name may not be fully qualified. Internet host names may be up to 256 bytes in length (plus the terminating null).	SI_SET_HOSTNAME	Copy the null-terminated contents of the array pointed to by <i>buf</i> into the string maintained by the kernel whose value will be returned by succeeding calls to <code>sysinfo()</code> with the command <code>SI_HOSTNAME</code> . This command requires that the calling process have the <code>PRIV_SYS_NET_CONFIG</code> privilege.	SI_RELEASE	Copy into the array pointed to by <i>buf</i> the string that would be returned by <code>uname(2)</code> in the <i>release</i> field. Typical values might be 5.2 or 4.1.	SI_VERSION	Copy into the array pointed to by <i>buf</i> the string that would be returned by <code>uname(2)</code> in the <i>version</i> field. The syntax and semantics of this string are defined by the system provider.
SI_SYSNAME	Copy into the array pointed to by <i>buf</i> the string that would be returned by <code>uname(2)</code> in the <i>sysname</i> field. This is the name of the implementation of the operating system, for example, SunOS or UTS.										
SI_HOSTNAME	Copy into the array pointed to by <i>buf</i> a string that names the present host machine. This is the string that would be returned by <code>uname(2)</code> in the <i>nodename</i> field. This hostname or nodename is often the name the machine is known by locally. The <i>hostname</i> is the name of this machine as a node in some network. Different networks may have different names for the node, but presenting the nodename to the appropriate network directory or name-to-address mapping service should produce a transport end point address. The name may not be fully qualified. Internet host names may be up to 256 bytes in length (plus the terminating null).										
SI_SET_HOSTNAME	Copy the null-terminated contents of the array pointed to by <i>buf</i> into the string maintained by the kernel whose value will be returned by succeeding calls to <code>sysinfo()</code> with the command <code>SI_HOSTNAME</code> . This command requires that the calling process have the <code>PRIV_SYS_NET_CONFIG</code> privilege.										
SI_RELEASE	Copy into the array pointed to by <i>buf</i> the string that would be returned by <code>uname(2)</code> in the <i>release</i> field. Typical values might be 5.2 or 4.1.										
SI_VERSION	Copy into the array pointed to by <i>buf</i> the string that would be returned by <code>uname(2)</code> in the <i>version</i> field. The syntax and semantics of this string are defined by the system provider.										

sysinfo(2)

SI_MACHINE	Copy into the array pointed to by <i>buf</i> the string that would be returned by <code>uname(2)</code> in the <i>machine</i> field, for example, <i>sun4u</i> , <i>sun4d</i> , or <i>sun4m</i> .
SI_ARCHITECTURE	Copy into the array pointed to by <i>buf</i> a string describing the basic instruction set architecture of the current system, for example, <i>sparc</i> , <i>mc68030</i> , <i>m32100</i> , or <i>i386</i> . These names may not match predefined names in the C language compilation system.
SI_ISALIST	<p>Copy into the array pointed to by <i>buf</i> the names of the variant instruction set architectures executable on the current system.</p> <p>The names are space-separated and are ordered in the sense of best performance. That is, earlier-named instruction sets may contain more instructions than later-named instruction sets; a program that is compiled for an earlier-named instruction set will most likely run faster on this machine than the same program compiled for a later-named instruction set.</p> <p>Programs compiled for an instruction set that does not appear in the list will most likely experience performance degradation or not run at all on this machine.</p> <p>The instruction set names known to the system are listed in <code>isalist(5)</code>; these names may or may not match predefined names or compiler options in the C language compilation system.</p>
SI_PLATFORM	Copy into the array pointed to by <i>buf</i> a string describing the specific model of the hardware platform, for example, <i>SUNW</i> , <i>Sun_4_75</i> , <i>SUNW</i> , <i>SPARCsystem-600</i> , or <i>i86pc</i> .
SI_HW_PROVIDER	Copies the name of the hardware manufacturer into the array pointed to by <i>buf</i> .
SI_HW_SERIAL	Copy into the array pointed to by <i>buf</i> a string which is the text representation of the hardware-specific serial number of the physical machine on which the function is executed. Note that this may be implemented in Read-Only Memory, using software constants set when building the operating system, or by other means, and may contain non-numeric characters. It is anticipated that manufacturers will not issue the same "serial number" to more than one physical machine. The pair of strings returned by <code>SI_HW_PROVIDER</code> and

	SI_HW_SERIAL	SI_HW_SERIAL is likely to be unique across all vendor's SVR4 implementations.
	SI_SRPC_DOMAIN	Copies the Secure Remote Procedure Call domain name into the array pointed to by <i>buf</i> .
	SI_SET_SRPC_DOMAIN	Set the string to be returned by <code>sysinfo()</code> with the <code>SI_SRPC_DOMAIN</code> command to the value contained in the array pointed to by <i>buf</i> . This command requires that the calling process have the <code>PRIV_SYS_NET_CONFIG</code> privilege.
	SI_DHCP_CACHE	Copy into the array pointed to by <i>buf</i> an ASCII string consisting of the ASCII hexadecimal encoding of the name of the interface configured by <code>boot(1M)</code> followed by the DHCPACK reply from the server. This command is intended for use only by the <code>dhcpgent(1M)</code> DHCP client daemon for the purpose of adopting the DHCP maintenance of the interface configured by <code>boot</code> .
RETURN VALUES	Upon successful completion, the value returned indicates the buffer size in bytes required to hold the complete value and the terminating null character. If this value is no greater than the value passed in <i>count</i> , the entire string was copied. If this value is greater than <i>count</i> , the string copied into <i>buf</i> has been truncated to <i>count</i> - 1 bytes plus a terminating null character.	
	Otherwise, -1 is returned and <code>errno</code> is set to indicate the error.	
ERRORS	The <code>sysinfo()</code> function will fail if:	
	EFAULT	The <i>buf</i> argument does not point to a valid address.
	EINVAL	The data for a SET command exceeds the limits established by the implementation.
	EPERM	The calling process does not have the <code>PRIV_SYS_NET_CONFIG</code> privilege.
USAGE	In many cases there is no corresponding programmatic interface to set these values; such strings are typically settable only by the system administrator modifying entries in the <code>/etc/system</code> directory or the code provided by the particular OEM reading a serial number or code out of read-only memory, or hard-coded in the version of the operating system.	
	A good estimation for <i>count</i> is 257, which is likely to cover all strings returned by this interface in typical installations.	
SUMMARY OF TRUSTED SOLARIS CHANGES	The calling process must have the <code>PRIV_SYS_NET_CONFIG</code> privilege in order to perform the <code>SI_SET_HOSTNAME</code> , and <code>SI_SET_SRPC_DOMAIN</code> operations.	

sysinfo(2)

**SunOS 5.8
Reference Manual**

boot(1M), dhcpagent(1M), uname(2), gethostid(3C), gethostname(3C),
sysconf(3C), isalist(5), standards(5)

NAME	tokmapper – Manipulate kernel token mapping caches				
SYNOPSIS	<pre>cc [flags...] file... -ltsol #include <netinet/in.h> #include <sys/tiuser.h> #include <sys/tsol/tndb.h> int tokmapper(int cmd, void *buf);</pre>				
DESCRIPTION	<p>tokmapper() manipulates kernel token mapping caches. <i>cmd</i> is the operation to be performed. Currently, the only operation supported is MSIX_FLUSH, which flushes kernel token mappings for the specified MSIX host. For the MSIX_FLUSH operation, <i>buf</i> should point to a netbuf structure declared in <sys/tiuser.h>. The network address in the netbuf structure should be a sockaddr_in structure declared in <netinet/in.h>.</p> <p>To make this call successfully, a process must have the PRIV_SYS_NET_CONFIG privilege.</p>				
ATTRIBUTES	<p>See attributes(5) for descriptions of the following attributes:</p> <table> <tr> <th>ATTRIBUTE TYPE</th><th>ATTRIBUTE VALUE</th></tr> <tr> <td>Availability</td><td>SUNWtsu</td></tr> </table>	ATTRIBUTE TYPE	ATTRIBUTE VALUE	Availability	SUNWtsu
ATTRIBUTE TYPE	ATTRIBUTE VALUE				
Availability	SUNWtsu				
RETURN VALUES	<p>tokmapper() returns:</p> <p>0 On success.</p> <p>-1 On failure and sets errno to indicate the error.</p>				
ERRORS	<p>EFAULT <i>buf</i> points to an invalid address.</p> <p>EINVAL A field in the netbuf or sockaddr_in structure is invalid; or the operation specified in <i>cmd</i> is not supported.</p> <p>EPERM The process has insufficient privilege to perform the operation. To make this call successfully, a process must have the PRIV_SYS_NET_CONFIG privilege.</p>				
Trusted Solaris 8 4/01 Reference Manual	<p>tokmapd(1M), tokmapctl(1M)</p> <p>attributes(5)</p>				

uadmin(2)

NAME	uadmin – administrative control																										
SYNOPSIS	<pre>#include <sys/uadmin.h> int uadmin(int <i>cmd</i>, int <i>fcn</i>, uintptr_t <i>mdep</i>);</pre>																										
DESCRIPTION	<p>The <code>uadmin()</code> function provides control for basic administrative functions. This function is tightly coupled to the system administrative procedures and is not intended for general use. The argument <i>mdep</i> is provided for machine-dependent use and is not defined here.</p> <p>As specified by <i>cmd</i>, the following commands are available:</p> <table> <tr> <td>A_SHUTDOWN</td><td>The system is shut down. All user processes are killed, the buffer cache is flushed, and the root file system is unmounted. The action to be taken after the system has been shut down is specified by <i>fcn</i>. The functions are generic; the hardware capabilities vary on specific machines.</td></tr> <tr> <td>AD_HALT</td><td>Halt the processor(s).</td></tr> <tr> <td>AD_POWEROFF</td><td>Halt the processor(s) and turn off the power.</td></tr> <tr> <td>AD_BOOT</td><td>Reboot the system, using the kernel file.</td></tr> <tr> <td>AD_IBOOT</td><td>Interactive reboot; user is prompted for bootable program name.</td></tr> <tr> <td></td><td>The calling process must have the <code>PRIV_SYS_BOOT</code> privilege in order to perform this command.</td></tr> <tr> <td>A_REBOOT</td><td>The system stops immediately without any further processing. The action to be taken next is specified by <i>fcn</i> as above.</td></tr> <tr> <td></td><td>The calling process must have the <code>PRIV_SYS_BOOT</code> privilege in order to perform this command.</td></tr> <tr> <td>A_REMOUNT</td><td>The root file system is mounted again after having been fixed. This should be used only during the startup process.</td></tr> <tr> <td></td><td>The calling process must have the <code>PRIV_SYS_BOOT</code> privilege in order to perform this command.</td></tr> <tr> <td>A_FREEZE</td><td>Suspend the whole system. The system state is preserved in the state file. The following three subcommands are available.</td></tr> <tr> <td>AD_COMPRESS</td><td>Save the system state to the state file with compression of data.</td></tr> <tr> <td>AD_CHECK</td><td>Check if your system supports suspend and resume. Without performing a system suspend/resume, this command checks if this feature is currently available on your system.</td></tr> </table>	A_SHUTDOWN	The system is shut down. All user processes are killed, the buffer cache is flushed, and the root file system is unmounted. The action to be taken after the system has been shut down is specified by <i>fcn</i> . The functions are generic; the hardware capabilities vary on specific machines.	AD_HALT	Halt the processor(s).	AD_POWEROFF	Halt the processor(s) and turn off the power.	AD_BOOT	Reboot the system, using the kernel file.	AD_IBOOT	Interactive reboot; user is prompted for bootable program name.		The calling process must have the <code>PRIV_SYS_BOOT</code> privilege in order to perform this command.	A_REBOOT	The system stops immediately without any further processing. The action to be taken next is specified by <i>fcn</i> as above.		The calling process must have the <code>PRIV_SYS_BOOT</code> privilege in order to perform this command.	A_REMOUNT	The root file system is mounted again after having been fixed. This should be used only during the startup process.		The calling process must have the <code>PRIV_SYS_BOOT</code> privilege in order to perform this command.	A_FREEZE	Suspend the whole system. The system state is preserved in the state file. The following three subcommands are available.	AD_COMPRESS	Save the system state to the state file with compression of data.	AD_CHECK	Check if your system supports suspend and resume. Without performing a system suspend/resume, this command checks if this feature is currently available on your system.
A_SHUTDOWN	The system is shut down. All user processes are killed, the buffer cache is flushed, and the root file system is unmounted. The action to be taken after the system has been shut down is specified by <i>fcn</i> . The functions are generic; the hardware capabilities vary on specific machines.																										
AD_HALT	Halt the processor(s).																										
AD_POWEROFF	Halt the processor(s) and turn off the power.																										
AD_BOOT	Reboot the system, using the kernel file.																										
AD_IBOOT	Interactive reboot; user is prompted for bootable program name.																										
	The calling process must have the <code>PRIV_SYS_BOOT</code> privilege in order to perform this command.																										
A_REBOOT	The system stops immediately without any further processing. The action to be taken next is specified by <i>fcn</i> as above.																										
	The calling process must have the <code>PRIV_SYS_BOOT</code> privilege in order to perform this command.																										
A_REMOUNT	The root file system is mounted again after having been fixed. This should be used only during the startup process.																										
	The calling process must have the <code>PRIV_SYS_BOOT</code> privilege in order to perform this command.																										
A_FREEZE	Suspend the whole system. The system state is preserved in the state file. The following three subcommands are available.																										
AD_COMPRESS	Save the system state to the state file with compression of data.																										
AD_CHECK	Check if your system supports suspend and resume. Without performing a system suspend/resume, this command checks if this feature is currently available on your system.																										

			uadmin(2)
	A_DUMP	The system is forced to panic immediately without any further processing and a crash dump is written to the dump device (see dumpadm(1M)). The action to be taken next is specified by <i>fcn</i> as above.	
	AD_FORCE	Force AD_COMPRESS even when threads of drivers are not suspendable.	
	The calling process must have the PRIV_SYS_BOOT privilege in order to perform this command.		
RETURN VALUES	Upon successful completion, the value returned depends on <i>cmd</i> as follows:		
	A_SHUTDOWN	Never returns.	
	A_REBOOT	Never returns.	
	A_FREEZE	0 upon resume.	
	A_REMOUNT	0.	
	Otherwise, -1 is returned and <i>errno</i> is set to indicate the error.		
ERRORS	The uadmin() function will fail if:		
	EPERM	The calling process does not have sufficient privilege.	
	ENOMEM	Suspend/resume ran out of physical memory.	
	ENOSPC	Suspend/resume could not allocate enough space on the root file system to store system information.	
	ENOTSUP	Suspend/resume not supported on this platform.	
	ENXIO	Unable to successfully suspend system.	
	EBUSY	Suspend already in progress.	
	The calling process must have the PRIV_SYS_BOOT privilege in order to perform the A_FREEZE, A_REBOOT, A_REMOUNT, and A_SHUTDOWN commands.		
	uadmin(1M)		
	kernel(1M)		

SUMMARY OF TRUSTED SOLARIS CHANGES

Trusted Solaris 8 4/01 Reference Manual

Nov 03

Reference Manual

ulimit(2)

NAME	ulimit – Get and set process limits								
SYNOPSIS	<pre>#include <ulimit.h> long ulimit(int cmd, /* newlimit */...);</pre>								
DESCRIPTION	<p>The <code>ulimit()</code> function provides for control over process limits. It is effective in limiting the growth of regular files. Pipes are limited to <code>PIPE_MAX</code> bytes.</p> <p>The <i>cmd</i> values, defined in <code><ulimit.h></code>, include:</p> <table><tr><td>UL_GETFSIZE</td><td>Return the soft file size limit of the process. The limit is in units of 512-byte blocks and is inherited by child processes. Files of any size can be read. The return value is the integer part of the soft file size limit divided by 512. If the result cannot be represented as a long int, the result is unspecified.</td></tr><tr><td>UL_SETFSIZE</td><td>Set the hard and soft file size limits for output operations of the process to the value of the second argument, taken as a long int. Any process may decrease its own hard limit, but only a process with an effective <code>PRIV_SYS_CONFIG</code> privilege may increase the limit. The new file size limit is returned. The hard and soft file size limits are set to the specified value multiplied by 512. If the result would overflow an <code>rlimit_t</code>, the actual value set is unspecified.</td></tr><tr><td>UL_GMEMLIM</td><td>Get the maximum possible break value (see <code>brk(2)</code>).</td></tr><tr><td>UL_GDESLIM</td><td>Get the current value of the maximum number of open files per process configured in the system.</td></tr></table>	UL_GETFSIZE	Return the soft file size limit of the process. The limit is in units of 512-byte blocks and is inherited by child processes. Files of any size can be read. The return value is the integer part of the soft file size limit divided by 512. If the result cannot be represented as a long int, the result is unspecified.	UL_SETFSIZE	Set the hard and soft file size limits for output operations of the process to the value of the second argument, taken as a long int. Any process may decrease its own hard limit, but only a process with an effective <code>PRIV_SYS_CONFIG</code> privilege may increase the limit. The new file size limit is returned. The hard and soft file size limits are set to the specified value multiplied by 512. If the result would overflow an <code>rlimit_t</code> , the actual value set is unspecified.	UL_GMEMLIM	Get the maximum possible break value (see <code>brk(2)</code>).	UL_GDESLIM	Get the current value of the maximum number of open files per process configured in the system.
UL_GETFSIZE	Return the soft file size limit of the process. The limit is in units of 512-byte blocks and is inherited by child processes. Files of any size can be read. The return value is the integer part of the soft file size limit divided by 512. If the result cannot be represented as a long int, the result is unspecified.								
UL_SETFSIZE	Set the hard and soft file size limits for output operations of the process to the value of the second argument, taken as a long int. Any process may decrease its own hard limit, but only a process with an effective <code>PRIV_SYS_CONFIG</code> privilege may increase the limit. The new file size limit is returned. The hard and soft file size limits are set to the specified value multiplied by 512. If the result would overflow an <code>rlimit_t</code> , the actual value set is unspecified.								
UL_GMEMLIM	Get the maximum possible break value (see <code>brk(2)</code>).								
UL_GDESLIM	Get the current value of the maximum number of open files per process configured in the system.								
RETURN VALUES	Upon successful completion, <code>ulimit()</code> returns the value of the requested limit. Otherwise, <code>-1</code> is returned, the limit is not changed, and <code>errno</code> is set to indicate the error.								
ERRORS	<p>The <code>ulimit()</code> function will fail if:</p> <table><tr><td>EINVAL</td><td>The <i>cmd</i> argument is not valid.</td></tr><tr><td>EPERM</td><td>A process not having an effective <code>PRIV_SYS_CONFIG</code> privilege attempts to increase its file size limit.</td></tr></table>	EINVAL	The <i>cmd</i> argument is not valid.	EPERM	A process not having an effective <code>PRIV_SYS_CONFIG</code> privilege attempts to increase its file size limit.				
EINVAL	The <i>cmd</i> argument is not valid.								
EPERM	A process not having an effective <code>PRIV_SYS_CONFIG</code> privilege attempts to increase its file size limit.								
USAGE	<p>Since all return values are permissible in a successful situation, an application wishing to check for error situations should set <code>errno</code> to 0, then call <code>ulimit()</code>, and if it returns <code>-1</code>, check if <code>errno</code> is non-zero.</p> <p>The <code>getrlimit()</code> and <code>setrlimit()</code> functions provide a more general interface for controlling process limits, and are preferred over <code>ulimit()</code>. See <code>getrlimit(2)</code>.</p>								
SUMMARY OF TRUSTED SQUARES CHANGES	<p>The <code>PRIV_SYS_CONFIG</code> privilege is checked.</p> <p><code>getrlimit(2)</code>, <code>write(2)</code></p> <p><code>brk(2)</code></p>								

NAME	umount, umount2 – unmount a file system																				
SYNOPSIS	<pre>#include <sys/mount.h> int umount(const char *file); int umount2(const char *file, int mflag);</pre>																				
DESCRIPTION	<p>The <code>umount()</code> function requests that a previously mounted file system contained on the block special device or directory identified by <i>file</i> be unmounted. The <i>file</i> argument is a pointer to a path name. After unmounting the file system, the directory upon which the file system was mounted reverts to its ordinary interpretation.</p> <p>For all file system types except <i>namefs</i>, <code>umount()</code> may be invoked by a calling process with the <code>PRIV_SYS_MOUNT</code> privilege. For the <i>namefs</i> file system, the calling process must either be the owner of <i>file</i> or assert the <code>PRIV_FILE_OWNER</code> privilege.</p>																				
RETURN VALUES	<p><code>umount()</code> returns:</p> <p>0 On success.</p> <p>-1 On failure, and sets <code>errno</code> to indicate the error.</p>																				
ERRORS	<p>The <code>umount()</code> function will fail if:</p> <table> <tr> <td>EACCES</td><td>Search permission is denied on a component of <i>file</i>. To override this restriction, the calling process may assert one or both of these privileges: <code>PRIV_FILE_DAC_SEARCH</code> and <code>PRIV_FILE_MAC_SEARCH</code>.</td></tr> <tr> <td>EBUSY</td><td>A file on <i>file</i> is busy.</td></tr> <tr> <td>EFAULT</td><td>The file pointed to by <i>file</i> points to an illegal address.</td></tr> <tr> <td>EINVAL</td><td>The file pointed to by <i>file</i> is not mounted.</td></tr> <tr> <td>ENOENT</td><td>The file pointed to by <i>file</i> does not exist.</td></tr> <tr> <td>ELOOP</td><td>Too many symbolic links were encountered in translating the path pointed to by <i>file</i>.</td></tr> <tr> <td>ENAMETOOLONG</td><td>The length of the <i>file</i> argument exceeds <code>PATH_MAX</code>, or the length of a <i>file</i> component exceeds <code>NAME_MAX</code> while <code>_POSIX_NO_TRUNC</code> is in effect.</td></tr> <tr> <td>ENOLINK</td><td>The file pointed to by <i>file</i> is on a remote machine, and the link to that machine is no longer active.</td></tr> <tr> <td>ENOTBLK</td><td>The file pointed to by <i>file</i> is not a block special device.</td></tr> <tr> <td>EPERM</td><td>The calling process does not own <i>file</i> and <i>file</i> is a file system of type <i>namefs</i>. To override this restriction, the calling process may assert the <code>PRIV_FILE_OWNER</code>.</td></tr> </table>	EACCES	Search permission is denied on a component of <i>file</i> . To override this restriction, the calling process may assert one or both of these privileges: <code>PRIV_FILE_DAC_SEARCH</code> and <code>PRIV_FILE_MAC_SEARCH</code> .	EBUSY	A file on <i>file</i> is busy.	EFAULT	The file pointed to by <i>file</i> points to an illegal address.	EINVAL	The file pointed to by <i>file</i> is not mounted.	ENOENT	The file pointed to by <i>file</i> does not exist.	ELOOP	Too many symbolic links were encountered in translating the path pointed to by <i>file</i> .	ENAMETOOLONG	The length of the <i>file</i> argument exceeds <code>PATH_MAX</code> , or the length of a <i>file</i> component exceeds <code>NAME_MAX</code> while <code>_POSIX_NO_TRUNC</code> is in effect.	ENOLINK	The file pointed to by <i>file</i> is on a remote machine, and the link to that machine is no longer active.	ENOTBLK	The file pointed to by <i>file</i> is not a block special device.	EPERM	The calling process does not own <i>file</i> and <i>file</i> is a file system of type <i>namefs</i> . To override this restriction, the calling process may assert the <code>PRIV_FILE_OWNER</code> .
EACCES	Search permission is denied on a component of <i>file</i> . To override this restriction, the calling process may assert one or both of these privileges: <code>PRIV_FILE_DAC_SEARCH</code> and <code>PRIV_FILE_MAC_SEARCH</code> .																				
EBUSY	A file on <i>file</i> is busy.																				
EFAULT	The file pointed to by <i>file</i> points to an illegal address.																				
EINVAL	The file pointed to by <i>file</i> is not mounted.																				
ENOENT	The file pointed to by <i>file</i> does not exist.																				
ELOOP	Too many symbolic links were encountered in translating the path pointed to by <i>file</i> .																				
ENAMETOOLONG	The length of the <i>file</i> argument exceeds <code>PATH_MAX</code> , or the length of a <i>file</i> component exceeds <code>NAME_MAX</code> while <code>_POSIX_NO_TRUNC</code> is in effect.																				
ENOLINK	The file pointed to by <i>file</i> is on a remote machine, and the link to that machine is no longer active.																				
ENOTBLK	The file pointed to by <i>file</i> is not a block special device.																				
EPERM	The calling process does not own <i>file</i> and <i>file</i> is a file system of type <i>namefs</i> . To override this restriction, the calling process may assert the <code>PRIV_FILE_OWNER</code> .																				

umount(2)

file is not a file system of type *namefs* and the calling process has not asserted the `PRIV_SYS_MOUNT` privilege.

EREMOTE

The file pointed to by *file* is remote.

USAGE

The `umount ()` function may be invoked by a calling process with the appropriate privilege.

**SUMMARY OF
TRUSTED
SOLARIS
CHANGES**

Appropriate privilege is required to override access or ownership checks.

For all file system types except *namefs*, the `umount ()` system call may be invoked by a calling process with the `PRIV_SYS_MOUNT` privilege. For the *namefs* file system, the calling process must either be the owner of *file* or assert the `PRIV_FILE_OWNER` privilege.

**Trusted Solaris 8
4/01 Reference
Manual**

`mount(2)`

NAME	umount, umount2 – unmount a file system																				
SYNOPSIS	<pre>#include <sys/mount.h> int umount(const char *file); int umount2(const char *file, int mflag);</pre>																				
DESCRIPTION	<p>The <code>umount()</code> function requests that a previously mounted file system contained on the block special device or directory identified by <i>file</i> be unmounted. The <i>file</i> argument is a pointer to a path name. After unmounting the file system, the directory upon which the file system was mounted reverts to its ordinary interpretation.</p> <p>For all file system types except <i>namefs</i>, <code>umount()</code> may be invoked by a calling process with the <code>PRIV_SYS_MOUNT</code> privilege. For the <i>namefs</i> file system, the calling process must either be the owner of <i>file</i> or assert the <code>PRIV_FILE_OWNER</code> privilege.</p>																				
RETURN VALUES	<p><code>umount()</code> returns:</p> <p>0 On success.</p> <p>-1 On failure, and sets <code>errno</code> to indicate the error.</p>																				
ERRORS	<p>The <code>umount()</code> function will fail if:</p> <table> <tr> <td>EACCES</td><td>Search permission is denied on a component of <i>file</i>. To override this restriction, the calling process may assert one or both of these privileges: <code>PRIV_FILE_DAC_SEARCH</code> and <code>PRIV_FILE_MAC_SEARCH</code>.</td></tr> <tr> <td>EBUSY</td><td>A file on <i>file</i> is busy.</td></tr> <tr> <td>EFAULT</td><td>The file pointed to by <i>file</i> points to an illegal address.</td></tr> <tr> <td>EINVAL</td><td>The file pointed to by <i>file</i> is not mounted.</td></tr> <tr> <td>ENOENT</td><td>The file pointed to by <i>file</i> does not exist.</td></tr> <tr> <td>ELOOP</td><td>Too many symbolic links were encountered in translating the path pointed to by <i>file</i>.</td></tr> <tr> <td>ENAMETOOLONG</td><td>The length of the <i>file</i> argument exceeds <code>PATH_MAX</code>, or the length of a <i>file</i> component exceeds <code>NAME_MAX</code> while <code>_POSIX_NO_TRUNC</code> is in effect.</td></tr> <tr> <td>ENOLINK</td><td>The file pointed to by <i>file</i> is on a remote machine, and the link to that machine is no longer active.</td></tr> <tr> <td>ENOTBLK</td><td>The file pointed to by <i>file</i> is not a block special device.</td></tr> <tr> <td>EPERM</td><td>The calling process does not own <i>file</i> and <i>file</i> is a file system of type <i>namefs</i>. To override this restriction, the calling process may assert the <code>PRIV_FILE_OWNER</code>.</td></tr> </table>	EACCES	Search permission is denied on a component of <i>file</i> . To override this restriction, the calling process may assert one or both of these privileges: <code>PRIV_FILE_DAC_SEARCH</code> and <code>PRIV_FILE_MAC_SEARCH</code> .	EBUSY	A file on <i>file</i> is busy.	EFAULT	The file pointed to by <i>file</i> points to an illegal address.	EINVAL	The file pointed to by <i>file</i> is not mounted.	ENOENT	The file pointed to by <i>file</i> does not exist.	ELOOP	Too many symbolic links were encountered in translating the path pointed to by <i>file</i> .	ENAMETOOLONG	The length of the <i>file</i> argument exceeds <code>PATH_MAX</code> , or the length of a <i>file</i> component exceeds <code>NAME_MAX</code> while <code>_POSIX_NO_TRUNC</code> is in effect.	ENOLINK	The file pointed to by <i>file</i> is on a remote machine, and the link to that machine is no longer active.	ENOTBLK	The file pointed to by <i>file</i> is not a block special device.	EPERM	The calling process does not own <i>file</i> and <i>file</i> is a file system of type <i>namefs</i> . To override this restriction, the calling process may assert the <code>PRIV_FILE_OWNER</code> .
EACCES	Search permission is denied on a component of <i>file</i> . To override this restriction, the calling process may assert one or both of these privileges: <code>PRIV_FILE_DAC_SEARCH</code> and <code>PRIV_FILE_MAC_SEARCH</code> .																				
EBUSY	A file on <i>file</i> is busy.																				
EFAULT	The file pointed to by <i>file</i> points to an illegal address.																				
EINVAL	The file pointed to by <i>file</i> is not mounted.																				
ENOENT	The file pointed to by <i>file</i> does not exist.																				
ELOOP	Too many symbolic links were encountered in translating the path pointed to by <i>file</i> .																				
ENAMETOOLONG	The length of the <i>file</i> argument exceeds <code>PATH_MAX</code> , or the length of a <i>file</i> component exceeds <code>NAME_MAX</code> while <code>_POSIX_NO_TRUNC</code> is in effect.																				
ENOLINK	The file pointed to by <i>file</i> is on a remote machine, and the link to that machine is no longer active.																				
ENOTBLK	The file pointed to by <i>file</i> is not a block special device.																				
EPERM	The calling process does not own <i>file</i> and <i>file</i> is a file system of type <i>namefs</i> . To override this restriction, the calling process may assert the <code>PRIV_FILE_OWNER</code> .																				

umount2(2)

	<p><i>file</i> is not a file system of type <i>namefs</i> and the calling process has not asserted the <code>PRIV_SYS_MOUNT</code> privilege.</p>
	<p>EREMOTE The file pointed to by <i>file</i> is remote.</p>
USAGE	<p>The <code>umount ()</code> function may be invoked by a calling process with the appropriate privilege.</p>
SUMMARY OF TRUSTED SOLARIS CHANGES	<p>Appropriate privilege is required to override access or ownership checks.</p> <p>For all file system types except <i>namefs</i>, the <code>umount ()</code> system call may be invoked by a calling process with the <code>PRIV_SYS_MOUNT</code> privilege. For the <i>namefs</i> file system, the calling process must either be the owner of <i>file</i> or assert the <code>PRIV_FILE_OWNER</code> privilege.</p>
Trusted Solaris 8 4/01 Reference Manual	<p><code>mount(2)</code></p>

NAME	unlink – remove directory entry				
SYNOPSIS	<pre>#include <unistd.h> int unlink(const char *path);</pre>				
DESCRIPTION	<p>The <code>unlink()</code> function removes a link to a file. If <i>path</i> names a symbolic link, <code>unlink()</code> removes the symbolic link named by <i>path</i> and does not affect any file or directory named by the contents of the symbolic link. Otherwise, <code>unlink()</code> removes the link named by the pathname pointed to by <i>path</i> and decrements the link count of the file referenced by the link.</p> <p>When the file's link count becomes 0 and no process has the file open, the space occupied by the file will be freed and the file will no longer be accessible. If one or more processes have the file open when the last link is removed, the link will be removed before <code>unlink()</code> returns, but the removal of the file contents will be postponed until all references to the file are closed.</p> <p>The <i>path</i> argument must not name a directory unless the process has asserted the <code>PRIV_SYS_CONFIG</code> privilege and the implementation supports using <code>unlink()</code> on directories.</p> <p>Upon successful completion, <code>unlink()</code> will mark for update the <code>st_ctime</code> and <code>st_mtime</code> fields of the parent directory. If the file's link count is not 0, the <code>st_ctime</code> field of the file will be marked for update.</p>				
RETURN VALUES	Upon successful completion, 0 is returned. Otherwise, -1 is returned, <code>errno</code> is set to indicate the error, and the file is not unlinked.				
ERRORS	<p>The <code>unlink()</code> function will fail if:</p> <table> <tr> <td>EACCES</td><td> <p>Search permission is denied for a component of the <i>path</i> prefix.</p> <p>Write permission is denied on the directory containing the link to be removed. To override this restriction, the calling process must assert one or both of these privileges: <code>PRIV_FILE_DAC_WRITE</code> and <code>PRIV_FILE_MAC_WRITE</code>.</p> <p>The parent directory has the sticky bit set and the file is not writable by the user; or the user does not own the parent directory and the user does not own the file. To override this restriction, the calling process must assert one or more of these privileges: <code>PRIV_FILE_DAC_WRITE</code>, <code>PRIV_FILE_MAC_WRITE</code>, and <code>PRIV_FILE_OWNER</code>.</p> </td></tr> <tr> <td>EBUSY</td><td>The entry to be unlinked is the mount point for a mounted file system.</td></tr> </table>	EACCES	<p>Search permission is denied for a component of the <i>path</i> prefix.</p> <p>Write permission is denied on the directory containing the link to be removed. To override this restriction, the calling process must assert one or both of these privileges: <code>PRIV_FILE_DAC_WRITE</code> and <code>PRIV_FILE_MAC_WRITE</code>.</p> <p>The parent directory has the sticky bit set and the file is not writable by the user; or the user does not own the parent directory and the user does not own the file. To override this restriction, the calling process must assert one or more of these privileges: <code>PRIV_FILE_DAC_WRITE</code>, <code>PRIV_FILE_MAC_WRITE</code>, and <code>PRIV_FILE_OWNER</code>.</p>	EBUSY	The entry to be unlinked is the mount point for a mounted file system.
EACCES	<p>Search permission is denied for a component of the <i>path</i> prefix.</p> <p>Write permission is denied on the directory containing the link to be removed. To override this restriction, the calling process must assert one or both of these privileges: <code>PRIV_FILE_DAC_WRITE</code> and <code>PRIV_FILE_MAC_WRITE</code>.</p> <p>The parent directory has the sticky bit set and the file is not writable by the user; or the user does not own the parent directory and the user does not own the file. To override this restriction, the calling process must assert one or more of these privileges: <code>PRIV_FILE_DAC_WRITE</code>, <code>PRIV_FILE_MAC_WRITE</code>, and <code>PRIV_FILE_OWNER</code>.</p>				
EBUSY	The entry to be unlinked is the mount point for a mounted file system.				

unlink(2)

EFAULT	The <i>path</i> argument points to an illegal address.
EINTR	A signal was caught during the execution of the <code>unlink()</code> function.
ELOOP	Too many symbolic links were encountered in translating <i>path</i> .
ENAMETOOLONG	The length of the <i>path</i> argument exceeds <code>PATH_MAX</code> , or the length of a <i>path</i> component exceeds <code>NAME_MAX</code> while <code>_POSIX_NO_TRUNC</code> is in effect.
ENOENT	The named file does not exist or is a null pathname.
ENOLINK	The <i>path</i> argument points to a remote machine and the link to that machine is no longer active.
ENOTDIR	A component of the <i>path</i> prefix is not a directory.
EPERM	The named file is a directory and the calling process must assert the <code>PRIV_SYS_CONFIG</code> privilege.
EROFS	The directory entry to be unlinked is part of a read-only file system.

The `unlink()` function may fail if:

ENAMETOOLONG	Pathname resolution of a symbolic link produced an intermediate result whose length exceeds <code>PATH_MAX</code> .
ETXTBSY	The entry to be unlinked is the last directory entry to a pure procedure (shared text) file that is being executed.

USAGE Applications should use `rmdir(2)` to remove a directory.

ATTRIBUTES See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	Async-Signal-Safe

SUMMARY OF TRUSTED SOLARIS CHANGES Appropriate privilege is required to override access checks.

If the named file is a directory, the calling process must assert the `PRIV_SYS_CONFIG` privilege.

**Trusted Solaris 8
4/01 Reference
Manual** `rm(1)`, `link(2)`, `open(2)`, `rmdir(2)`
`close(2)`, `remove(3C)`, `attributes(5)`

NAME	utimes – set file access and modification times								
SYNOPSIS	<pre>#include <sys/time.h> int utimes(const char *path, const struct timeval times[2]);</pre>								
DESCRIPTION	<p>The <code>utimes()</code> function sets the access and modification times of the file pointed to by the <i>path</i> argument to the value of the <i>times</i> argument. It allows time specifications accurate to the microsecond.</p> <p>The <i>times</i> argument is an array of <code>timeval</code> structures. The first array member represents the date and time of last access, and the second member represents the date and time of last modification. The times in the <code>timeval</code> structure are measured in seconds and microseconds since the Epoch, although rounding toward the nearest second may occur.</p> <p>If the <i>times</i> argument is a null pointer, the access and modification times of the file are set to the current time. A process must be the owner of the file or must assert the <code>PRIV_FILE_OWNER</code> privilege to use this call in this manner. Upon completion, <code>utimes()</code> will mark the time of the last file status change, <code>st_ctime</code>, for update.</p>								
RETURN VALUES	<p><code>utimes()</code> returns:</p> <table> <tr> <td>0</td><td>On success.</td></tr> <tr> <td>-1</td><td>On failure, sets <code>errno</code> to indicate the error, and the file times will not be affected</td></tr> </table>	0	On success.	-1	On failure, sets <code>errno</code> to indicate the error, and the file times will not be affected				
0	On success.								
-1	On failure, sets <code>errno</code> to indicate the error, and the file times will not be affected								
ERRORS	<p>The <code>utimes()</code> function will fail if:</p> <table> <tr> <td>EACCES</td><td> <p>Search permission is denied by a component of the path prefix. To override the calling process may assert one or both of these privileges: <code>PRIV_FILE_DAC_SEARCH</code> and <code>PRIV_FILE_MAC_SEARCH</code>.</p> <p>The <i>times</i> argument is a null pointer and the effective user ID of the process does not match the owner of the file and write access is denied. To override this restriction, the calling process may assert one or both of these privileges: <code>PRIV_FILE_DAC_WRITE</code> and <code>PRIV_FILE_MAC_WRITE</code>.</p> </td></tr> <tr> <td>EFAULT</td><td>The <i>path</i> or <i>times</i> argument points to an illegal address.</td></tr> <tr> <td>EINTR</td><td>A signal was caught during the execution of the <code>utimes()</code> function.</td></tr> <tr> <td>EINVAL</td><td>The number of microseconds specified in one or both of the <code>timeval</code> structures pointed to by <i>times</i> was greater than or equal to 1,000,000 or less than 0.</td></tr> </table>	EACCES	<p>Search permission is denied by a component of the path prefix. To override the calling process may assert one or both of these privileges: <code>PRIV_FILE_DAC_SEARCH</code> and <code>PRIV_FILE_MAC_SEARCH</code>.</p> <p>The <i>times</i> argument is a null pointer and the effective user ID of the process does not match the owner of the file and write access is denied. To override this restriction, the calling process may assert one or both of these privileges: <code>PRIV_FILE_DAC_WRITE</code> and <code>PRIV_FILE_MAC_WRITE</code>.</p>	EFAULT	The <i>path</i> or <i>times</i> argument points to an illegal address.	EINTR	A signal was caught during the execution of the <code>utimes()</code> function.	EINVAL	The number of microseconds specified in one or both of the <code>timeval</code> structures pointed to by <i>times</i> was greater than or equal to 1,000,000 or less than 0.
EACCES	<p>Search permission is denied by a component of the path prefix. To override the calling process may assert one or both of these privileges: <code>PRIV_FILE_DAC_SEARCH</code> and <code>PRIV_FILE_MAC_SEARCH</code>.</p> <p>The <i>times</i> argument is a null pointer and the effective user ID of the process does not match the owner of the file and write access is denied. To override this restriction, the calling process may assert one or both of these privileges: <code>PRIV_FILE_DAC_WRITE</code> and <code>PRIV_FILE_MAC_WRITE</code>.</p>								
EFAULT	The <i>path</i> or <i>times</i> argument points to an illegal address.								
EINTR	A signal was caught during the execution of the <code>utimes()</code> function.								
EINVAL	The number of microseconds specified in one or both of the <code>timeval</code> structures pointed to by <i>times</i> was greater than or equal to 1,000,000 or less than 0.								

utimes(2)

EIO	An I/O error occurred while reading from or writing to the file system.
ELOOP	Too many symbolic links were encountered in resolving <i>path</i> .
ENAMETOOLONG	The length of the <i>path</i> argument exceeds <code>PATH_MAX</code> or a pathname component is longer than <code>NAME_MAX</code> .
ENOLINK	The <i>path</i> argument points to a remote machine and the link to that machine is no longer active.
ENOENT	A component of <i>path</i> does not name an existing file or <i>path</i> is an empty string.
ENOTDIR	A component of the path prefix is not a directory.
EPERM	The <i>times</i> argument is not a null pointer and the calling process's effective user ID has write access to the file but does not match the owner of the file and the calling process does not have the appropriate privileges. To override this restriction, the calling process may assert the <code>PRIV_FILE_OWNER</code> privilege.
EROFS	The file system containing the file is read-only.

The `utimes()` function may fail if:

ENAMETOOLONG	Path name resolution of a symbolic link produced an intermediate result whose length exceeds <code>PATH_MAX</code> .
--------------	--

SUMMARY OF TRUSTED SOLARIS CHANGES

Appropriate privilege is required to override access checks.

To change the access and modification times on a file not owned by the calling process, the calling process may assert the `PRIV_FILE_OWNER` privilege.

Trusted Solaris 8 4/01 Reference Manual

`stat(2)`

NAME	vfork – Spawn new process in a virtual memory efficient way				
SYNOPSIS	<pre>#include <unistd.h> pid_t vfork(void);</pre>				
DESCRIPTION	<p>The <code>vfork()</code> function creates new processes without fully copying the address space of the old process. This function is useful in instances where the purpose of a <code>fork(2)</code> operation would be to create a new system context for an <code>execve()</code> operation (see <code>exec(2)</code>).</p> <p>Unlike with the <code>fork()</code> function, the child process borrows the parent's memory and thread of control until a call to <code>execve()</code> or an exit (either abnormally or by a call to <code>_exit()</code> (see <code>exit(2)</code>). The parent process is suspended while the child is using its resources.</p> <p>In a multithreaded application, <code>vfork()</code> borrows only the thread of control that called <code>vfork()</code> in the parent; that is, the child contains only one thread. In that sense, <code>vfork()</code> behaves like <code>fork()</code>.</p> <p>The <code>vfork()</code> function can normally be used the same way as <code>fork()</code>. The procedure that called <code>vfork()</code>, however, should not return while running in the child's context, since the eventual return from <code>vfork()</code> would be to a stack frame that no longer exists. The <code>_exit()</code> function should be used in favor of <code>exit(3C)</code> if unable to perform an <code>execve()</code> operation, since <code>exit()</code> will flush and close standard I/O channels, and thereby corrupt the parent process's standard I/O data structures. The <code>_exit()</code> function should be used even with <code>fork()</code> to avoid flushing the buffered data twice.</p>				
RETURN VALUES	Upon successful completion, <code>vfork()</code> returns 0 to the child process and returns the process ID of the child process to the parent process. Otherwise, -1 is returned to the parent process, no child process is created, and <code>errno</code> is set to indicate the error.				
ERRORS	<p>The <code>vfork()</code> function will fail if:</p> <table> <tr> <td><code>EAGAIN</code></td><td>The system-imposed limit on the total number of processes under execution (either system-quality or by a single user) would be exceeded. Moreover, the calling process does not have the <code>PRIV_SYS_MAXPROC</code> privilege to override the limit. This limit is determined when the system is generated.</td></tr> <tr> <td><code>ENOMEM</code></td><td>There is insufficient swap space for the new process.</td></tr> </table>	<code>EAGAIN</code>	The system-imposed limit on the total number of processes under execution (either system-quality or by a single user) would be exceeded. Moreover, the calling process does not have the <code>PRIV_SYS_MAXPROC</code> privilege to override the limit. This limit is determined when the system is generated.	<code>ENOMEM</code>	There is insufficient swap space for the new process.
<code>EAGAIN</code>	The system-imposed limit on the total number of processes under execution (either system-quality or by a single user) would be exceeded. Moreover, the calling process does not have the <code>PRIV_SYS_MAXPROC</code> privilege to override the limit. This limit is determined when the system is generated.				
<code>ENOMEM</code>	There is insufficient swap space for the new process.				
SUMMARY OF TRUSTED SOLARIS CHANGES	A process with the <code>PRIV_SYS_MAXPROC</code> privilege may override the limit on the number of processes a user may have.				
Trusted Solaris 8 4/01 Reference Manual	<p><code>exec(2)</code>, <code>fork(2)</code></p> <p><code>exit(2)</code>, <code>ioctl(2)</code>, <code>wait(2)</code>, <code>exit(3C)</code></p>				

vfork(2)

NOTES	<p>The use of <code>vfork()</code> for any purpose other than as a prelude to an immediate call to a function from the <code>exec</code> family or to <code>_exit()</code> is not advised.</p> <p>The <code>vfork()</code> function is unsafe in multithreaded applications.</p> <p>This function will be eliminated in a future release. The memory sharing semantics of <code>vfork()</code> can be obtained through other mechanisms.</p> <p>To avoid a possible deadlock situation, processes that are children in the middle of a <code>vfork()</code> are never sent <code>SIGTTOU</code> or <code>SIGTTIN</code> signals; rather, output or <code>ioctl</code>s are allowed and input attempts result in an EOF indication.</p> <p>On some systems, the implementation of <code>vfork()</code> causes the parent to inherit register values from the child. This can create problems for certain optimizing compilers if <code><unistd.h></code> is not included in the source calling <code>vfork()</code>.</p>
--------------	---

NAME	write, pwrite, writev, writel, pwritel, writev1 – write on a file
SYNOPSIS	<pre> #include <unistd.h> ssize_t write(int <i>fd</i>, const void *<i>buf</i>, size_t <i>nbyte</i>); ssize_t pwrite(int <i>fd</i>, const void *<i>buf</i>, size_t <i>nbyte</i>, off_t <i>offset</i>); #include <sys/uio.h> ssize_t writev(int <i>fd</i>, const struct iovec *<i>iov</i>, int <i>iovcnt</i>); #include <tsol/rdwrl.h> ssize_t writel(int <i>fd</i>, void *<i>buf</i>, size_t <i>nbyte</i>, bclabel_t *<i>label_p</i>); ssize_t pwritel(int <i>fd</i>, void *<i>buf</i>, size_t <i>nbyte</i>, off_t <i>offset</i>, bclabel_t *<i>label_p</i>); ssize_t writev1(int <i>fd</i>, struct iovec *<i>iov</i>, int <i>iovcnt</i>, bclabel_t *<i>label_p</i>); </pre>
DESCRIPTION	<p>The <code>write()</code> function attempts to write <i>nbyte</i> bytes from the buffer pointed to by <i>buf</i> to the file associated with the open file descriptor, <i>fd</i>.</p> <p>If <i>nbyte</i> is 0, <code>write()</code> will return 0 and have no other results if the file is a regular file; otherwise, the results are unspecified.</p> <p>On a regular file or other file capable of seeking, the actual writing of data proceeds from the position in the file indicated by the file offset associated with <i>fd</i>. Before successful return from <code>write()</code>, the file offset is incremented by the number of bytes actually written. On a regular file, if this incremented file offset is greater than the length of the file, the length of the file will be set to this file offset.</p> <p>If the <code>O_SYNC</code> flag of the file status flags is set and <i>fd</i> refers to a regular file, a successful <code>write()</code> does not return until the data is delivered to the underlying hardware.</p> <p>If <i>fd</i> refers to a socket, <code>write()</code> is equivalent to <code>send(3SOCKET)</code> with no flags set.</p> <p>On a file not capable of seeking, writing always takes place starting at the current position. The value of a file offset associated with such a device is undefined.</p> <p>If the <code>O_APPEND</code> flag of the file status flags is set, the file offset will be set to the end of the file prior to each write and no intervening file modification operation will occur between changing the file offset and the write operation.</p> <p>For regular files, no data transfer will occur past the offset maximum established in the open file description with <i>fd</i>.</p> <p>A <code>write()</code> to a regular file is blocked if mandatory file/record locking is set (see <code>chmod(2)</code>), and there is a record lock owned by another process on the segment of the file to be written:</p>

write(2)

- If `O_NDELAY` or `O_NONBLOCK` is set, `write()` returns `-1` and sets `errno` to `EAGAIN`.
- If `O_NDELAY` and `O_NONBLOCK` are clear, `write()` sleeps until all blocking locks are removed or the `write()` is terminated by a signal.

If a `write()` requests that more bytes be written than there is room for—for example, if the write would exceed the process file size limit (see `getrlimit(2)` and `ulimit(2)`), the system file size limit, or the free space on the device—only as many bytes as there is room for will be written. For example, suppose there is space for 20 bytes more in a file before reaching a limit. A `write()` of 512-bytes returns 20. The next `write()` of a non-zero number of bytes gives a failure return (except as noted for pipes and FIFO below).

If `write()` is interrupted by a signal before it writes any data, it will return `-1` with `errno` set to `EINTR`.

If `write()` is interrupted by a signal after it successfully writes some data, it will return the number of bytes written.

If the value of *nbyte* is greater than `SSIZE_MAX`, the result is implementation-dependent.

After a `write()` to a regular file has successfully returned:

- Any successful `read(2)` from each byte position in the file that was modified by that write will return the data specified by the `write()` for that position until such byte positions are again modified.
- Any subsequent successful `write()` to the same byte position in the file will overwrite that file data.

Write requests to a pipe or FIFO are handled the same as a regular file with the following exceptions:

- There is no file offset associated with a pipe, hence each write request appends to the end of the pipe.
- Write requests of `{PIPE_BUF}` bytes or less are guaranteed not to be interleaved with data from other processes doing writes on the same pipe. Writes of greater than `{PIPE_BUF}` bytes may have data interleaved, on arbitrary boundaries, with writes by other processes, whether or not the `O_NONBLOCK` or `O_NDELAY` flags are set.
- If `O_NONBLOCK` and `O_NDELAY` are clear, a write request may cause the process to block, but on normal completion it returns *nbyte*.
- If `O_NONBLOCK` and `O_NDELAY` are set, `write()` does not block the process. If a `write()` request for `PIPE_BUF` or fewer bytes succeeds completely `write()` returns *nbyte*. Otherwise, if `O_NONBLOCK` is set, it returns `-1` and sets `errno` to `EAGAIN` or if `O_NDELAY` is set, it returns 0. A `write()` request for greater than `{PIPE_BUF}` bytes transfers what it can and returns the number of bytes written.

or it transfers no data and, if `O_NONBLOCK` is set, returns `-1` with `errno` set to `EAGAIN` or if `O_NDELAY` is set, it returns `0`. Finally, if a request is greater than `PIPE_BUF` bytes and all data previously written to the pipe has been read, `write()` transfers at least `PIPE_BUF` bytes.

When attempting to write to a file descriptor (other than a pipe, a FIFO, a socket, or a STREAM) that supports nonblocking writes and cannot accept the data immediately:

- If `O_NONBLOCK` and `O_NDELAY` are clear, `write()` blocks until the data can be accepted.
- If `O_NONBLOCK` or `O_NDELAY` is set, `write()` does not block the process. If some data can be written without blocking the process, `write()` writes what it can and returns the number of bytes written. Otherwise, if `O_NONBLOCK` is set, it returns `-1` and sets `errno` to `EAGAIN` or if `O_NDELAY` is set, it returns `0`.

Upon successful completion, where *nbyte* is greater than 0, `write()` will mark for update the `st_ctime` and `st_mtime` fields of the file, and if the file is a regular file, the `S_ISUID` and `S_ISGID` bits of the file mode may be cleared.

For STREAMS files (see `intro(3)` and `streamio(7I)`), the operation of `write()` is determined by the values of the minimum and maximum *nbyte* range ("packet size") accepted by the STREAM. These values are contained in the topmost STREAM module, and cannot be set or tested from user level. If *nbyte* falls within the packet size range, *nbyte* bytes are written. If *nbyte* does not fall within the range and the minimum packet size value is zero, `write()` breaks the buffer into maximum packet size segments prior to sending the data downstream (the last segment may be smaller than the maximum packet size). If *nbyte* does not fall within the range and the minimum value is non-zero, `write()` fails and sets `errno` to `ERANGE`. Writing a zero-length buffer (*nbyte* is zero) to a STREAMS device sends a zero length message with zero returned. However, writing a zero-length buffer to a pipe or FIFO sends no message and zero is returned. The user program may issue the `I_SWROPT` `ioctl(2)` to enable zero-length messages to be sent across the pipe or FIFO (see `streamio(7I)`).

When writing to a STREAM, data messages are created with a priority band of zero. When writing to a socket or to a STREAM that is not a pipe or a FIFO:

- If `O_NDELAY` and `O_NONBLOCK` are not set, and the STREAM cannot accept data (the STREAM write queue is full due to internal flow control conditions), `write()` blocks until data can be accepted.
- If `O_NDELAY` or `O_NONBLOCK` is set and the STREAM cannot accept data, `write()` returns `-1` and sets `errno` to `EAGAIN`.
- If `O_NDELAY` or `O_NONBLOCK` is set and part of the buffer has already been written when a condition occurs in which the STREAM cannot accept additional data, `write()` terminates and returns the number of bytes written.

The `write()` and `writew()` functions will fail if the STREAM head had processed an asynchronous error before the call. In this case, the value of `errno` does not reflect the result of `write()` or `writew()` but reflects the prior error.

write(2)

`pwrite()` The `pwrite()` function performs the same action as `write()`, except that it writes into a given position without changing the file pointer. The first three arguments to `pwrite()` are the same as `write()` with the addition of a fourth argument *offset* for the desired position inside the file.

`writev()` The `writev()` function performs the same action as `write()`, but gathers the output data from the *iovcnt* buffers specified by the members of the *iov* array: *iov*[0], *iov*[1], ..., *iov*[*iovcnt* - 1]. The *iovcnt* buffer is valid if greater than 0 and less than or equal to `IOV_MAX`. See `intro(2)` for a definition of `IOV_MAX`.

The `iovec` structure contains the following members:

```
caddr_t  iov_base;
int       iov_len;
```

Each `iovec` entry specifies the base address and length of an area in memory from which data should be written. The `writev()` function always writes all data from an area before proceeding to the next.

If *fdes* refers to a regular file and all of the *iov_len* members in the array pointed to by *iov* are 0, `writev()` will return 0 and have no other effect. For other file types, the behavior is unspecified.

If the sum of the *iov_len* values is greater than `SSIZE_MAX`, the operation fails and no data is transferred.

`writel()`,
`pwritel()`, and
`writevl()`

`writel()`, `pwritel()`, and `writevl()` perform the same actions as `write()`, `pwrite()`, and `writev()`, respectively, and additionally provide the CMW label *label_p* to associate with the data that is written. The label associated with the data that is written to *fd* has this restriction:

- If the descriptor refers to a file or a FIFO, then the sensitivity label portion of *label_p* is ignored.

In all other respects, the `writel()`, `pwritel()`, and `writevl()` interfaces are analogous to the `write()`, `pwrite()`, and `writev()` interfaces.

If the set-user-ID or get-group-ID bits of *fdes* are set, they are cleared by the write. The calling process may assert the `PRIV_FILE_SETID` privilege to suppress this action.

If the forced or allowed privilege sets of *fdes* are not empty, they are cleared by the write. The calling process may assert the `PRIV_FILE_SETPRIV` privilege to suppress this action.

If the public object attributes flag, `FAF_PUBLIC`, of *fdes* is set, the flag is cleared by the write. The calling process may assert the `PRIV_FILE_AUDIT` privilege to suppress this action.

	If the write causes the file-system free space to fall below its minimum level, the write fails. The calling process may assert the <code>PRIV_SYS_MINFREE</code> privilege to bypass this restriction.	
RETURN VALUES	Upon successful completion, <code>write()</code> returns the number of bytes actually written to the file associated with <i>fil</i> . This number is never greater than <i>nbyte</i> . Otherwise, <code>-1</code> is returned, the file-pointer remains unchanged, and <code>errno</code> is set to indicate the error.	
	Upon successful completion, <code>writew()</code> returns the number of bytes actually written. Otherwise, it returns <code>-1</code> , the file-pointer remains unchanged, and <code>errno</code> is set to indicate an error.	
ERRORS	The <code>write()</code> , <code>pwrite()</code> , <code>writew()</code> , <code>writel()</code> , <code>pwritel()</code> , and <code>writewl()</code> functions will fail if:	
	EAGAIN	Mandatory file/record locking is set, <code>O_NDELAY</code> or <code>O_NONBLOCK</code> is set, and there is a blocking record lock; total amount of system memory available when reading using raw I/O is temporarily insufficient; an attempt is made to write to a STREAM that can not accept data with the <code>O_NDELAY</code> or <code>O_NONBLOCK</code> flag set; or a write to a pipe or FIFO of <code>PIPE_BUF</code> bytes or less is requested and less than <i>nbytes</i> of free space is available.
	EBADF	The <i>fil</i> argument is not a valid file descriptor open for writing.
	EDEADLK	The write was going to go to sleep and cause a deadlock situation to occur.
	EDQUOT	The user's quota of disk blocks on the file system containing the file has been exhausted.
	EFAULT	The <i>buf</i> argument points to an illegal address.
	EFBIG	An attempt is made to write a file that exceeds the process's file size limit or the maximum file size (see <code>getrlimit(2)</code> and <code>ulimit(2)</code>).
		The file is a regular file, <i>nbyte</i> is greater than 0, and the starting position is greater than or equal to the offset maximum established in the file description associated with <i>fil</i> .
	EINTR	A signal was caught during the write operation and no data was transferred.
	EIO	The process is in the background and is attempting to write to its controlling terminal whose <code>TOSTOP</code> flag is set, or the process is neither ignoring nor blocking <code>SIGTTOU</code> signals and the process group of the process is orphaned.

write(2)

ENOLCK	Enforced record locking was enabled and LOCK_MAX regions are already locked in the system, or the system record lock table was full and the write could not go to sleep until the blocking record lock was removed.
ENOLINK	The <i>fildev</i> argument is on a remote machine and the link to that machine is no longer active.
ENOSPC	During a write to an ordinary file, there is no free space left on the device.
ENOSR	An attempt is made to write to a STREAMS with insufficient STREAMS memory resources available in the system.
ENXIO	A hangup occurred on the STREAM being written to.
EPIPE	An attempt is made to write to a pipe or a FIFO that is not open for reading by any process, or that has only one end open (or to a file descriptor created by <code>socket(3SOCKET)</code> , using type <code>SOCK_STREAM</code> that is no longer connected to a peer endpoint). A <code>SIGPIPE</code> signal will also be sent to the process. The process dies unless special provisions were taken to catch or ignore the signal.
ERANGE	The transfer request size was outside the range supported by the STREAMS file associated with <i>fildev</i> .

The `pwrite()` function fails and the file pointer remains unchanged if:

ESPIPE	The <i>fildev</i> argument is associated with a pipe or FIFO.
--------	---

The `writev()` function will fail if:

EINVAL	The sum of the <code>iov_len</code> values in the <i>iov</i> array would overflow an <code>ssize_t</code> .
--------	---

The `write()` and `writev()` functions may fail if:

EINVAL	The STREAM or multiplexer referenced by <i>fildev</i> is linked (directly or indirectly) downstream from a multiplexer.
ENXIO	A request was made of a non-existent device, or the request was outside the capabilities of the device.
ENXIO	A hangup occurred on the STREAM being written to.

A write to a STREAMS file may fail if an error message has been received at the STREAM head. In this case, `errno` is set to the value included in the error message.

The `writev()` function may fail if:

EINVAL	The <i>iovcnt</i> argument was less than or equal to 0 or greater than <code>IOV_MAX</code> ; one of the <code>iov_len</code> values in the <i>iov</i> array was negative; or the sum of the <code>iov_len</code> values in the <i>iov</i> array overflowed an <code>int</code> .
--------	---

write(2)

In addition, `writel()`, `pwritel()`, and `writavl()` may set `errno` to:

`EFAULT` *label_p* points outside the allocated address space of the process.
The seek pointer remains unchanged if this error occurs.

USAGE The `pwrite()` function has a transitional interface for 64-bit file offsets. See `lf64(5)`.

ATTRIBUTES See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	<code>write()</code> is Async-Signal-Safe

SUMMARY OF TRUSTED SOLARIS CHANGES

If set-user-ID or get-group-ID permission bits of *fildev* are set, they are cleared by the write. The calling process may assert the `PRIV_FILE_SETID` privilege to suppress this action.

If the forced or allowed privilege set of *fildev* is not empty, it is cleared by the write. The calling process may assert the `PRIV_FILE_SETPRIV` privilege to suppress this action.

If the public object attributes flag, `FAF_PUBLIC`, of *fildev* is set, the flag is cleared by the write. The calling process may assert the `PRIV_FILE_AUDIT` privilege to suppress this action.

If the write causes the file-system free space to fall below its minimum level, the write fails. The calling process may assert the `PRIV_SYS_MINFREE` privilege to bypass this restriction.

Mandatory and discretionary access checks have already been performed when the object was opened.

**Trusted Solaris 8
4/01 Reference
Manual** `chmod(2)`, `creat(2)`, `fcntl(2)`, `getrlimit(2)`, `lseek(2)`, `open(2)`, `ulimit(2)`,
`intro(3)`, `send(3SOCKET)`, `socket(3SOCKET)`

**SunOS 5.8
Reference Manual** `dup(2)`, `ioctl(2)`, `pipe(2)`, `attributes(5)`, `lf64(5)`, `streamio(7I)`

writel(2)

NAME	write, pwrite, writev, writel, pwritel, writevl – write on a file
SYNOPSIS	<pre>#include <unistd.h> ssize_t write(int <i>fildes</i>, const void *<i>buf</i>, size_t <i>nbyte</i>); ssize_t pwrite(int <i>fildes</i>, const void *<i>buf</i>, size_t <i>nbyte</i>, off_t <i>offset</i>); #include <sys/uio.h> ssize_t writev(int <i>fildes</i>, const struct iovec *<i>iov</i>, int <i>iovcnt</i>); #include <tsol/rdwrl.h> ssize_t writel(int <i>fildes</i>, void *<i>buf</i>, size_t <i>nbyte</i>, bclabel_t *<i>label_p</i>); ssize_t pwritel(int <i>fildes</i>, void *<i>buf</i>, size_t <i>nbyte</i>, off_t <i>offset</i>, bclabel_t *<i>label_p</i>); ssize_t writevl(int <i>fildes</i>, struct iovec *<i>iov</i>, int <i>iovcnt</i>, bclabel_t *<i>label_p</i>);</pre>
DESCRIPTION	<p>The <code>write()</code> function attempts to write <i>nbyte</i> bytes from the buffer pointed to by <i>buf</i> to the file associated with the open file descriptor, <i>fildes</i>.</p> <p>If <i>nbyte</i> is 0, <code>write()</code> will return 0 and have no other results if the file is a regular file; otherwise, the results are unspecified.</p> <p>On a regular file or other file capable of seeking, the actual writing of data proceeds from the position in the file indicated by the file offset associated with <i>fildes</i>. Before successful return from <code>write()</code>, the file offset is incremented by the number of bytes actually written. On a regular file, if this incremented file offset is greater than the length of the file, the length of the file will be set to this file offset.</p> <p>If the <code>O_SYNC</code> flag of the file status flags is set and <i>fildes</i> refers to a regular file, a successful <code>write()</code> does not return until the data is delivered to the underlying hardware.</p> <p>If <i>fildes</i> refers to a socket, <code>write()</code> is equivalent to <code>send(3SOCKET)</code> with no flags set.</p> <p>On a file not capable of seeking, writing always takes place starting at the current position. The value of a file offset associated with such a device is undefined.</p> <p>If the <code>O_APPEND</code> flag of the file status flags is set, the file offset will be set to the end of the file prior to each write and no intervening file modification operation will occur between changing the file offset and the write operation.</p> <p>For regular files, no data transfer will occur past the offset maximum established in the open file description with <i>fildes</i>.</p> <p>A <code>write()</code> to a regular file is blocked if mandatory file/record locking is set (see <code>chmod(2)</code>), and there is a record lock owned by another process on the segment of the file to be written:</p>

- If `O_NDELAY` or `O_NONBLOCK` is set, `write()` returns `-1` and sets `errno` to `EAGAIN`.
- If `O_NDELAY` and `O_NONBLOCK` are clear, `write()` sleeps until all blocking locks are removed or the `write()` is terminated by a signal.

If a `write()` requests that more bytes be written than there is room for—for example, if the write would exceed the process file size limit (see `getrlimit(2)` and `ulimit(2)`), the system file size limit, or the free space on the device—only as many bytes as there is room for will be written. For example, suppose there is space for 20 bytes more in a file before reaching a limit. A `write()` of 512-bytes returns 20. The next `write()` of a non-zero number of bytes gives a failure return (except as noted for pipes and FIFO below).

If `write()` is interrupted by a signal before it writes any data, it will return `-1` with `errno` set to `EINTR`.

If `write()` is interrupted by a signal after it successfully writes some data, it will return the number of bytes written.

If the value of *nbyte* is greater than `SSIZE_MAX`, the result is implementation-dependent.

After a `write()` to a regular file has successfully returned:

- Any successful `read(2)` from each byte position in the file that was modified by that write will return the data specified by the `write()` for that position until such byte positions are again modified.
- Any subsequent successful `write()` to the same byte position in the file will overwrite that file data.

Write requests to a pipe or FIFO are handled the same as a regular file with the following exceptions:

- There is no file offset associated with a pipe, hence each write request appends to the end of the pipe.
- Write requests of `{PIPE_BUF}` bytes or less are guaranteed not to be interleaved with data from other processes doing writes on the same pipe. Writes of greater than `{PIPE_BUF}` bytes may have data interleaved, on arbitrary boundaries, with writes by other processes, whether or not the `O_NONBLOCK` or `O_NDELAY` flags are set.
- If `O_NONBLOCK` and `O_NDELAY` are clear, a write request may cause the process to block, but on normal completion it returns *nbyte*.
- If `O_NONBLOCK` and `O_NDELAY` are set, `write()` does not block the process. If a `write()` request for `PIPE_BUF` or fewer bytes succeeds completely `write()` returns *nbyte*. Otherwise, if `O_NONBLOCK` is set, it returns `-1` and sets `errno` to `EAGAIN` or if `O_NDELAY` is set, it returns 0. A `write()` request for greater than `{PIPE_BUF}` bytes transfers what it can and returns the number of bytes written.

writel(2)

or it transfers no data and, if `O_NONBLOCK` is set, returns `-1` with `errno` set to `EAGAIN` or if `O_NDELAY` is set, it returns `0`. Finally, if a request is greater than `PIPE_BUF` bytes and all data previously written to the pipe has been read, `write()` transfers at least `PIPE_BUF` bytes.

When attempting to write to a file descriptor (other than a pipe, a FIFO, a socket, or a STREAM) that supports nonblocking writes and cannot accept the data immediately:

- If `O_NONBLOCK` and `O_NDELAY` are clear, `write()` blocks until the data can be accepted.
- If `O_NONBLOCK` or `O_NDELAY` is set, `write()` does not block the process. If some data can be written without blocking the process, `write()` writes what it can and returns the number of bytes written. Otherwise, if `O_NONBLOCK` is set, it returns `-1` and sets `errno` to `EAGAIN` or if `O_NDELAY` is set, it returns `0`.

Upon successful completion, where *nbyte* is greater than 0, `write()` will mark for update the `st_ctime` and `st_mtime` fields of the file, and if the file is a regular file, the `S_ISUID` and `S_ISGID` bits of the file mode may be cleared.

For STREAMS files (see `intro(3)` and `streamio(7I)`), the operation of `write()` is determined by the values of the minimum and maximum *nbyte* range ("packet size") accepted by the STREAM. These values are contained in the topmost STREAM module, and cannot be set or tested from user level. If *nbyte* falls within the packet size range, *nbyte* bytes are written. If *nbyte* does not fall within the range and the minimum packet size value is zero, `write()` breaks the buffer into maximum packet size segments prior to sending the data downstream (the last segment may be smaller than the maximum packet size). If *nbyte* does not fall within the range and the minimum value is non-zero, `write()` fails and sets `errno` to `ERANGE`. Writing a zero-length buffer (*nbyte* is zero) to a STREAMS device sends a zero length message with zero returned. However, writing a zero-length buffer to a pipe or FIFO sends no message and zero is returned. The user program may issue the `I_SWROPT` `ioctl(2)` to enable zero-length messages to be sent across the pipe or FIFO (see `streamio(7I)`).

When writing to a STREAM, data messages are created with a priority band of zero. When writing to a socket or to a STREAM that is not a pipe or a FIFO:

- If `O_NDELAY` and `O_NONBLOCK` are not set, and the STREAM cannot accept data (the STREAM write queue is full due to internal flow control conditions), `write()` blocks until data can be accepted.
- If `O_NDELAY` or `O_NONBLOCK` is set and the STREAM cannot accept data, `write()` returns `-1` and sets `errno` to `EAGAIN`.
- If `O_NDELAY` or `O_NONBLOCK` is set and part of the buffer has already been written when a condition occurs in which the STREAM cannot accept additional data, `write()` terminates and returns the number of bytes written.

The `write()` and `writew()` functions will fail if the STREAM head had processed an asynchronous error before the call. In this case, the value of `errno` does not reflect the result of `write()` or `writew()` but reflects the prior error.

<code>pwrite()</code>	<p>The <code>pwrite()</code> function performs the same action as <code>write()</code>, except that it writes into a given position without changing the file pointer. The first three arguments to <code>pwrite()</code> are the same as <code>write()</code> with the addition of a fourth argument <i>offset</i> for the desired position inside the file.</p>
<code>writev()</code>	<p>The <code>writev()</code> function performs the same action as <code>write()</code>, but gathers the output data from the <i>iovcnt</i> buffers specified by the members of the <i>iov</i> array: <i>iov</i>[0], <i>iov</i>[1], ..., <i>iov</i>[<i>iovcnt</i> - 1]. The <i>iovcnt</i> buffer is valid if greater than 0 and less than or equal to <code>IOV_MAX</code>. See <code>intro(2)</code> for a definition of <code>IOV_MAX</code>.</p> <p>The <code>iovec</code> structure contains the following members:</p> <pre>caddr_t iov_base; int iov_len;</pre> <p>Each <code>iovec</code> entry specifies the base address and length of an area in memory from which data should be written. The <code>writev()</code> function always writes all data from an area before proceeding to the next.</p> <p>If <i>fdes</i> refers to a regular file and all of the <i>iov_len</i> members in the array pointed to by <i>iov</i> are 0, <code>writev()</code> will return 0 and have no other effect. For other file types, the behavior is unspecified.</p> <p>If the sum of the <i>iov_len</i> values is greater than <code>SSIZE_MAX</code>, the operation fails and no data is transferred.</p>
<code>writel()</code> , <code>pwritel()</code> , and <code>writev()</code>	<p><code>writel()</code>, <code>pwritel()</code>, and <code>writev()</code> perform the same actions as <code>write()</code>, <code>pwrite()</code>, and <code>writev()</code>, respectively, and additionally provide the CMW label <i>label_p</i> to associate with the data that is written. The label associated with the data that is written to <i>fd</i> has this restriction:</p> <ul style="list-style-type: none"> ■ If the descriptor refers to a file or a FIFO, then the sensitivity label portion of <i>label_p</i> is ignored. <p>In all other respects, the <code>writel()</code>, <code>pwritel()</code>, and <code>writev()</code> interfaces are analogous to the <code>write()</code>, <code>pwrite()</code>, and <code>writev()</code> interfaces.</p> <p>If the set-user-ID or get-group-ID bits of <i>fdes</i> are set, they are cleared by the write. The calling process may assert the <code>PRIV_FILE_SETID</code> privilege to suppress this action.</p> <p>If the forced or allowed privilege sets of <i>fdes</i> are not empty, they are cleared by the write. The calling process may assert the <code>PRIV_FILE_SETPRIV</code> privilege to suppress this action.</p> <p>If the public object attributes flag, <code>FAF_PUBLIC</code>, of <i>fdes</i> is set, the flag is cleared by the write. The calling process may assert the <code>PRIV_FILE_AUDIT</code> privilege to suppress this action.</p>

writel(2)

	<p>If the write causes the file-system free space to fall below its minimum level, the write fails. The calling process may assert the <code>PRIV_SYS_MINFREE</code> privilege to bypass this restriction.</p>																		
RETURN VALUES	<p>Upon successful completion, <code>write()</code> returns the number of bytes actually written to the file associated with <i>fil</i>. This number is never greater than <i>nbyte</i>. Otherwise, <code>-1</code> is returned, the file-pointer remains unchanged, and <code>errno</code> is set to indicate the error.</p> <p>Upon successful completion, <code>writew()</code> returns the number of bytes actually written. Otherwise, it returns <code>-1</code>, the file-pointer remains unchanged, and <code>errno</code> is set to indicate an error.</p>																		
ERRORS	<p>The <code>write()</code>, <code>pwrite()</code>, <code>writew()</code>, <code>writel()</code>, <code>pwritel()</code>, and <code>writewl()</code> functions will fail if:</p> <table><tr><td>EAGAIN</td><td>Mandatory file/record locking is set, <code>O_NDELAY</code> or <code>O_NONBLOCK</code> is set, and there is a blocking record lock; total amount of system memory available when reading using raw I/O is temporarily insufficient; an attempt is made to write to a STREAM that can not accept data with the <code>O_NDELAY</code> or <code>O_NONBLOCK</code> flag set; or a write to a pipe or FIFO of <code>PIPE_BUF</code> bytes or less is requested and less than <i>nbytes</i> of free space is available.</td></tr><tr><td>EBADF</td><td>The <i>fil</i> argument is not a valid file descriptor open for writing.</td></tr><tr><td>EDEADLK</td><td>The write was going to go to sleep and cause a deadlock situation to occur.</td></tr><tr><td>EDQUOT</td><td>The user's quota of disk blocks on the file system containing the file has been exhausted.</td></tr><tr><td>EFAULT</td><td>The <i>buf</i> argument points to an illegal address.</td></tr><tr><td>EFBIG</td><td>An attempt is made to write a file that exceeds the process's file size limit or the maximum file size (see <code>getrlimit(2)</code> and <code>ulimit(2)</code>).</td></tr><tr><td></td><td>The file is a regular file, <i>nbyte</i> is greater than 0, and the starting position is greater than or equal to the offset maximum established in the file description associated with <i>fil</i>.</td></tr><tr><td>EINTR</td><td>A signal was caught during the write operation and no data was transferred.</td></tr><tr><td>EIO</td><td>The process is in the background and is attempting to write to its controlling terminal whose <code>TOSTOP</code> flag is set, or the process is neither ignoring nor blocking <code>SIGTTOU</code> signals and the process group of the process is orphaned.</td></tr></table>	EAGAIN	Mandatory file/record locking is set, <code>O_NDELAY</code> or <code>O_NONBLOCK</code> is set, and there is a blocking record lock; total amount of system memory available when reading using raw I/O is temporarily insufficient; an attempt is made to write to a STREAM that can not accept data with the <code>O_NDELAY</code> or <code>O_NONBLOCK</code> flag set; or a write to a pipe or FIFO of <code>PIPE_BUF</code> bytes or less is requested and less than <i>nbytes</i> of free space is available.	EBADF	The <i>fil</i> argument is not a valid file descriptor open for writing.	EDEADLK	The write was going to go to sleep and cause a deadlock situation to occur.	EDQUOT	The user's quota of disk blocks on the file system containing the file has been exhausted.	EFAULT	The <i>buf</i> argument points to an illegal address.	EFBIG	An attempt is made to write a file that exceeds the process's file size limit or the maximum file size (see <code>getrlimit(2)</code> and <code>ulimit(2)</code>).		The file is a regular file, <i>nbyte</i> is greater than 0, and the starting position is greater than or equal to the offset maximum established in the file description associated with <i>fil</i> .	EINTR	A signal was caught during the write operation and no data was transferred.	EIO	The process is in the background and is attempting to write to its controlling terminal whose <code>TOSTOP</code> flag is set, or the process is neither ignoring nor blocking <code>SIGTTOU</code> signals and the process group of the process is orphaned.
EAGAIN	Mandatory file/record locking is set, <code>O_NDELAY</code> or <code>O_NONBLOCK</code> is set, and there is a blocking record lock; total amount of system memory available when reading using raw I/O is temporarily insufficient; an attempt is made to write to a STREAM that can not accept data with the <code>O_NDELAY</code> or <code>O_NONBLOCK</code> flag set; or a write to a pipe or FIFO of <code>PIPE_BUF</code> bytes or less is requested and less than <i>nbytes</i> of free space is available.																		
EBADF	The <i>fil</i> argument is not a valid file descriptor open for writing.																		
EDEADLK	The write was going to go to sleep and cause a deadlock situation to occur.																		
EDQUOT	The user's quota of disk blocks on the file system containing the file has been exhausted.																		
EFAULT	The <i>buf</i> argument points to an illegal address.																		
EFBIG	An attempt is made to write a file that exceeds the process's file size limit or the maximum file size (see <code>getrlimit(2)</code> and <code>ulimit(2)</code>).																		
	The file is a regular file, <i>nbyte</i> is greater than 0, and the starting position is greater than or equal to the offset maximum established in the file description associated with <i>fil</i> .																		
EINTR	A signal was caught during the write operation and no data was transferred.																		
EIO	The process is in the background and is attempting to write to its controlling terminal whose <code>TOSTOP</code> flag is set, or the process is neither ignoring nor blocking <code>SIGTTOU</code> signals and the process group of the process is orphaned.																		

ENOLCK	Enforced record locking was enabled and LOCK_MAX regions are already locked in the system, or the system record lock table was full and the write could not go to sleep until the blocking record lock was removed.
ENOLINK	The <i>fildev</i> argument is on a remote machine and the link to that machine is no longer active.
ENOSPC	During a write to an ordinary file, there is no free space left on the device.
ENOSR	An attempt is made to write to a STREAMS with insufficient STREAMS memory resources available in the system.
ENXIO	A hangup occurred on the STREAM being written to.
EPIPE	An attempt is made to write to a pipe or a FIFO that is not open for reading by any process, or that has only one end open (or to a file descriptor created by <code>socket(3SOCKET)</code> , using type <code>SOCK_STREAM</code> that is no longer connected to a peer endpoint). A <code>SIGPIPE</code> signal will also be sent to the process. The process dies unless special provisions were taken to catch or ignore the signal.
ERANGE	The transfer request size was outside the range supported by the STREAMS file associated with <i>fildev</i> .

The `pwrite()` function fails and the file pointer remains unchanged if:

ESPIPE	The <i>fildev</i> argument is associated with a pipe or FIFO.
--------	---

The `writev()` function will fail if:

EINVAL	The sum of the <code>iov_len</code> values in the <i>iov</i> array would overflow an <code>ssize_t</code> .
--------	---

The `write()` and `writev()` functions may fail if:

EINVAL	The STREAM or multiplexer referenced by <i>fildev</i> is linked (directly or indirectly) downstream from a multiplexer.
ENXIO	A request was made of a non-existent device, or the request was outside the capabilities of the device.
ENXIO	A hangup occurred on the STREAM being written to.

A write to a STREAMS file may fail if an error message has been received at the STREAM head. In this case, `errno` is set to the value included in the error message.

The `writev()` function may fail if:

EINVAL	The <i>iovcnt</i> argument was less than or equal to 0 or greater than <code>IOV_MAX</code> ; one of the <code>iov_len</code> values in the <i>iov</i> array was negative; or the sum of the <code>iov_len</code> values in the <i>iov</i> array overflowed an <code>int</code> .
--------	---

writel(2)

In addition, `writel()`, `pwritel()`, and `writevl()` may set `errno` to:

`EFAULT` *label_p* points outside the allocated address space of the process.
The seek pointer remains unchanged if this error occurs.

USAGE The `pwrite()` function has a transitional interface for 64-bit file offsets. See `lf64(5)`.

ATTRIBUTES See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	<code>write()</code> is Async-Signal-Safe

**SUMMARY OF
TRUSTED
SOLARIS
CHANGES**

If set-user-ID or get-group-ID permission bits of *fildev* are set, they are cleared by the write. The calling process may assert the `PRIV_FILE_SETID` privilege to suppress this action.

If the forced or allowed privilege set of *fildev* is not empty, it is cleared by the write. The calling process may assert the `PRIV_FILE_SETPRIV` privilege to suppress this action.

If the public object attributes flag, `FAF_PUBLIC`, of *fildev* is set, the flag is cleared by the write. The calling process may assert the `PRIV_FILE_AUDIT` privilege to suppress this action.

If the write causes the file-system free space to fall below its minimum level, the write fails. The calling process may assert the `PRIV_SYS_MINFREE` privilege to bypass this restriction.

Mandatory and discretionary access checks have already been performed when the object was opened.

**Trusted Solaris 8
4/01 Reference
Manual**
**SunOS 5.8
Reference Manual**

`chmod(2)`, `creat(2)`, `fcntl(2)`, `getrlimit(2)`, `lseek(2)`, `open(2)`, `ulimit(2)`,
`intro(3)`, `send(3SOCKET)`, `socket(3SOCKET)`

`dup(2)`, `ioctl(2)`, `pipe(2)`, `attributes(5)`, `lf64(5)`, `streamio(7I)`

NAME	write, pwrite, writev, writel, pwriter, writev1 – write on a file
SYNOPSIS	<pre> #include <unistd.h> ssize_t write(int fildes, const void *buf, size_t nbyte); ssize_t pwrite(int fildes, const void *buf, size_t nbyte, off_t offset); #include <sys/uio.h> ssize_t writev(int fildes, const struct iovec *iov, int iovcnt); #include <tsol/rdwrl.h> ssize_t writel(int fildes, void *buf, size_t nbyte, bclabel_t *label_p); ssize_t pwriter(int fildes, void *buf, size_t nbyte, off_t offset, bclabel_t *label_p); ssize_t writev1(int fildes, struct iovec *iov, int iovcnt, bclabel_t *label_p); </pre>
DESCRIPTION	<p>The <code>write()</code> function attempts to write <i>nbyte</i> bytes from the buffer pointed to by <i>buf</i> to the file associated with the open file descriptor, <i>fildes</i>.</p> <p>If <i>nbyte</i> is 0, <code>write()</code> will return 0 and have no other results if the file is a regular file; otherwise, the results are unspecified.</p> <p>On a regular file or other file capable of seeking, the actual writing of data proceeds from the position in the file indicated by the file offset associated with <i>fildes</i>. Before successful return from <code>write()</code>, the file offset is incremented by the number of bytes actually written. On a regular file, if this incremented file offset is greater than the length of the file, the length of the file will be set to this file offset.</p> <p>If the <code>O_SYNC</code> flag of the file status flags is set and <i>fildes</i> refers to a regular file, a successful <code>write()</code> does not return until the data is delivered to the underlying hardware.</p> <p>If <i>fildes</i> refers to a socket, <code>write()</code> is equivalent to <code>send(3SOCKET)</code> with no flags set.</p> <p>On a file not capable of seeking, writing always takes place starting at the current position. The value of a file offset associated with such a device is undefined.</p> <p>If the <code>O_APPEND</code> flag of the file status flags is set, the file offset will be set to the end of the file prior to each write and no intervening file modification operation will occur between changing the file offset and the write operation.</p> <p>For regular files, no data transfer will occur past the offset maximum established in the open file description with <i>fildes</i>.</p> <p>A <code>write()</code> to a regular file is blocked if mandatory file/record locking is set (see <code>chmod(2)</code>), and there is a record lock owned by another process on the segment of the file to be written:</p>

writev(2)

- If `O_NDELAY` or `O_NONBLOCK` is set, `write()` returns `-1` and sets `errno` to `EAGAIN`.
- If `O_NDELAY` and `O_NONBLOCK` are clear, `write()` sleeps until all blocking locks are removed or the `write()` is terminated by a signal.

If a `write()` requests that more bytes be written than there is room for—for example, if the write would exceed the process file size limit (see `getrlimit(2)` and `ulimit(2)`), the system file size limit, or the free space on the device—only as many bytes as there is room for will be written. For example, suppose there is space for 20 bytes more in a file before reaching a limit. A `write()` of 512-bytes returns 20. The next `write()` of a non-zero number of bytes gives a failure return (except as noted for pipes and FIFO below).

If `write()` is interrupted by a signal before it writes any data, it will return `-1` with `errno` set to `EINTR`.

If `write()` is interrupted by a signal after it successfully writes some data, it will return the number of bytes written.

If the value of *nbyte* is greater than `SSIZE_MAX`, the result is implementation-dependent.

After a `write()` to a regular file has successfully returned:

- Any successful `read(2)` from each byte position in the file that was modified by that write will return the data specified by the `write()` for that position until such byte positions are again modified.
- Any subsequent successful `write()` to the same byte position in the file will overwrite that file data.

Write requests to a pipe or FIFO are handled the same as a regular file with the following exceptions:

- There is no file offset associated with a pipe, hence each write request appends to the end of the pipe.
- Write requests of `{PIPE_BUF}` bytes or less are guaranteed not to be interleaved with data from other processes doing writes on the same pipe. Writes of greater than `{PIPE_BUF}` bytes may have data interleaved, on arbitrary boundaries, with writes by other processes, whether or not the `O_NONBLOCK` or `O_NDELAY` flags are set.
- If `O_NONBLOCK` and `O_NDELAY` are clear, a write request may cause the process to block, but on normal completion it returns *nbyte*.
- If `O_NONBLOCK` and `O_NDELAY` are set, `write()` does not block the process. If a `write()` request for `PIPE_BUF` or fewer bytes succeeds completely `write()` returns *nbyte*. Otherwise, if `O_NONBLOCK` is set, it returns `-1` and sets `errno` to `EAGAIN` or if `O_NDELAY` is set, it returns 0. A `write()` request for greater than `{PIPE_BUF}` bytes transfers what it can and returns the number of bytes written.

or it transfers no data and, if `O_NONBLOCK` is set, returns `-1` with `errno` set to `EAGAIN` or if `O_NDELAY` is set, it returns `0`. Finally, if a request is greater than `PIPE_BUF` bytes and all data previously written to the pipe has been read, `write()` transfers at least `PIPE_BUF` bytes.

When attempting to write to a file descriptor (other than a pipe, a FIFO, a socket, or a STREAM) that supports nonblocking writes and cannot accept the data immediately:

- If `O_NONBLOCK` and `O_NDELAY` are clear, `write()` blocks until the data can be accepted.
- If `O_NONBLOCK` or `O_NDELAY` is set, `write()` does not block the process. If some data can be written without blocking the process, `write()` writes what it can and returns the number of bytes written. Otherwise, if `O_NONBLOCK` is set, it returns `-1` and sets `errno` to `EAGAIN` or if `O_NDELAY` is set, it returns `0`.

Upon successful completion, where *nbyte* is greater than 0, `write()` will mark for update the `st_ctime` and `st_mtime` fields of the file, and if the file is a regular file, the `S_ISUID` and `S_ISGID` bits of the file mode may be cleared.

For STREAMS files (see `intro(3)` and `streamio(7I)`), the operation of `write()` is determined by the values of the minimum and maximum *nbyte* range ("packet size") accepted by the STREAM. These values are contained in the topmost STREAM module, and cannot be set or tested from user level. If *nbyte* falls within the packet size range, *nbyte* bytes are written. If *nbyte* does not fall within the range and the minimum packet size value is zero, `write()` breaks the buffer into maximum packet size segments prior to sending the data downstream (the last segment may be smaller than the maximum packet size). If *nbyte* does not fall within the range and the minimum value is non-zero, `write()` fails and sets `errno` to `ERANGE`. Writing a zero-length buffer (*nbyte* is zero) to a STREAMS device sends a zero length message with zero returned. However, writing a zero-length buffer to a pipe or FIFO sends no message and zero is returned. The user program may issue the `I_SWROPT` `ioctl(2)` to enable zero-length messages to be sent across the pipe or FIFO (see `streamio(7I)`).

When writing to a STREAM, data messages are created with a priority band of zero. When writing to a socket or to a STREAM that is not a pipe or a FIFO:

- If `O_NDELAY` and `O_NONBLOCK` are not set, and the STREAM cannot accept data (the STREAM write queue is full due to internal flow control conditions), `write()` blocks until data can be accepted.
- If `O_NDELAY` or `O_NONBLOCK` is set and the STREAM cannot accept data, `write()` returns `-1` and sets `errno` to `EAGAIN`.
- If `O_NDELAY` or `O_NONBLOCK` is set and part of the buffer has already been written when a condition occurs in which the STREAM cannot accept additional data, `write()` terminates and returns the number of bytes written.

The `write()` and `writev()` functions will fail if the STREAM head had processed an asynchronous error before the call. In this case, the value of `errno` does not reflect the result of `write()` or `writev()` but reflects the prior error.

writev(2)

`pwrite()` The `pwrite()` function performs the same action as `write()`, except that it writes into a given position without changing the file pointer. The first three arguments to `pwrite()` are the same as `write()` with the addition of a fourth argument *offset* for the desired position inside the file.

`writev()` The `writev()` function performs the same action as `write()`, but gathers the output data from the *iovcnt* buffers specified by the members of the *iov* array: *iov*[0], *iov*[1], ..., *iov*[*iovcnt* - 1]. The *iovcnt* buffer is valid if greater than 0 and less than or equal to `IOV_MAX`. See `intro(2)` for a definition of `IOV_MAX`.

The `iovec` structure contains the following members:

```
caddr_t  iov_base;
int      iov_len;
```

Each `iovec` entry specifies the base address and length of an area in memory from which data should be written. The `writev()` function always writes all data from an area before proceeding to the next.

If *fdes* refers to a regular file and all of the *iov_len* members in the array pointed to by *iov* are 0, `writev()` will return 0 and have no other effect. For other file types, the behavior is unspecified.

If the sum of the *iov_len* values is greater than `SSIZE_MAX`, the operation fails and no data is transferred.

`writel()`,
`pwritel()`, and
`writevl()`

`writel()`, `pwritel()`, and `writevl()` perform the same actions as `write()`, `pwrite()`, and `writev()`, respectively, and additionally provide the CMW label *label_p* to associate with the data that is written. The label associated with the data that is written to *fd* has this restriction:

- If the descriptor refers to a file or a FIFO, then the sensitivity label portion of *label_p* is ignored.

In all other respects, the `writel()`, `pwritel()`, and `writevl()` interfaces are analogous to the `write()`, `pwrite()`, and `writev()` interfaces.

If the set-user-ID or get-group-ID bits of *fdes* are set, they are cleared by the write. The calling process may assert the `PRIV_FILE_SETID` privilege to suppress this action.

If the forced or allowed privilege sets of *fdes* are not empty, they are cleared by the write. The calling process may assert the `PRIV_FILE_SETPRIV` privilege to suppress this action.

If the public object attributes flag, `FAF_PUBLIC`, of *fdes* is set, the flag is cleared by the write. The calling process may assert the `PRIV_FILE_AUDIT` privilege to suppress this action.

	If the write causes the file-system free space to fall below its minimum level, the write fails. The calling process may assert the <code>PRIV_SYS_MINFREE</code> privilege to bypass this restriction.	
RETURN VALUES	Upon successful completion, <code>write()</code> returns the number of bytes actually written to the file associated with <i>fil</i> . This number is never greater than <i>nbyte</i> . Otherwise, <code>-1</code> is returned, the file-pointer remains unchanged, and <code>errno</code> is set to indicate the error.	
	Upon successful completion, <code>writev()</code> returns the number of bytes actually written. Otherwise, it returns <code>-1</code> , the file-pointer remains unchanged, and <code>errno</code> is set to indicate an error.	
ERRORS	The <code>write()</code> , <code>pwrite()</code> , <code>writev()</code> , <code>writel()</code> , <code>pwritel()</code> , and <code>writevl()</code> functions will fail if:	
	EAGAIN	Mandatory file/record locking is set, <code>O_NDELAY</code> or <code>O_NONBLOCK</code> is set, and there is a blocking record lock; total amount of system memory available when reading using raw I/O is temporarily insufficient; an attempt is made to write to a STREAM that can not accept data with the <code>O_NDELAY</code> or <code>O_NONBLOCK</code> flag set; or a write to a pipe or FIFO of <code>PIPE_BUF</code> bytes or less is requested and less than <i>nbytes</i> of free space is available.
	EBADF	The <i>fil</i> argument is not a valid file descriptor open for writing.
	EDEADLK	The write was going to go to sleep and cause a deadlock situation to occur.
	EDQUOT	The user's quota of disk blocks on the file system containing the file has been exhausted.
	EFAULT	The <i>buf</i> argument points to an illegal address.
	EFBIG	An attempt is made to write a file that exceeds the process's file size limit or the maximum file size (see <code>getrlimit(2)</code> and <code>ulimit(2)</code>).
		The file is a regular file, <i>nbyte</i> is greater than 0, and the starting position is greater than or equal to the offset maximum established in the file description associated with <i>fil</i> .
	EINTR	A signal was caught during the write operation and no data was transferred.
	EIO	The process is in the background and is attempting to write to its controlling terminal whose <code>TOSTOP</code> flag is set, or the process is neither ignoring nor blocking <code>SIGTTOU</code> signals and the process group of the process is orphaned.

writev(2)

ENOLCK	Enforced record locking was enabled and LOCK_MAX regions are already locked in the system, or the system record lock table was full and the write could not go to sleep until the blocking record lock was removed.
ENOLINK	The <i>fildev</i> argument is on a remote machine and the link to that machine is no longer active.
ENOSPC	During a write to an ordinary file, there is no free space left on the device.
ENOSR	An attempt is made to write to a STREAMS with insufficient STREAMS memory resources available in the system.
ENXIO	A hangup occurred on the STREAM being written to.
EPIPE	An attempt is made to write to a pipe or a FIFO that is not open for reading by any process, or that has only one end open (or to a file descriptor created by <code>socket(3SOCKET)</code> , using type <code>SOCK_STREAM</code> that is no longer connected to a peer endpoint). A <code>SIGPIPE</code> signal will also be sent to the process. The process dies unless special provisions were taken to catch or ignore the signal.
ERANGE	The transfer request size was outside the range supported by the STREAMS file associated with <i>fildev</i> .

The `pwrite()` function fails and the file pointer remains unchanged if:

ESPIPE	The <i>fildev</i> argument is associated with a pipe or FIFO.
--------	---

The `writev()` function will fail if:

EINVAL	The sum of the <code>iov_len</code> values in the <i>iov</i> array would overflow an <code>ssize_t</code> .
--------	---

The `write()` and `writev()` functions may fail if:

EINVAL	The STREAM or multiplexer referenced by <i>fildev</i> is linked (directly or indirectly) downstream from a multiplexer.
ENXIO	A request was made of a non-existent device, or the request was outside the capabilities of the device.
ENXIO	A hangup occurred on the STREAM being written to.

A write to a STREAMS file may fail if an error message has been received at the STREAM head. In this case, `errno` is set to the value included in the error message.

The `writev()` function may fail if:

EINVAL	The <i>iovcnt</i> argument was less than or equal to 0 or greater than <code>IOV_MAX</code> ; one of the <code>iov_len</code> values in the <i>iov</i> array was negative; or the sum of the <code>iov_len</code> values in the <i>iov</i> array overflowed an <code>int</code> .
--------	---

writev(2)

In addition, `writel()`, `pwritel()`, and `writevl()` may set `errno` to:

`EFAULT` *label_p* points outside the allocated address space of the process.
The seek pointer remains unchanged if this error occurs.

USAGE The `pwrite()` function has a transitional interface for 64-bit file offsets. See `lf64(5)`.

ATTRIBUTES See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	<code>write()</code> is Async-Signal-Safe

SUMMARY OF TRUSTED SOLARIS CHANGES

If set-user-ID or get-group-ID permission bits of *fildev* are set, they are cleared by the write. The calling process may assert the `PRIV_FILE_SETID` privilege to suppress this action.

If the forced or allowed privilege set of *fildev* is not empty, it is cleared by the write. The calling process may assert the `PRIV_FILE_SETPRIV` privilege to suppress this action.

If the public object attributes flag, `FAF_PUBLIC`, of *fildev* is set, the flag is cleared by the write. The calling process may assert the `PRIV_FILE_AUDIT` privilege to suppress this action.

If the write causes the file-system free space to fall below its minimum level, the write fails. The calling process may assert the `PRIV_SYS_MINFREE` privilege to bypass this restriction.

Mandatory and discretionary access checks have already been performed when the object was opened.

**Trusted Solaris 8
4/01 Reference
Manual**

`chmod(2)`, `creat(2)`, `fcntl(2)`, `getrlimit(2)`, `lseek(2)`, `open(2)`, `ulimit(2)`,
`intro(3)`, `send(3SOCKET)`, `socket(3SOCKET)`

**SunOS 5.8
Reference Manual**

`dup(2)`, `ioctl(2)`, `pipe(2)`, `attributes(5)`, `lf64(5)`, `streamio(7I)`

writev(2)

NAME	write, pwrite, writev, writel, pwritel, writevl – write on a file
SYNOPSIS	<pre>#include <unistd.h> ssize_t write(int fildes, const void *buf, size_t nbyte); ssize_t pwrite(int fildes, const void *buf, size_t nbyte, off_t offset); #include <sys/uio.h> ssize_t writev(int fildes, const struct iovec *iov, int iovcnt); #include <tsol/rdwrl.h> ssize_t writel(int fildes, void *buf, size_t nbyte, bclabel_t *label_p); ssize_t pwritel(int fildes, void *buf, size_t nbyte, off_t offset, bclabel_t *label_p); ssize_t writevl(int fildes, struct iovec *iov, int iovcnt, bclabel_t *label_p);</pre>
DESCRIPTION	<p>The <code>write()</code> function attempts to write <i>nbyte</i> bytes from the buffer pointed to by <i>buf</i> to the file associated with the open file descriptor, <i>fildes</i>.</p> <p>If <i>nbyte</i> is 0, <code>write()</code> will return 0 and have no other results if the file is a regular file; otherwise, the results are unspecified.</p> <p>On a regular file or other file capable of seeking, the actual writing of data proceeds from the position in the file indicated by the file offset associated with <i>fildes</i>. Before successful return from <code>write()</code>, the file offset is incremented by the number of bytes actually written. On a regular file, if this incremented file offset is greater than the length of the file, the length of the file will be set to this file offset.</p> <p>If the <code>O_SYNC</code> flag of the file status flags is set and <i>fildes</i> refers to a regular file, a successful <code>write()</code> does not return until the data is delivered to the underlying hardware.</p> <p>If <i>fildes</i> refers to a socket, <code>write()</code> is equivalent to <code>send(3SOCKET)</code> with no flags set.</p> <p>On a file not capable of seeking, writing always takes place starting at the current position. The value of a file offset associated with such a device is undefined.</p> <p>If the <code>O_APPEND</code> flag of the file status flags is set, the file offset will be set to the end of the file prior to each write and no intervening file modification operation will occur between changing the file offset and the write operation.</p> <p>For regular files, no data transfer will occur past the offset maximum established in the open file description with <i>fildes</i>.</p> <p>A <code>write()</code> to a regular file is blocked if mandatory file/record locking is set (see <code>chmod(2)</code>), and there is a record lock owned by another process on the segment of the file to be written:</p>

- If `O_NDELAY` or `O_NONBLOCK` is set, `write()` returns `-1` and sets `errno` to `EAGAIN`.
- If `O_NDELAY` and `O_NONBLOCK` are clear, `write()` sleeps until all blocking locks are removed or the `write()` is terminated by a signal.

If a `write()` requests that more bytes be written than there is room for—for example, if the write would exceed the process file size limit (see `getrlimit(2)` and `ulimit(2)`), the system file size limit, or the free space on the device—only as many bytes as there is room for will be written. For example, suppose there is space for 20 bytes more in a file before reaching a limit. A `write()` of 512-bytes returns 20. The next `write()` of a non-zero number of bytes gives a failure return (except as noted for pipes and FIFO below).

If `write()` is interrupted by a signal before it writes any data, it will return `-1` with `errno` set to `EINTR`.

If `write()` is interrupted by a signal after it successfully writes some data, it will return the number of bytes written.

If the value of *nbyte* is greater than `SSIZE_MAX`, the result is implementation-dependent.

After a `write()` to a regular file has successfully returned:

- Any successful `read(2)` from each byte position in the file that was modified by that write will return the data specified by the `write()` for that position until such byte positions are again modified.
- Any subsequent successful `write()` to the same byte position in the file will overwrite that file data.

Write requests to a pipe or FIFO are handled the same as a regular file with the following exceptions:

- There is no file offset associated with a pipe, hence each write request appends to the end of the pipe.
- Write requests of `{PIPE_BUF}` bytes or less are guaranteed not to be interleaved with data from other processes doing writes on the same pipe. Writes of greater than `{PIPE_BUF}` bytes may have data interleaved, on arbitrary boundaries, with writes by other processes, whether or not the `O_NONBLOCK` or `O_NDELAY` flags are set.
- If `O_NONBLOCK` and `O_NDELAY` are clear, a write request may cause the process to block, but on normal completion it returns *nbyte*.
- If `O_NONBLOCK` and `O_NDELAY` are set, `write()` does not block the process. If a `write()` request for `PIPE_BUF` or fewer bytes succeeds completely `write()` returns *nbyte*. Otherwise, if `O_NONBLOCK` is set, it returns `-1` and sets `errno` to `EAGAIN` or if `O_NDELAY` is set, it returns 0. A `write()` request for greater than `{PIPE_BUF}` bytes transfers what it can and returns the number of bytes written.

writev(2)

or it transfers no data and, if `O_NONBLOCK` is set, returns `-1` with `errno` set to `EAGAIN` or if `O_NDELAY` is set, it returns `0`. Finally, if a request is greater than `PIPE_BUF` bytes and all data previously written to the pipe has been read, `write()` transfers at least `PIPE_BUF` bytes.

When attempting to write to a file descriptor (other than a pipe, a FIFO, a socket, or a STREAM) that supports nonblocking writes and cannot accept the data immediately:

- If `O_NONBLOCK` and `O_NDELAY` are clear, `write()` blocks until the data can be accepted.
- If `O_NONBLOCK` or `O_NDELAY` is set, `write()` does not block the process. If some data can be written without blocking the process, `write()` writes what it can and returns the number of bytes written. Otherwise, if `O_NONBLOCK` is set, it returns `-1` and sets `errno` to `EAGAIN` or if `O_NDELAY` is set, it returns `0`.

Upon successful completion, where *nbyte* is greater than 0, `write()` will mark for update the `st_ctime` and `st_mtime` fields of the file, and if the file is a regular file, the `S_ISUID` and `S_ISGID` bits of the file mode may be cleared.

For STREAMS files (see `intro(3)` and `streamio(7I)`), the operation of `write()` is determined by the values of the minimum and maximum *nbyte* range ("packet size") accepted by the STREAM. These values are contained in the topmost STREAM module, and cannot be set or tested from user level. If *nbyte* falls within the packet size range, *nbyte* bytes are written. If *nbyte* does not fall within the range and the minimum packet size value is zero, `write()` breaks the buffer into maximum packet size segments prior to sending the data downstream (the last segment may be smaller than the maximum packet size). If *nbyte* does not fall within the range and the minimum value is non-zero, `write()` fails and sets `errno` to `ERANGE`. Writing a zero-length buffer (*nbyte* is zero) to a STREAMS device sends a zero length message with zero returned. However, writing a zero-length buffer to a pipe or FIFO sends no message and zero is returned. The user program may issue the `I_SWROPT` `ioctl(2)` to enable zero-length messages to be sent across the pipe or FIFO (see `streamio(7I)`).

When writing to a STREAM, data messages are created with a priority band of zero. When writing to a socket or to a STREAM that is not a pipe or a FIFO:

- If `O_NDELAY` and `O_NONBLOCK` are not set, and the STREAM cannot accept data (the STREAM write queue is full due to internal flow control conditions), `write()` blocks until data can be accepted.
- If `O_NDELAY` or `O_NONBLOCK` is set and the STREAM cannot accept data, `write()` returns `-1` and sets `errno` to `EAGAIN`.
- If `O_NDELAY` or `O_NONBLOCK` is set and part of the buffer has already been written when a condition occurs in which the STREAM cannot accept additional data, `write()` terminates and returns the number of bytes written.

The `write()` and `writev()` functions will fail if the STREAM head had processed an asynchronous error before the call. In this case, the value of `errno` does not reflect the result of `write()` or `writev()` but reflects the prior error.

<code>pwrite()</code>	<p>The <code>pwrite()</code> function performs the same action as <code>write()</code>, except that it writes into a given position without changing the file pointer. The first three arguments to <code>pwrite()</code> are the same as <code>write()</code> with the addition of a fourth argument <i>offset</i> for the desired position inside the file.</p>
<code>writev()</code>	<p>The <code>writev()</code> function performs the same action as <code>write()</code>, but gathers the output data from the <i>iovcnt</i> buffers specified by the members of the <i>iov</i> array: <i>iov</i>[0], <i>iov</i>[1], ..., <i>iov</i>[<i>iovcnt</i> - 1]. The <i>iovcnt</i> buffer is valid if greater than 0 and less than or equal to <code>IOV_MAX</code>. See <code>intro(2)</code> for a definition of <code>IOV_MAX</code>.</p> <p>The <code>iovec</code> structure contains the following members:</p> <pre>caddr_t iov_base; int iov_len;</pre> <p>Each <code>iovec</code> entry specifies the base address and length of an area in memory from which data should be written. The <code>writev()</code> function always writes all data from an area before proceeding to the next.</p> <p>If <i>fdes</i> refers to a regular file and all of the <i>iov_len</i> members in the array pointed to by <i>iov</i> are 0, <code>writev()</code> will return 0 and have no other effect. For other file types, the behavior is unspecified.</p> <p>If the sum of the <i>iov_len</i> values is greater than <code>SSIZE_MAX</code>, the operation fails and no data is transferred.</p>
<code>writel()</code> , <code>pwritel()</code> , and <code>writev()</code>	<p><code>writel()</code>, <code>pwritel()</code>, and <code>writev()</code> perform the same actions as <code>write()</code>, <code>pwrite()</code>, and <code>writev()</code>, respectively, and additionally provide the CMW label <i>label_p</i> to associate with the data that is written. The label associated with the data that is written to <i>fd</i> has this restriction:</p> <ul style="list-style-type: none"> ■ If the descriptor refers to a file or a FIFO, then the sensitivity label portion of <i>label_p</i> is ignored. <p>In all other respects, the <code>writel()</code>, <code>pwritel()</code>, and <code>writev()</code> interfaces are analogous to the <code>write()</code>, <code>pwrite()</code>, and <code>writev()</code> interfaces.</p> <p>If the set-user-ID or get-group-ID bits of <i>fdes</i> are set, they are cleared by the write. The calling process may assert the <code>PRIV_FILE_SETID</code> privilege to suppress this action.</p> <p>If the forced or allowed privilege sets of <i>fdes</i> are not empty, they are cleared by the write. The calling process may assert the <code>PRIV_FILE_SETPRIV</code> privilege to suppress this action.</p> <p>If the public object attributes flag, <code>FAF_PUBLIC</code>, of <i>fdes</i> is set, the flag is cleared by the write. The calling process may assert the <code>PRIV_FILE_AUDIT</code> privilege to suppress this action.</p>

writev(2)

	<p>If the write causes the file-system free space to fall below its minimum level, the write fails. The calling process may assert the <code>PRIV_SYS_MINFREE</code> privilege to bypass this restriction.</p>																		
RETURN VALUES	<p>Upon successful completion, <code>write()</code> returns the number of bytes actually written to the file associated with <i>fdes</i>. This number is never greater than <i>nbyte</i>. Otherwise, <code>-1</code> is returned, the file-pointer remains unchanged, and <code>errno</code> is set to indicate the error.</p> <p>Upon successful completion, <code>writev()</code> returns the number of bytes actually written. Otherwise, it returns <code>-1</code>, the file-pointer remains unchanged, and <code>errno</code> is set to indicate an error.</p>																		
ERRORS	<p>The <code>write()</code>, <code>pwrite()</code>, <code>writev()</code>, <code>writel()</code>, <code>pwritel()</code>, and <code>writev1()</code> functions will fail if:</p> <table><tr><td>EAGAIN</td><td>Mandatory file/record locking is set, <code>O_NDELAY</code> or <code>O_NONBLOCK</code> is set, and there is a blocking record lock; total amount of system memory available when reading using raw I/O is temporarily insufficient; an attempt is made to write to a STREAM that can not accept data with the <code>O_NDELAY</code> or <code>O_NONBLOCK</code> flag set; or a write to a pipe or FIFO of <code>PIPE_BUF</code> bytes or less is requested and less than <i>nbytes</i> of free space is available.</td></tr><tr><td>EBADF</td><td>The <i>fdes</i> argument is not a valid file descriptor open for writing.</td></tr><tr><td>EDEADLK</td><td>The write was going to go to sleep and cause a deadlock situation to occur.</td></tr><tr><td>EDQUOT</td><td>The user's quota of disk blocks on the file system containing the file has been exhausted.</td></tr><tr><td>EFAULT</td><td>The <i>buf</i> argument points to an illegal address.</td></tr><tr><td>EFBIG</td><td>An attempt is made to write a file that exceeds the process's file size limit or the maximum file size (see <code>getrlimit(2)</code> and <code>ulimit(2)</code>).</td></tr><tr><td></td><td>The file is a regular file, <i>nbyte</i> is greater than 0, and the starting position is greater than or equal to the offset maximum established in the file description associated with <i>fdes</i>.</td></tr><tr><td>EINTR</td><td>A signal was caught during the write operation and no data was transferred.</td></tr><tr><td>EIO</td><td>The process is in the background and is attempting to write to its controlling terminal whose <code>TOSTOP</code> flag is set, or the process is neither ignoring nor blocking <code>SIGTTOU</code> signals and the process group of the process is orphaned.</td></tr></table>	EAGAIN	Mandatory file/record locking is set, <code>O_NDELAY</code> or <code>O_NONBLOCK</code> is set, and there is a blocking record lock; total amount of system memory available when reading using raw I/O is temporarily insufficient; an attempt is made to write to a STREAM that can not accept data with the <code>O_NDELAY</code> or <code>O_NONBLOCK</code> flag set; or a write to a pipe or FIFO of <code>PIPE_BUF</code> bytes or less is requested and less than <i>nbytes</i> of free space is available.	EBADF	The <i>fdes</i> argument is not a valid file descriptor open for writing.	EDEADLK	The write was going to go to sleep and cause a deadlock situation to occur.	EDQUOT	The user's quota of disk blocks on the file system containing the file has been exhausted.	EFAULT	The <i>buf</i> argument points to an illegal address.	EFBIG	An attempt is made to write a file that exceeds the process's file size limit or the maximum file size (see <code>getrlimit(2)</code> and <code>ulimit(2)</code>).		The file is a regular file, <i>nbyte</i> is greater than 0, and the starting position is greater than or equal to the offset maximum established in the file description associated with <i>fdes</i> .	EINTR	A signal was caught during the write operation and no data was transferred.	EIO	The process is in the background and is attempting to write to its controlling terminal whose <code>TOSTOP</code> flag is set, or the process is neither ignoring nor blocking <code>SIGTTOU</code> signals and the process group of the process is orphaned.
EAGAIN	Mandatory file/record locking is set, <code>O_NDELAY</code> or <code>O_NONBLOCK</code> is set, and there is a blocking record lock; total amount of system memory available when reading using raw I/O is temporarily insufficient; an attempt is made to write to a STREAM that can not accept data with the <code>O_NDELAY</code> or <code>O_NONBLOCK</code> flag set; or a write to a pipe or FIFO of <code>PIPE_BUF</code> bytes or less is requested and less than <i>nbytes</i> of free space is available.																		
EBADF	The <i>fdes</i> argument is not a valid file descriptor open for writing.																		
EDEADLK	The write was going to go to sleep and cause a deadlock situation to occur.																		
EDQUOT	The user's quota of disk blocks on the file system containing the file has been exhausted.																		
EFAULT	The <i>buf</i> argument points to an illegal address.																		
EFBIG	An attempt is made to write a file that exceeds the process's file size limit or the maximum file size (see <code>getrlimit(2)</code> and <code>ulimit(2)</code>).																		
	The file is a regular file, <i>nbyte</i> is greater than 0, and the starting position is greater than or equal to the offset maximum established in the file description associated with <i>fdes</i> .																		
EINTR	A signal was caught during the write operation and no data was transferred.																		
EIO	The process is in the background and is attempting to write to its controlling terminal whose <code>TOSTOP</code> flag is set, or the process is neither ignoring nor blocking <code>SIGTTOU</code> signals and the process group of the process is orphaned.																		

ENOLCK	Enforced record locking was enabled and LOCK_MAX regions are already locked in the system, or the system record lock table was full and the write could not go to sleep until the blocking record lock was removed.
ENOLINK	The <i>fildev</i> argument is on a remote machine and the link to that machine is no longer active.
ENOSPC	During a write to an ordinary file, there is no free space left on the device.
ENOSR	An attempt is made to write to a STREAMS with insufficient STREAMS memory resources available in the system.
ENXIO	A hangup occurred on the STREAM being written to.
EPIPE	An attempt is made to write to a pipe or a FIFO that is not open for reading by any process, or that has only one end open (or to a file descriptor created by <code>socket(3SOCKET)</code> , using type <code>SOCK_STREAM</code> that is no longer connected to a peer endpoint). A <code>SIGPIPE</code> signal will also be sent to the process. The process dies unless special provisions were taken to catch or ignore the signal.
ERANGE	The transfer request size was outside the range supported by the STREAMS file associated with <i>fildev</i> .

The `pwrite()` function fails and the file pointer remains unchanged if:

ESPIPE	The <i>fildev</i> argument is associated with a pipe or FIFO.
--------	---

The `writev()` function will fail if:

EINVAL	The sum of the <code>iov_len</code> values in the <i>iov</i> array would overflow an <code>ssize_t</code> .
--------	---

The `write()` and `writev()` functions may fail if:

EINVAL	The STREAM or multiplexer referenced by <i>fildev</i> is linked (directly or indirectly) downstream from a multiplexer.
ENXIO	A request was made of a non-existent device, or the request was outside the capabilities of the device.
ENXIO	A hangup occurred on the STREAM being written to.

A write to a STREAMS file may fail if an error message has been received at the STREAM head. In this case, `errno` is set to the value included in the error message.

The `writev()` function may fail if:

EINVAL	The <i>iovcnt</i> argument was less than or equal to 0 or greater than <code>IOV_MAX</code> ; one of the <code>iov_len</code> values in the <i>iov</i> array was negative; or the sum of the <code>iov_len</code> values in the <i>iov</i> array overflowed an <code>int</code> .
--------	---

writev(2)

In addition, `writel()`, `pwritel()`, and `writev()` may set `errno` to:

`EFAULT` *label_p* points outside the allocated address space of the process.
The seek pointer remains unchanged if this error occurs.

USAGE The `pwrite()` function has a transitional interface for 64-bit file offsets. See `lf64(5)`.

ATTRIBUTES See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT-Level	<code>write()</code> is Async-Signal-Safe

**SUMMARY OF
TRUSTED
SOLARIS
CHANGES**

If set-user-ID or get-group-ID permission bits of *fildev* are set, they are cleared by the write. The calling process may assert the `PRIV_FILE_SETID` privilege to suppress this action.

If the forced or allowed privilege set of *fildev* is not empty, it is cleared by the write. The calling process may assert the `PRIV_FILE_SETPRIV` privilege to suppress this action.

If the public object attributes flag, `FAF_PUBLIC`, of *fildev* is set, the flag is cleared by the write. The calling process may assert the `PRIV_FILE_AUDIT` privilege to suppress this action.

If the write causes the file-system free space to fall below its minimum level, the write fails. The calling process may assert the `PRIV_SYS_MINFREE` privilege to bypass this restriction.

Mandatory and discretionary access checks have already been performed when the object was opened.

**Trusted Solaris 8
4/01 Reference
Manual**

**SunOS 5.8
Reference Manual**

`chmod(2)`, `creat(2)`, `fcntl(2)`, `getrlimit(2)`, `lseek(2)`, `open(2)`, `ulimit(2)`,
`intro(3)`, `send(3SOCKET)`, `socket(3SOCKET)`

`dup(2)`, `ioctl(2)`, `pipe(2)`, `attributes(5)`, `lf64(5)`, `streamio(7I)`

Index

A

- access — determine accessibility of a file, 48
- Access Control List, 29
- access permission mode of file
 - change — `chmod`, 66, 133
- accounting
 - enable or disable process accounting — `acct`, 50
- accreditation range, 29
- `acct` — enable or disable process accounting, 50
- `acl` — get or set a file's Access Control List (ACL), 51, 129
- ACL, 28
- ACL Mask, 29
- `adjtime` — correct the time to allow synchronization of the system clock, 53
- audit — write an audit record, 55
- `auditon` — manipulate auditing, 57
- `auditsvc` — write audit log to specified file descriptor, 62

B

- bind LWPs to a processor — `processor_bind`, 335

C

- change
 - CMW label of file — `setcmwlabel`, 181, 261, 404
 - view of a host state between labeled and unlabeled — `chstate`, 75
- `chdir` — change working directory, 64, 131
- `chmod` — change access permission mode of file, 66, 133
- `chown` — change owner and group of a file, 70, 137, 251
- `chroot` — change root directory, 73, 140
- `chstate` — change the view of a host state between labeled and unlabeled, 75
- classification, 29
- clearance, 29
- CMW label, 29
- CMW label of file
 - `fgetcmwlabel`, 153, 207, 254
 - `getcmwlabel`, 153, 207, 254
 - `lgetcmwlabel`, 153, 207, 254
- compartment, 29
- `creat` — create a new file or rewrite an existing one, 76
- create a new process — `fork`, 169, 173
 - `fork1`, 169, 173

D

- DAC, 30
- determine accessibility of a file — `access`, 48

- files
 - change access permission mode of file — `chmod`, 66, 70, 133, 137, 251
 - change the name of a file — `rename`, 377
 - create a new file or rewrite an existing one — `creat`, 76
 - execute — `exec`, 80, 87, 94, 101, 108, 115, 122
 - get configurable pathname variables — `pathconf`, 177, 305
 - link to a file — `link`, 256
 - move read/write file pointer — `lseek`, 258, 259
 - set security flags of a file — `setfattrflag`, 155, 185, 212, 276, 280, 413
- `fork` — create a new process, 169, 173
- `fork`
 - spawn new process in a virtual memory efficient way — `vfork`, 487
- `fork1` — create a new process, 169, 173
- `fpathconf` — get configurable pathname variables, 177, 305
- `fsetcmwlabel` — set CMW label of a file, 181, 261, 404
- `fsetfattrflag` — set security flags of a file, 155, 185, 212, 276, 280, 413
- `fsetfpriv` — set a privilege set associated with a file, 159, 189, 216, 417
- `fstat` — get status on open file known by file descriptor, 192, 265, 457
- `fstatvfs` — get file system information, 196, 461

G

- get and set process limits — `ulimit`, 478
- get or change processor operational status — `p_online`, 309
- get or set a file's Access Control List (ACL)
 - `acl`, 51, 129
 - `facl`, 51, 129
- get security configuration information — `seconf`, 382
- `getaudit` — get process audit information, 199, 201, 398, 400
- `getclearance` — obtain process clearance, 204
- `getcmwfsrange` — get file system sensitivity label range, 151, 205

- `getcmwlabel` — get file CMW label, 153, 207, 254
- `getcmwplabel` — get process CMW label, 209
- `getdents` — read directory entries and put in a file system independent format, 210
- `getfattrflag` — get the security attribute flags of a file, 155, 185, 212, 276, 280, 413
- `getfpriv` — return a privilege set associated with a file, 159, 189, 216, 417
- `getfsattr` — get file system security attributes, 162, 219
- `getgroups` — get supplementary group access list IDs, 221, 422
- `getmldadorn` — get file system MLD adornment, 164, 223
- `getmsgqcmwlabel` — get the CMW labels associated with System V IPC structures, 225, 243, 244
- `getpattr` — get process attribute flags, 226, 424
- `getpgid` — get process group IDs, 229, 231, 233, 235
- `getpgrp` — get process group IDs, 229, 231, 233, 235
- `getpid` — get process IDs, 229, 231, 233, 235
- `getppid` — get parent process IDs, 229, 231, 233, 235
- `getppriv` — return a privilege set associated with the invoking process, 237, 427
- `getrlimit` — control maximum system resource consumption, 239, 431
- `getsemcmwlabel` — get the CMW labels associated with System V IPC structures, 225, 243, 244
- `getshmcmwlabel` — get the CMW labels associated with System V IPC structures, 225, 243, 244
- `getsid` — get session ID, 245
- `getslldname` — get file system SLD name, 166, 246
- group ID
 - set real and effective — `setregid`, 429
- group IDs
 - set — `setgid`, 409, 411, 420, 435
 - supplementary group access list IDs — `getgroups`, `setgroups`, 221, 422

H

halt system — uadmin, 476

I

inheritable privileges, 34

I/O

audit — audit, 55

K

kill — send a signal to a process or a group of processes, 249

L

label, 34

label range, 34

label translation flags, 34

label view flags, 34

lchown — change owner and group of a file, 70, 137, 251

lgetcmwlabel — get file CMW label, 153, 207, 254

link — link to a file, 256

link

remove — unlink, 483

link, symbolic

make one to a file — symlink, 469

lseek — move extended read/write file pointer, 258

llseek — move extended read/write file pointer, 258

lseek — move read/write file pointer, 259

lsetcmwlabel — set CMW label of a file, 181, 261, 404

lstat — get status on symbolic link file, 192, 265, 457

LWP

scheduler control — priocntl, 324

M

MAC, 35

make a directory, or a special or ordinary file —
mknod, 272

mandatory access control, 35

manipulate auditing — auditon, 57

memory, shared

control operations — shmctl, 440

get segment identifier — shmget, 446, 448

operations — shmop, 437, 443, 450

message control operations

— msgctl, 288

message queue

get — msgget, 290, 292

message receive operation — msgrcv, 294

message send operation — msgsnd, 296

mkdir — make a directory, 269

mknod — make a directory, or a special or
ordinary file, 272

MLD, 35

MLD adornment of file system

fgetmldadorn, getmldadorn, 164, 223

mldgetattrflag — get the security attribute flags
of a file, 155, 185, 212, 276, 280, 413

mldsetattrflag — set security flags of a
file, 155, 185, 212, 276, 280, 413

mount — mount a file system, 284

mount a file system — mount, 284

msgctl — message control operations, 288

msgget — get message queue, 290, 292

msgrcv — message receive operation, 294

msgsnd — message send operation, 296

multilevel directory, 37

N

network endpoint objects, 37

nice — change priority of a time-sharing
process, 298

O

object, 37

open — open a file, 299

open a file — open, 299

owner of file
change — `chown`, 70, 137, 251

P

`pathconf` — get configurable pathname variables, 177, 305
`pathname`
get configurable variables — `pathconf`, 177, 305
`p_online` — get or change processor operational status, 309
`pread` — read from a file, 312, 318, 351, 357, 365, 371
`preadl` — read from a file, 312, 318, 351, 357, 365, 371
`prctl` — process scheduler control, 324
`prctlset` — generalized process scheduler control, 333
privilege debugging flag, 38
process accounting
enable or disable — `acct`, 50
process attribute flags, 38
get — `getpattr`, 226, 424
set — `setpattr`, 226, 424
process audit information
set process audit information — `setaudit`, 199, 201, 398, 400
process CMW label — `getcmwplabel`, 209
process objects, 39
process privilege sets, 39
process scheduler
control — `prctl`, 324
generalized control — `prctlset`, 333
process security attribute, 40
process, time-sharing
change priority — `nice`, 298
processes
change priority of a time-sharing process — `nice`, 298
create a new one — `fork`, 169, 173
execute a file — `exec`, 80, 87, 94, 101, 108, 115, 122
generalized scheduler control — `prctlset`, 333

processes (*continued*)

get identification — `getpid`, `getpgrp`, `getppid`, `getpgid`, 229, 231, 233, 235
get or set session ID — `getsid`, `setsid`, 245
read from a file — `read`, 312, 318, 351, 357, 365, 371
read directory entries and put in a file system independent format — `getdents`, 210
send a signal to a process or a group of processes — `kill`, 249
spawn new process in a virtual memory efficient way — `vfork`, 487
supplementary group access list IDs — `getgroups`, `setgroups`, 221, 422
processes and protection
set group IDs — `setregid`, 429
set user IDs — `setreuid`, 430
`processor_bind` — bind LWPs to a processor, 335
`pwrite` — write on a file, 340, 347, 492, 499, 506, 513
`p writel` — write on a file, 340, 347, 492, 499, 506, 513

R

read from a file
`pread`, `preadl`, `read`, `readv`, `readvl`, 312, 318, 351, 357, 365, 371
`read` — read from a file, 312, 318, 351, 357, 365, 371
read the contents of a symbolic link — `readlink`, 363
`readl` — read from a file, 312, 318, 351, 357, 365, 371
`readlink` — read the contents of a symbolic link, 363
`readv` — read from a file, 312, 318, 351, 357, 365, 371
`readvl` — read from a file, 312, 318, 351, 357, 365, 371
read/write file pointer
move — `lseek`, 258, 259
real group ID
set — `setregid`, 429

- real user IDs
 - set — setreuid, 430
- reboot system
 - uadmin, 476
- remount root file system
 - uadmin, 476
- rename — change the name of a file, 377
- rmdir — remove a directory, 380
- root directory
 - change — chroot, 73, 140

S

- seccomp — get security configuration information, 382
- security attribute, 40
- security attributes of file system
 - fgetfsattr, getfsattr, 162, 219
- security flags of a file
 - set — setfattrflag, 155, 185, 212, 276, 280, 413
- security policy, 41
- semaphore operations — semop, 390, 394
- semaphores
 - control operations — semctl, 383
 - get a set — semget, 386, 388
- semctl — semaphore control operations, 383
- semget — get set of semaphores, 386, 388
- semop — semaphore operations, 390, 394
- semtimedop — semaphore operations, 390, 394
- sensitivity label, 42
- sensitivity label of file system
 - fgetcmwfsrange, 151, 205
 - getcmwfsrange, 151, 205
- session ID
 - get or set — getsid, setsid, 245
- set file access and modification times — utimes, 485
- set process CMW label — setcmwplabel, 408
- setaudit — set process audit information, 199, 201, 398, 400
- setclearance — set process clearance, 403
- setcmwlabel — set CMW label of a file, 181, 261, 404
- setcmwplabel — set process CMW label, 408

- setegid — set effective group ID, 409, 411, 420, 435
- seteuid — set effective user ID, 409, 411, 420, 435
- setfattrflag — set security flags of file, 155, 185, 212, 276, 280, 413
- setfpriv — set a privilege set associated with a file, 159, 189, 216, 417
- setgid — set group ID, 409, 411, 420, 435
- setgroups — set supplementary group access list IDs, 221, 422
- setpattr — set process attribute flags, 226, 424
- setppriv — assign a privilege set associated with the invoking process, 237, 427
- setregid — set real and effective group ID, 429
- setreuid — set real and effective user IDs, 430
- setrlimit — control maximum system resource consumption, 239, 431
- setsid — set session ID, 245
- setuid — set user ID, 409, 411, 420, 435
- shared memory
 - control operations — shmctl, 440
 - get segment identifier — shmget, 446, 448
 - operations — shmop, 437, 443, 450
- shmctl — shared memory control operations, 440
- shmget — get shared memory segment identifier, 446, 448
- shmgetl — get shared memory segment identifier, 446, 448
- shmop — shared memory operations, 437, 443, 450
- shutdown
 - uadmin, 476
- sigsend — send a signal to a process or a group of processes, 453, 455
- sigsendset — provides an alternate interface to sigsend for sending signals to sets of processes, 453, 455
- single-level directory, 44
- SLD, 40
- SLD name of file system
 - fgetslldname, 166, 246
 - getslldname, 166, 246
- special files
 - create a new one — mknod, 272
- stat — get file status, 192, 265, 457

- statvfs — get file system information, 196, 461
- stime — set system time and date, 464
- strictly dominate, 45
- swap space
 - manage — swapctl, 465
- swapctl — manage swap space, 465
- symbolic link
 - make one to a file — symlink, 469
- symlink — make a symbolic link to a file, 469
- sysinfo — get and set system information strings, 471
- system accreditation range, 45
- system administration
 - administrative control — uadmin, 476
- system clock
 - synchronization — adjtime, 53
- system information
 - get and set strings — sysinfo, 471
- system resources
 - control maximum system resource consumption — getrlimit, setrlimit, 239, 431

T

- time
 - correct the time to allow synchronization of the system clock — adjtime, 53
 - set system time and date — stime, 464
- tokmapper — manipulates kernel token mapping caches, 475
- trusted path flag, 45

U

- uadmin — administrative control, 476
- ulimit — get and set process limits, 478
- umount — unmount a file system, 479, 481
- unlink — remove directory entry, 483
- unmount a file system — umount, 479, 481
- user audit identity
 - get process audit information — getaudit, 199, 201, 398, 400
- user IDs
 - set — setuid, 409, 411, 420, 435

- user IDs (*continued*)
 - set real and effective — setreuid, 430
- utimes — set file access and modification times, 485

V

- vfork — spawn new process in a virtual memory efficient way, 487

W

- write — write on a file, 337, 344, 489, 496, 503, 510
- write on a file
 - pwrite, pwrite, write, writel, writev, writev, 337, 344, 489, 496, 503, 510
- writel — write on a file, 340, 347, 492, 499, 506, 513
- writev — write on a file, 340, 347, 492, 499, 506, 513
- writev — write on a file, 340, 347, 492, 499, 506, 513

