



ChorusOS 4.0.1 Simulator for the Solaris Operating Environment (SPARC Platform Edition) User's Guide

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

Part Number 806-4654-10
July 2000

Copyright 2000 Sun Microsystems, Inc. 901 San Antonio Road, Palo Alto, California 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, ChorusOS, Sun Embedded WorkShop, Solstice, JDMK, 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 2000 Sun Microsystems, Inc. 901 San Antonio Road, Palo Alto, Californie 94303-4900 Etats-Unis. 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, ChorusOS, Sun Embedded WorkShop, Solstice, JDMK, 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 REpondre 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.



Contents

	Preface	11
1.	ChorusOS 4.0.1 Simulator Architecture	15
	Boot Mechanism	15
	Remote Inter-Process Communication	16
	Network Architecture	17
2.	Running the ChorusOS 4.0.1 Simulator	19
	Planning Your Configuration	19
	Getting Started Quickly	20
	Enabling Dynamic Configuration	22
	Configuring the Solaris Ethernet Pseudo-driver	23
	Running the Solaris Ethernet Pseudo-driver	23
	Configuring the Simulator IP Address	24
	Configuring the Gateway for the Simulator	25
	Configuring IP Forwarding for Your Host	26
	Booting the Simulator System Image	26
	Accessing the Simulator From Your Host	28
	Accessing the Simulator from Remote Hosts	30
3.	BSP for the ChorusOS 4.0.1 Simulator	33
	UNIX Signals	33

	Use of UNIX Services	35
	Retrieving UNIX Function Addresses	37
	Writing a Device Driver	38
	Include the Appropriate APIs (DKI/DDI)	39
	Register the Driver (using the <code>main()</code> function)	39
	Write the Device Driver Class-Specific Functions	39
	Write the Device Driver Registry Functions	39
	Write the Bus Event Handler Function	41
4.	Using the ChorusOS 4.0.1 Simulator	43
	Using the ChorusOS 4.0.1 Simulator	43
	Using XRAY with the ChorusOS Simulator	43
	System Debugging	44
	Application Debugging	44
	Index	45

Tables

TABLE P-1	Typographical Conventions	12
TABLE P-2	Shell Prompts	13
TABLE 3-1	UNIX Signal Classification	33

Figures

Figure 1-1	Loading Mechanism	16
Figure 1-2	Remote IPC Mechanism	17
Figure 1-3	Software Architecture	18
Figure 1-4	Network Data Flow	18
Figure 2-1	IP Configuration With Two Hosts	30

Code Examples

CODE EXAMPLE 3-1	Retrieving UNIX Function Addresses	37
CODE EXAMPLE 3-2	Retrieving Structure Pointers from Device Tree	40

Preface

The ChorusOS™ 4.0.1 Simulator for the Solaris™ Operating Environment (SPARC™ Platform Edition) is a port of the ChorusOS 4.0.1 operating system to the Solaris 2.6 and Solaris 7 (formerly 2.7) operating environments.

The ChorusOS 4.0.1 Simulator is a UNIX process that uses the Solaris API as if it were a hardware platform.

This book describes specific aspects of this port that relate to building, installing, using and programming the simulator. It is intended as a guide for engineers who wish to develop and test ChorusOS applications without the need for a physical target.

Prerequisites

This book assumes you are familiar with the ChorusOS 4.0.1 APIs. For information about the ChorusOS 4.0.1 operating system, please refer to the books within the *ChorusOS 4.0.1 Common Documentation Collection*.

How This Guide is Organized

- Chapter 1 introduces the ChorusOS 4.0.1 Simulator for the Solaris Operating Environment (SPARC Platform Edition) and describes the boot mechanism, remote inter-process communication, and the network architecture.

- Chapter 2 describes how to configure and run one or more instances of the simulator in a networked environment.
- Chapter 3 describes how to write a device driver for the simulator.
- Chapter 4 describes the differences between using the ChorusOS 4.0.1 Simulator and using the ChorusOS 4.0.1 operating system on a physical target.

Related Books

See the *ChorusOS 4.0 Installation Guide for Solaris Hosts* for a description of how to install the ChorusOS 4.0.1 product on your Solaris™ host workstation.

See the *ChorusOS 4.0 Simulator for the Solaris Operating Environment (SPARC Platform Edition) Target Family Guide* for a description of the product and installation instructions.

See the *ChorusOS 4.0 Introduction* for a complete description of ChorusOS 4.0.1 features.

Typographical Conventions

The following table describes the typographic conventions used in this book.

TABLE P-1 Typographical Conventions

Typeface or Symbol	Meaning	Example
AaBbCc123	The names of commands, files, and directories; on-screen computer output	Edit your <code>.login</code> file. Use <code>ls -a</code> to list all files. <code>machine_name%</code> you have mail.
AaBbCc123	What you type, contrasted with on-screen computer output	<code>machine_name%</code> su Password:

TABLE P-1 Typographical Conventions (continued)

Typeface or Symbol	Meaning	Example
<i>AaBbCc123</i>	Command-line placeholder: replace with a real name or value	To delete a file, type <code>rm filename</code> .
<i>AaBbCc123</i>	Book titles, new words or terms, or words to be emphasized.	Read Chapter 6 in <i>User's Guide</i> . These are called <i>class</i> options. You must be <i>root</i> to do this.

Shell Prompts

The following table shows the default system prompt and superuser prompt for the C shell, Bourne shell, and Korn shell.

TABLE P-2 Shell Prompts

Shell	Prompt
C shell prompt	<code>machine_name%</code>
C shell superuser prompt	<code>machine_name#</code>
Bourne shell and Korn shell prompt	<code>\$</code>
Bourne shell and Korn shell superuser prompt	<code>#</code>

Ordering Sun Documents

Fatbrain.com, an Internet professional bookstore, stocks select product documentation from Sun Microsystems, Inc.

For a list of documents and how to order them, visit the Sun Documentation Center on Fatbrain.com at <http://www1.fatbrain.com/documentation/sun>.

Accessing Sun Documentation Online

The docs.sun.comSM Web site enables you to access Sun technical documentation online. You can browse the docs.sun.com archive or search for a specific book title or subject. The URL is <http://docs.sun.com>.

Obtaining Technical Support

Sun Support Access offerings are available exclusively to members of the Sun Developer Connection Program. To get free membership in the Sun Developer Connection Program, go to <http://www.sun.com/developers>. For more information or to purchase Sun Support Access offerings, visit: <http://www.sun.com/developers/support> or contact the Sun Developer Connection Program office near you.

ChorusOS 4.0.1 Simulator Architecture

The ChorusOS 4.0.1 Simulator for the Solaris Operating Environment (SPARC Platform Edition) was designed to imitate the behavior of the ChorusOS operating system embedded on a physical target. In particular it:

- Supports remote Chorus IPC communication using Solaris UDP sockets.
- Supports Ethernet functionality allowing:
 - Solaris hosts to access the simulator with commands such as `rsh` and `ping`.
 - The simulator to access Solaris hosts' resources through standard Solaris daemons such as `nfsd` and `rarpd`.

The simulator is restricted to the flat memory model and supervisor actors.

Each simulator is a UNIX process running independently of other simulators, and up to 254 simulators can run and interact simultaneously on the same host or distributed over several hosts.

The simulator uses Solaris libraries in order to:

- Manage memory, interrupts, and process contexts.
- Access UDP socket functions for remote Chorus IPC data linking.
- Access stream functionality to allow communication between the ChorusOS Ethernet pseudo-driver and the Solaris Ethernet pseudo-driver.

Boot Mechanism

The simulator system image is booted with a utility called `loader` which:

1. Allocates 32 Megabytes of virtual memory for the simulator on the host.

2. Copies the system image to the allocated memory.
3. Configures the remote IPC-oriented UDP port and the IP address based on information in the site configuration file, `site_number.conf`.
4. Jumps to the entry point of the system image and runs the simulator.

Figure 1-1 illustrates the virtual address space of the UNIX process after loading the system image.

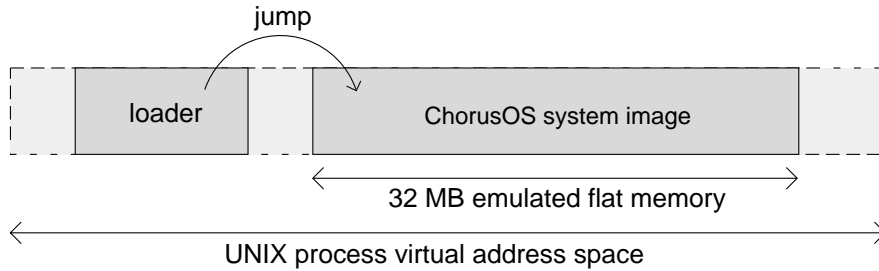


Figure 1-1 Loading Mechanism

Remote Inter-Process Communication

Remote IPC is based on Solaris UDP sockets which are used for sending, reading, and listening to IPC messages. Each simulator uses a single UDP socket to communicate with other simulators. The port number of this socket is specified in the `site_number.conf` file. Broadcasting is achieved by sending IPC messages to each simulator in this file.

Figure 1-2 illustrates the remote IPC mechanism.

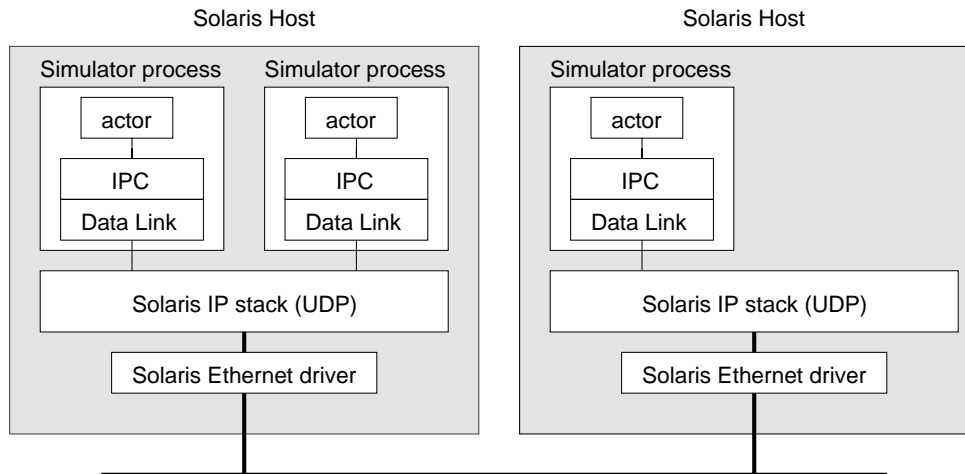


Figure 1-2 Remote IPC Mechanism

Network Architecture

Each simulator on a Solaris host accesses remote machines in the same way that a physical target accesses the outside world: through a network interface. As the network is not directly accessible through a dedicated Ethernet device, a Solaris Ethernet pseudo-driver simulates a specific sub-network, grouping all the simulators running on the same host. In particular, the Ethernet pseudo-driver allows each simulator to:

- Have its own IP address.
- Access remote files via NFS.
- Be accessed with the `rsh` command.

The pseudo-driver also supports any additional Ethernet-based functionality.

Each simulator interfaces with the Solaris Ethernet pseudo-driver sub-network. All simulator-specific network requests on this sub-network are routed by this driver. Each time a request concerns a host outside this sub-network, the request is forwarded to the host Solaris IP stack to be processed.

Figure 1-3 illustrates the software architecture simulating the network. The Ethernet pseudo-driver manages communication between simulators located on the same host and between simulators and the host.

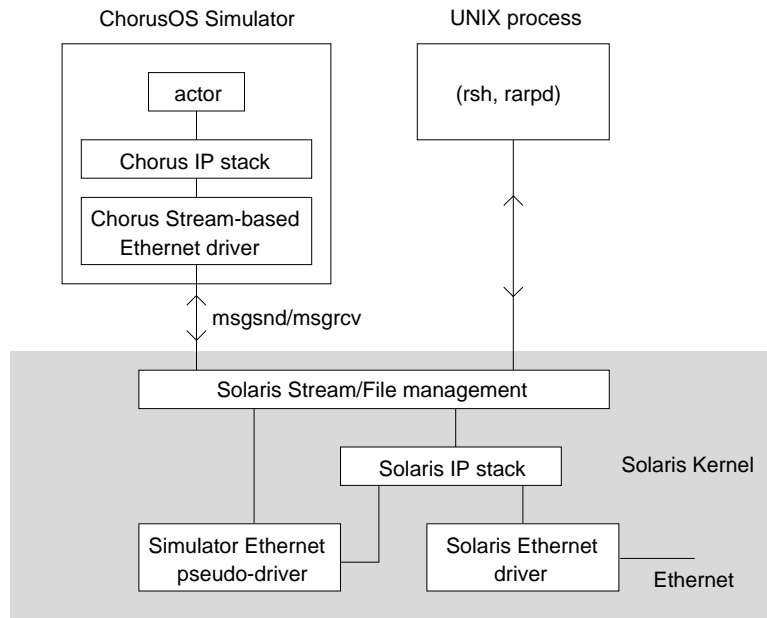


Figure 1-3 Software Architecture

This mechanism allows simulators to be accessed both from their host and also from any host on the network, after configuring the IP routing information.

Figure 1-4 illustrates the network data flow between simulators and the Ethernet network, with the Solaris IP stack relaying messages between networks.

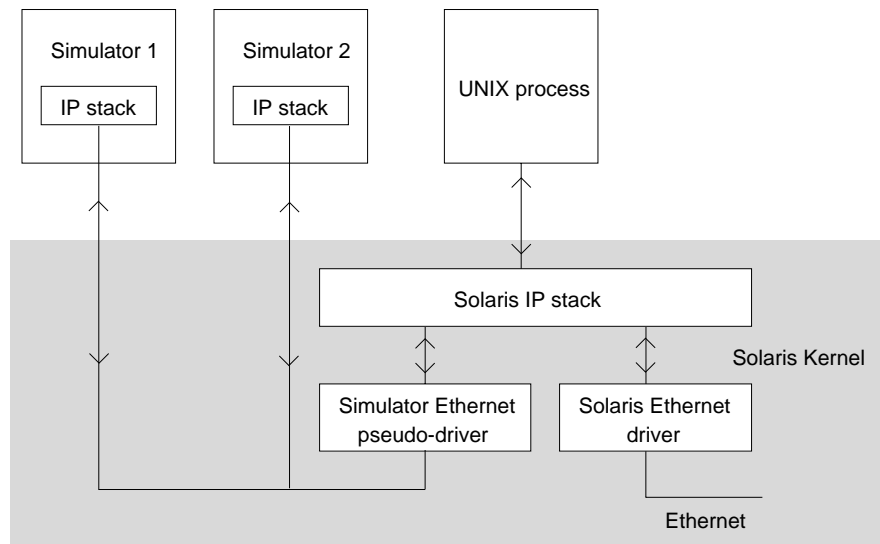


Figure 1-4 Network Data Flow

Running the ChorusOS 4.0.1 Simulator

This chapter describes how to configure and run the ChorusOS 4.0.1 simulator. It begins by describing what IP addresses must be chosen to access the simulator from your host, then presents a series of steps to get you quickly up and running, before going into further detail about each step.

If you have not created a simulator system image, please refer to the “Building the ChorusOS Simulator” section in the *ChorusOS 4.0.1 Simulator for the Solaris Operating Environment (SPARC Platform Edition) Target Family Guide* for details.

Planning Your Configuration

The following IP addresses must be selected to allow hosts to access one or more ChorusOS simulators:

1. A ChorusOS sub-network IP address for the ChorusOS sub-network which groups simulators running on the same host. Routing is achieved through comparisons with a netmask which must be selected according to IP routing rules. Contact your system administrator for help with selection if you are unsure.
2. A Solaris Ethernet pseudo-driver IP address. This address must belong to the ChorusOS sub-network and is used by simulators to access their supporting host.
3. A ChorusOS simulator instance IP address for each simulator in the ChorusOS sub-network that will run on your Solaris host. This address is used to access simulators from their supporting host and remote hosts. It is configured in the site configuration file, `site_number.conf`. See the `site_number.conf(4CC)` man page for more information.

An example configuration of a Solaris host with a IP address of 1.11.7.1 follows. Each simulator, in addition to the Solaris Ethernet pseudo-driver, is on a class A network, with a netmask of 0xff000000.

- ChorusOS sub-network IP address: 2.0.0.0.
- Solaris pseudo-driver IP address: 2.1.1.1.
- ChorusOS simulator instance IP address: in the range 2.1.1.2 to 2.254.254.254.

These settings will be used in the next section.

Note - If you run your simulator on the same machine as your host and do not communicate with other machines on your network then the same ChorusOS sub-network IP address can be used again by other hosts and simulators. It does not need to be declared to the rest of the network, a non-trivial task involving your system administrator.

See “Accessing the Simulator from Remote Hosts” on page 30 for more information.

Getting Started Quickly

Follow the procedure below to get quickly up and running with one or more simulators on a single host. References to later sections provide links to further information about each step.

1. **Create a file called `site_number.conf` in the `/usr/local/chorus/simu_admin` directory with the following lines:**

```
1  hostname      2052      2.1.1.2
2  hostname      2053      2.1.1.3
```

hostname is the name of your Solaris host.

See “Enabling Dynamic Configuration” on page 22 for more information.

2. **Create a file called `simudrv.conf` also in the `/usr/local/chorus/simu_admin` directory with the following line:**

```
2.1.1.1      hostname
```

hostname is the name of your Solaris host.

See “Configuring the Solaris Ethernet Pseudo-driver” on page 23 for more information.

3. **Within the `sysadm.ini` file in the `build_dir/conf` directory:**

- Find the comment `# Using ifconfig` then add the following line on the next line:

```
ifconfig ifeth0 TAG.TGT.IPA.TAG netmask 0xff000000 broadcast 2.255.255.255
```

See “Configuring the Simulator IP Address” on page 24 for more information.

- Add the following line on the line after:

```
route add default 2.1.1.1
```

See “Configuring the Gateway for the Simulator” on page 25 for more information.

- Find the line `rarp ifeth0` then comment out the `rarp` command by adding a hash to the beginning of the line:

```
#rarp ifeth0
```

4. If you wish to communicate with your simulator remotely from another machine, configure IP forwarding by entering the following command on your Solaris host with super-user privileges:

```
# ndd -set /dev/ip ip_forwarding 1
```

See “Configuring IP Forwarding for Your Host” on page 26 for more information.

5. Change to the Solaris Ethernet pseudo-driver sub-directory:

```
$ cd install_dir/tools/host/simu_drv
```

Launch the Solaris pseudo-driver by entering the following command on your Solaris host with super-user privileges:

```
# sh simudrv start
```

See “Running the Solaris Ethernet Pseudo-driver” on page 23 for more information.

6. Change to your build directory and build a simulator system image:

```
$ make chorus
```

7. Start the simulator using the `loader` command:

```
$ loader chorus.RAM 1
```

See “Booting the Simulator System Image” on page 26 for more information.
You can run a second simulator with the following command:

```
$ loader chorus.RAM 2
```

You can run additional simulators by adding entries to `site_number.conf` and using the `loader` command to start each simulator.

8. **Test that the simulator can be reached from the supporting Solaris host with the `ping` command:**

```
$ ping 2.1.1.2
```

You should receive the message `2.1.1.2 is alive`.

9. **Test that the supporting Solaris host can be reached from the simulator with the `ping` command:**

```
$ rsh 2.1.1.2 ping 2.1.1.1
```

If you receive the message `2.1.1.1 is alive`, you can start developing applications on the simulator as you would on a physical target.

Enabling Dynamic Configuration

The ChorusOS 4.0.1 Simulator for the Solaris Operating Environment (SPARC Platform Edition) architecture makes it possible to run multiple instances of the same system image. This requires a dynamic configuration at load time, which makes use of the UDP port for remote IPC communication and IP addresses for Ethernet communication.

To enable dynamic configuration, a site configuration file is required to identify each simulator. By default, this file must be named `site_number.conf` and placed in the `/usr/local/chorus/simu_admin` directory. You can override the default pathname by setting the `CHORUS_SITE_NUMBER_FILE` environment variable before launching the ChorusOS Simulator.

Each simulator must be assigned an instance number at boot time, which acts as an index to entries in the site configuration file, each of which defines:

- The hostname of the host system on which the simulator instance runs.

- The UDP port number used by the simulator instance for remote IPC.
- The IP address assigned to the simulator instance.

Here is the format of each entry:

```
instance_number  hostname  UDP_port  IP_address
```

Where:

- *instance_number* is a number between 1 and 254 used to reference an instance of the simulator.
- *hostname* is the name of your Solaris host.
- *UDP_port* is a UDP port number used for the remote IPC mechanism.
- *IP_address* is an IP address that will be used for simulator configuration.

Here is an example site configuration file:

```
#
# Site configuration file
#
1  server1      2052      2.1.1.2
2  server2      2053      2.1.1.3
```

A valid IP address is mandatory. This address is used to access the simulator from the supporting Solaris host and remote Solaris hosts.

Configuring the Solaris Ethernet Pseudo-driver

Create the file `simudrv.conf` in the `/usr/local/chorus/simu_admin` directory. This file must contain the IP address that will be attributed to the Ethernet pseudo-driver, and the name of the host on which your simulator is running. For example:

```
2.1.1.1      jericho
```

Running the Solaris Ethernet Pseudo-driver

Change to the Solaris Ethernet pseudo-driver sub-directory:

```
$ cd install_dir/tools/host/simu_drv
```

Launch the Solaris pseudo-driver with super-user privileges:

```
# sh simudrv start
```

Use the `ifconfig` command to ensure the driver is running correctly:

```
$ ifconfig -a
```

The output should include:

```
...  
  
simu0: flags=863<UP,BROADCAST,NOTRAILERS,RUNNING,MULTICAST> mtu 1500  
      inet 2.1.1.1 netmask 0xff000000 broadcast 2.255.255.255  
      ether 20:20:0:0:1:0  
  
...
```

This output will not be exactly as listed. It will vary depending on the configuration.

Configuring the Simulator IP Address

The simulator IP address is set with the `ifconfig` command which assigns an address to a network interface.

Add the following line to your `sysadm.ini` file in the *build_dir/conf* directory:

```
ifconfig ifeth0 TAG.TGT.IPA.TAG netmask netmask broadcast broadcast
```

netmask is the netmask corresponding to the ChorusOS sub-network, and *broadcast* is the broadcast address corresponding to the ChorusOS sub-network.

Comment out or remove the `rarp` command as the `ifconfig` command replaces it.

Here is an example of the `ifconfig` command:

```
ifconfig ifeth0 TAG.TGT.IPA.TAG netmask 0xff000000 broadcast 2.255.255.255
```

When you boot your system image with the `loader` command, the `TAG.TGT.IPA.TAG` IP tag is replaced by the IP address you registered in the site configuration file.

Every reference to the IP tag in the `sysadm.ini` file is replaced. This allows you to build one ChorusOS system image for every simulator running on the same host instead of having a different system image for each IP address.

You can also specify the IP address directly instead of using the IP tag, but this option offers less flexibility.

Note - Remember to rebuild your system image after modifying `sysadm.ini`.

Configuring the Gateway for the Simulator

A gateway is a device that enables data to flow between different networks. The gateway is configured using the following command on your simulator:

```
$ rsh simu_address route add default host_address
```

simu_address is the IP address of your simulator instance. *host_address* is the IP address of your Solaris host in the ChorusOS simulator sub-network, which is also the IP address of the Ethernet pseudo-driver.

Here is an example:

```
$ rsh 2.1.1.2 route add default 2.1.1.1
```

The `route` command can also be included in your `sysadm.ini` file, as well as executed on the local console.

Note - Simulators can only access their supporting host through the IP address of the Ethernet pseudo-driver. They cannot access their supporting host through its standard IP address.

The `mount` command must be run as follows:

```
$ rsh simu_address mount host_address:/pathname /
```

Here is an example:

```
$ rsh 2.1.1.2 mount 2.1.1.1:/build_dir/root /
```

See the *ChorusOS 4.0 File System Administration Guide* for more information about mounting file systems through NFS.

Configuring IP Forwarding for Your Host

A host supporting one or more ChorusOS simulators must be configured to allow IP packets to be forwarded to simulators on the ChorusOS pseudo-network. The forwarding option allows your host to forward any IP packet coming from its Ethernet controller to the Ethernet pseudo-driver. This is required if you want remote machines to access ChorusOS simulators, as well as allowing ChorusOS simulators to access remote machines.

To activate the forwarding option at boot time, create an empty file called `/etc/gateway`. After you reboot, forwarding will be activated automatically.

If you do not wish to either activate the forwarding option at boot time, or reboot your host, enter the following command with super-user privileges:

```
# ndd -set /dev/ip ip_forwarding 1
```

Booting the Simulator System Image

When the configuration file has been created, boot the simulator with the `loader` command.

1. **Make sure your PATH has been set correctly to include the directory `install_dir/tools/host/bin`.**
2. **Start the simulator by loading it into memory with the `loader` command:**

```
$ loader chorus.RAM instance_number
```

Where *instance_number* is the number of the simulator you have configured in the site configuration file.

The following messages are displayed on the standard output:

```
Loading chorus.RAM at 0x40000000 (1073741824)
IP address : 1 tag occurrence(s) patched
ChorusOS is booting ...
Reading site configuration file ...
```

```
ChorusOS r4.0.1 for Unix - Solaris/Sparc
Copyright (c) 1999 Sun Microsystems, Inc. All rights reserved.
```

```

Kernel modules : CORE SCHED_FIFO SEM MIPC IPC_L MEM_FLM KDB FAULT_KDB TICK MON ENV
ETIMER LOG LAPSAFE VTIMER MUTEX EVENT UI DATE PERF TIMEOUT LAPBIND DKI
MEM: memory device 'sys_bank' vaddr 0x40180000 size 0x148000
/UnixClock: clock -- resolution = 1000 nano-seconds
/UnixClock: simulator:solaris-timer driver started

/sigio/pseudoEther: sun:pseudo-ether driver started
/sigio: simulator:solaris-sigio driver started

/unixrtc: unixrtc-(rtc) driver started
TICK: warning -- timer device assigned to the system-tick not found
TICK: timer[0] (UnixClock) device is used for the system-tick
IOM: SOFTINTR DISABLED (-31). Using an Interrupt thread
IOM Init cluster space from: 0x400ad000 to: 0x400ed800 [129 items of size: 2048]
IOM Init io-buf pool from: 0x400ed890 to: 0x400eddb0 [8 items of size: 164]
IOM Init raw io-buffer pool from: 0x400eddb0 to: 0x400ef230 [32 items of size: 164]
Copyright (c) 1992-1998 FreeBSD Inc.
Copyright (c) 1982, 1986, 1989, 1991, 1993
    The Regents of the University of California. All rights reserved.

max disk buffer space = 0x10000
/rd: sun:ram--disk driver started
C_INIT: started
C_INIT: /image/sys_bank mounted on /dev/bd01
C_INIT: found /image/sys_bank/sysadm.ini
C_INIT: executing start-up file /image/sys_bank/sysadm.ini
iomMakeDevHandler name = ifeth
bpf: ifeth0 attached
IOM: ifnet ifeth0 bound to device /sigio/pseudoEther
iomMakeDevHandler name = lo
bpf: lo0 attached
iomMakeDevHandler name = bpf
iomMakeDevHandler name = bpf
add net default: gateway 2.1.1.1
ifeth0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> mtu 1500
        inet 2.1.1.2 netmask 0xff000000 broadcast 2.255.255.255
        ether 20:20:00:00:00:0c
lo0: flags=8049<UP,LOOPBACK,RUNNING,MULTICAST> mtu 16384
        inet 127.0.0.1 netmask 0xff000000
C_INIT: rshd started

```

Note - If you have more than one simulator running under the same instance number, the following error occurs:

```
/sigio/pseudoEther: error -- /dev/simul ioctl(ETHER_SET) failed (22)
```

The solution is to stop every instance of the simulator and run it again.

Accessing the Simulator From Your Host

1. Test that the simulator can be reached:

```
$ ping simu_address
simu_address is alive
```

Where *simu_address* is the IP address you configured for this simulator.
A message indicating that the simulator is working is displayed.

2. Use the `share` command to check that your build directory is available for mounting by remote hosts:

```
$ share
-          /build_dir  rw  ""
```

If your build directory is not shared, see “How to Mount and Unmount File Systems” in *ChorusOS 4.0 File System Administration Guide* for information on how to do this.

3. Create a root file system for your simulator in the build directory where you build system images:

```
$ make root
...
```

4. Attach the root file system of the simulator instance to your build directory:

```
$ rsh 2.1.1.2 mount 2.1.1.1:/build_dir/root /  
2.1.1.1:/build_dir/root on / (nfs)
```

The following error will occur if the build directory cannot be accessed due to the lack of access permission:

```
can't access /build_dir/root: Permission denied
```

To investigate this problem, check that the file system is mounted for access by everyone:

```
# share  
- /export/home rw=local "Local Disk"
```

Here, the file system is only available to members of the netgroup called local. To make the directory available from any machine, type the following command:

```
# share /export/home  
# share  
- /export/home rw ""
```

5. Issue the `ls` command to check that the simulator can be accessed from your machine:

```
$ rsh 2.1.1.2 arun ls /  
started aid = 2  
Makefile      dev          image      tmp  
bin           etc          lib
```

Accessing the Simulator from Remote Hosts

If forwarding is configured, any ChorusOS simulator located on your host can be reached remotely from any another machine supporting an IP stack. As ChorusOS simulators are only accessible through their supporting host, the IP routing protocol of the remote host must be configured to use the supporting host as a gateway for access.

Figure 2-1 shows the IP addresses of a configuration with two hosts, A and B, each running three simulators.

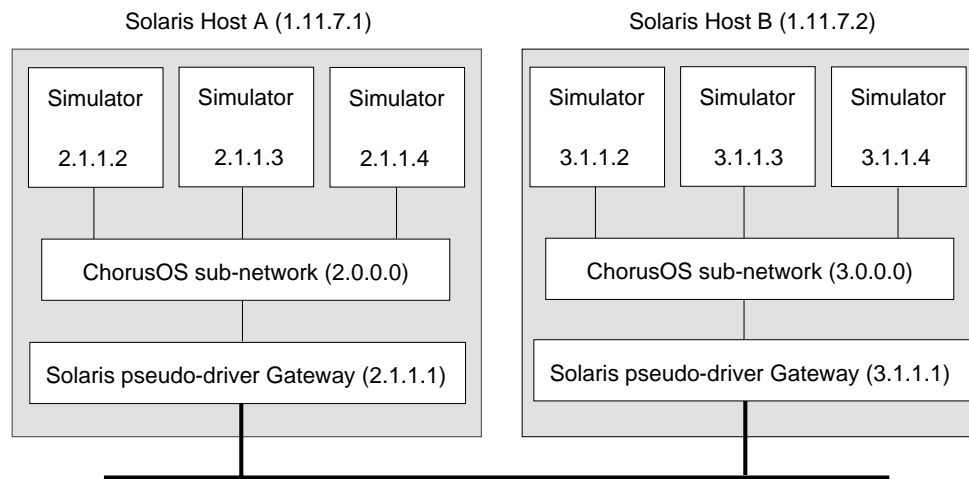


Figure 2-1 IP Configuration With Two Hosts

On each host, routing is configured using the following command with super-user privileges:

```
# route add -net sub-network hostname
```

sub-network is the ChorusOS IP address of the sub-network, and *hostname* is the name or IP address of your host running the ChorusOS simulator.

For example, to access simulators on Host A from any other host, type the following command on the other host with super-user privileges:

```
# route add -net 2 1.11.7.1
```

Note - Since the netmask has the value 0xff000000, the last three numbers of the IP sub-network address are ignored and the *sub-network* parameter can be shortened to 2.

Similarly, to access simulators on Host B from any other host, type the following command on the other host with super-user privileges:

```
# route add -net 3 1.11.7.2
```

Please consult your *Solaris System Administration Collection* for your specific configuration.

BSP for the ChorusOS 4.0.1 Simulator

This chapter describes the simulator BSP and includes the following sections:

- “UNIX Signals” on page 33 describes how signals are used by the ChorusOS Simulator and how they are classified.
- “Use of UNIX Services” on page 35 describes how to access system functions by making indirect system calls.
- “Writing a Device Driver” on page 38 describes how to write a device driver.

UNIX Signals

One of the ways supervisor actors communicate with the Solaris Operating Environment, through ChorusOS APIs, is by using signals.

The ChorusOS 4.0.1 Simulator for the Solaris Operating Environment (SPARC Platform Edition) converts UNIX signals into interrupts and exceptions. As a result, processes outside the ChorusOS operating system can signal to the ChorusOS process in order to generate asynchronous events. These events are then caught by supervisor processes through the standard ChorusOS interrupt management interface. Supervisor processes can also use the exception management interface to handle exception-type signals, just as they would for native exceptions on native hardware.

Table 3–1 illustrates how UNIX signals are classified.

TABLE 3-1 UNIX Signal Classification

Signal Number	Category	Interrupt Number	Exception Number
SIGHUP	UNIX	—	—
SIGINT	INTERRUPT	K_INTR_0	—
SIGQUIT	INTERRUPT	K_INTR_1	—
SIGILL	EXCEPTION	—	K_EXCP_ILL
SIGTRAP	EXCEPTION	—	K_EXCP_TRAP
SIGABRT	INTERRUPT	K_INTR_2	—
SIGEMT	EXCEPTION	—	K_EXCP_EMT
SIGFPE	EXCEPTION	—	K_EXCP_FPE
SIGKILL	UNIX	—	—
SIGBUS	EXCEPTION	—	K_EXCP_BUS
SIGSEGV	EXCEPTION	—	K_EXCP_SEGV
SIGSYS	EXCEPTION	—	K_EXCP_SYS
SIGPIPE	INTERRUPT	K_INTR_3	—
SIGALRM	INTERRUPT	K_INTR_TIMER	—
SIGTERM	INTERRUPT	K_INTR_5	—
SIGUSR1	INTERRUPT	K_INTR_6	—
SIGUSR2	INTERRUPT	K_INTR_7	—
SIGCLD	IGNORED	—	—
SIGPWR	INTERRUPT	K_INTR_8	—
SIGWINCH	INTERRUPT	K_INTR_9	—
SIGURG	INTERRUPT	K_INTR_10	—
SIGIO	INTERRUPT	K_INTR_IORDY	—
SIGSTOP	UNIX	—	—
SIGTSTP	UNIX	—	—
SIGCONT	IGNORED	—	—
SIGTTIN	UNIX	—	—
SIGTTOU	UNIX	—	—
SIGVTALRM	INTERRUPT	K_INTR_12	—
SIGPROF	INTERRUPT	K_INTR_13	—

TABLE 3-1 UNIX Signal Classification *(continued)*

Signal Number	Category	Interrupt Number	Exception Number
SIGXCPU	EXCEPTION	—	K_EXCP_XCPU
SIGXFSZ	EXCEPTION	—	K_EXCP_XFSZ

Signals which are not described in this table (those up to SIGRTMAX) are ignored.

Use of UNIX Services

Although it is possible for actors to make UNIX system calls directly, you should avoid doing this because some calls change the state of processes, leading to unpredictable behavior from the ChorusOS operating system. Examples of these system calls are:

- A blocking read blocks an entire process.
- Certain library calls change the signal mask.

In addition, the ChorusOS embedded library may conflict with the Standard C library because there are function names in both libraries which are identical.

UNIX system calls can be called indirectly, without linking with their corresponding libraries, by using a special structure whose fields are pointers to function entry points. However, this feature limits what functions can be accessed to the following:

- `accept()`
- `bind()`
- `calloc()`
- `close()`
- `connect()`
- `dlclose()`
- `dlopen()`
- `dlsym()`
- `errno()` (*errno* is returned by the `geterrno()` function)
- `fclose()`
- `fcntl()` (with some restrictions regarding arguments). Here is an extract from `exec/chLoader.h`:

```
int (*fcntl)(int, int);
```

```
int (*fcntl1)(int, int, int);
```

- fgetc()
- fopen()
- free()
- getcontext()
- gethostbyname()
- gethostent()
- getitimer()
- getpeername()
- getpid()
- getsockname()
- getsockopt()
- gmtime()
- gettimeofday()
- htonl()
- htons()
- inet_addr()
- ioctl() (with some restrictions regarding arguments). Here is an extract from `exec/chLoader.h`:

```
int (*ioctl1)(int, int, void *);
```

```
int (*ioctlInt)(int, int, int);
```

- listen()
- malloc()
- open()
- perror()
- poll()
- read()
- realloc()
- recv()

- `recvfrom()`
- `rewind()`
- `send()`
- `sendto()`
- `setcontext()`
- `setitimer()`
- `setsockopt()`
- `shutdown()`
- `sigaction()`
- `sigaddset()`
- `sigdelset()`
- `sigemptyset()`
- `sigfillset()`
- `sigismember()`
- `sigprocmask()`
- `socket()`
- `socketpair()`
- `sscanf()` (with some restrictions regarding arguments). Here is an extract from `exec/chLoader.h`:


```
int (*sscanf1)(const char*, const char*, void *);

int (*sscanf4)(const char*, const char*,
               void *, void *, void *, void *);
```
- `write()`

Retrieving UNIX Function Addresses

Code Example 3–1 illustrates how to retrieve a structure containing addresses of UNIX function calls which can later be used to make indirect system calls. This structure is stored as a property called `PROP_CPU_FUNCTION_ARRAY` which is attached to the CPU node of the device tree.

CODE EXAMPLE 3–1 Retrieving UNIX Function Addresses

```
/* include files */
#include <cpu/unixProp.h>
#include <exec/chLoader.h>
```

```

        .
        .

/* declaration of pointer for address handling */
static UnixFct* unixFct;

        .
        .

DevProperty prop;
void** tmpPtr;

/* looking for the CPU node in the device tree */
cpuNode = dtreeNodeFind(dtreeNodeRoot(), NODE_CPU);
if (cpuNode == NULL) {
    DKI_ERR(("No NODE_CPU in the device tree\n"));
    return;
}

/* Retrieve PROP_CPU_FUNCTION_ARRAY property */
prop = dtreePropFind(cpuNode, PROP_CPU_FUNCTION_ARRAY);
if (prop == NULL) {
    DKI_ERR(("No PROP_CPU_FUNCTION_ARRAY property in the NODE_CPU node\n"));
    return;
}

/* reading of its value */
tmpPtr = dtreePropValue(prop);
if (tmpPtr == NULL) {
    DKI_ERR(("No PROP_CPU_FUNCTION_ARRAY value in the NODE_CPU node\n"));
    return;
}
unixFct = *tmpPtr;

```

Access your UNIX functions using the `->` structure operator:

`unixFct->function_name`

function_name is the name of the function you want to call.

Please refer to the `exec/chLoader.h` header file for a list of types and parameters. Note that some types and constants have been prefixed to avoid conflicts with ChorusOS types and constants.

Writing a Device Driver

The architecture of the ChorusOS 4.0.1 Simulator for the Solaris Operating Environment (SPARC Platform Edition) is similar to ChorusOS 4.0.1 running on a physical target, except that direct access to hardware is not possible. Hardware can only be accessed through UNIX APIs based on UNIX file descriptors.

A typical example of a device driver would be one to emulate serial communication by using the `read()` and `write()` UNIX functions.

The success or failure of I/O operations are not signalled by interrupts as they are on a conventional target, but by a single UNIX signal, `SIGIO`. Instead of attaching an interrupt, drivers attach a file descriptor to the interrupt handler.

Here is the procedure for writing a device driver using file descriptors for I/O access:

1. Retrieve the structure containing the UNIX function addresses.
2. Attach a file descriptor to the interrupt handler by calling the `intr_attach()` function.
3. Handle the interrupts by calling the appropriate UNIX functions.

Please refer to the *ChorusOS 4.0 Device Driver Framework Guide* for details on writing a device driver.

The following sections follow the same format as “Writing Bus Drivers” in *ChorusOS 4.0 Device Driver Framework Guide*. Only the differences are described.

Include the Appropriate APIs (DKI/DDI)

In addition to including the header files for the DKI and DDI APIs involved in your device driver’s implementation, you must also include the appropriate header file to access UNIX function calls. See “Retrieving UNIX Function Addresses” on page 37 for more details.

Register the Driver (using the `main()` function)

The simulator BSP is fully compliant with the standard BSP.

Write the Device Driver Class-Specific Functions

The simulator BSP is fully compliant with the standard BSP.

Write the Device Driver Registry Functions

There are two differences that you need to be aware of when writing registry functions for your device driver.

- When accessing the device using UNIX function calls, you must perform these additional tasks:

- In the `close()` function, the file descriptor must be closed by calling the `close()` UNIX function.
 - In the `read()` and `write()` functions, a device must be accessed by using the `read()` and `write()` UNIX functions.
- When attaching to the interrupt handler, which processes the `SIGIO` signal, you must perform these additional tasks:
- In the `init()` function, the UNIX function structure pointer must be retrieved from the device tree (see “Retrieving UNIX Function Addresses” on page 37).
- The specific part of the `init()` function of the UNIX function structure pointer is retrieved from the device tree as follows:

CODE EXAMPLE 3-2 Retrieving Structure Pointers from Device Tree

```
device->fd = unixFct->open("/dev/tty0", UNIX_O_RDWR, 0);

if (device->fd == -1) {
    DKI_ERR(("s: /dev/tty0 open failed (%d)\n", path, unixFct->geterrno()));
    return;
}

/* Open SIGIO Parent Class */
res = sigioOps->open((BusId)busId,
                    myNode,
                    eventHandler,
                    NULL,
                    device,
                    &device->busDevId);

if (res != K_OK) {
    DKI_ERR(("s: parent bus open() failed (%d)\n", path, res));
    return;
}

/* Connect Interrupt Handler based on file descriptor */
res = sigioOps->intr_attach(device->busDevId,
                           (void *)device->fd,
                           (BusIntrHandler)intrHandler,
                           device,
                           &device->busIntrOps,
                           &device->busIntrId);

if (res != K_OK) {
    DKI_ERR(("s: intr_attach failed (%d)\n", path, res));
    return;
}
```

In the `init()` function, a device must be opened by calling the `open()` UNIX function.

The interrupt number parameter is replaced by the file descriptor when calling the `intr_attach()` function.

Write the Bus Event Handler Function

The simulator BSP is fully compliant with the standard BSP. However, the shutdown related events, specified by the common bus API, are not implemented:

- SYS_SHUTDOWN
- DEV_SHUTDOWN
- DEV_REMOVAL

Using the ChorusOS 4.0.1 Simulator

This chapter describes the differences between using the ChorusOS 4.0.1 Simulator and using the ChorusOS 4.0.1 operating system on a physical target.

Using the ChorusOS 4.0.1 Simulator

Using the ChorusOS 4.0.1 is similar to using a ChorusOS host and physical target development environment, apart from two restrictions:

- Only supervisor actors are supported, not user or system actors. For definitions of user, system and supervisor actors, please refer to the “Glossary” section in the *ChorusOS 4.0 Introduction*.
- Only the flat memory model is supported, not protected or virtual memory models. For information on memory models, please refer to the “Operating System Components” section in the *ChorusOS 4.0 Introduction*.

Actors can be embedded in the simulator system image, run and debugged in exactly the same way as actors embedded in a ChorusOS system image.

Using XRAY with the ChorusOS Simulator

Using the XRAY debugging tool with the ChorusOS 4.0.1 Simulator for the Solaris Operating Environment (SPARC Platform Edition) is similar to using XRAY with a

physical target configuration. Please refer to the *XRAY Debugger for ChorusOS User's Guide*, included with the XRAY distribution, for detailed information.

System Debugging

System debugging is not currently possible with the RDBS debug server and the XRAY debugging tool. However, since the simulator is a UNIX process, you can use any Solaris debugging tool to debug the operating system.

Application Debugging

Application debugging involves using a debug server called RDBC to communicate with the XRAY debugging tool. See the “Application Debugging Architecture” section in the *ChorusOS 4.0 Introduction* for details of how to set up an application debugging session. There are no restrictions for application debugging.

Index

A

- accessing function calls 38
- actors
 - supervisor 33, 43
 - system 43
 - user 43
- APIs
 - DDI 39
 - DKI 39
- application debugging 44

B

- boot mechanism 15

C

- chLoader.h 35, 38
- close 40
- commands
 - loader 15, 24
- configuring
 - gateway for simulator 25
- configuring IP forwarding 26
- configuring pseudo-driver 23
- configuring simulator IP address 24

D

- DDI APIs 39

- DEBUG_SYSTEM 44

- debugging

 - application 44
 - system 44

- device drivers

 - bus event handler function 41
 - class-specific functions 39
 - registering 39
 - registry functions 39
 - writing 38

- DKI APIs 39

E

- events

 - asynchronous 33

- exceptions 33

F

- feature

 - DEBUG_SYSTEM 44

- file descriptors 39

- files

 - gateway 26
 - simudrv.conf 23
 - site_number.conf 19
 - sysadm.ini 24, 25

- forwarding

- activating 26
- function addresses
 - retrieving 37
- function calls
 - accessing 38
 - close 40
 - init 40
 - intr_attach 40
 - read 40
 - write 40

G

- gateway 26
 - configuring for simulator 25

H

- header files
 - chLoader.h 35, 38

I

- init 40
- interrupts 33
- intr_attach 40
- IP address
 - pseudo-driver 19
 - simulator instance 19
 - sub-network 19
- IP configuration
 - planning 19
- IP forwarding
 - configuring 26
- IP tag 24
- IPC
 - remote 16

L

- library
 - Standard C 35
- loader 15, 24

N

- netmask 20

P

- processes 33
- properties
 - PROP_CPU_FUNCTION_ARRAY 37
- pseudo-driver 17
 - installing and configuring 23
 - launching 24
 - Solaris Ethernet 23
- pseudo-driver IP address 19

R

- RDBC 44
- RDBS 44
- read 40
- remote IPC 16
- retrieving function addresses 37
- routing
 - configuring 30

S

- SIGIO 39, 40
- signals
 - SIGABRT 34
 - SIGALRM 34
 - SIGBUS 34
 - SIGCLD 34
 - SIGCONT 34
 - SIGEMT 34
 - SIGFPE 34
 - SIGHUP 34
 - SIGILL 34
 - SIGINT 34
 - SIGIO 34, 39, 40
 - SIGKILL 34
 - SIGPIPE 34
 - SIGPROF 34
 - SIGPWR 34
 - SIGQUIT 34
 - SIGSEGV 34
 - SIGSTOP 34
 - SIGSYS 34
 - SIGTERM 34
 - SIGTRAP 34
 - SIGTSTP 34
 - SIGTTIN 34

- SIGTTOU 34
- SIGURG 34
- SIGUSR1 34
- SIGUSR2 34
- SIGVTALRM 34
- SIGWINCH 34
- SIGXCPU 35
- SIGXFSZ 35
- UNIX 33
- simudrv.conf 23
- simulator
 - accessing from remote hosts 30
 - accessing from your host 28
 - architecture 17
 - boot mechanism 15
 - error messages 27
 - errors 29
 - network data flow 18
 - restrictions 43
 - software architecture 18
- simulator instance IP address 19
- simulator IP address
 - configuring 24
- site_number.conf 19
- Standard C library 35
- sub-network
 - ChorusOS 19
- sub-network IP address 19
- supervisor actors 33
- sysadm.ini 24, 25
- system calls
 - accept 35
 - bind 35
 - calloc 35
 - close 35
 - connect 35
 - direct use of 35
 - dlclose 35
 - dlopen 35
 - dlsym 35
 - errno 35
 - fclose 35
 - fcntl 35
 - fgets 36
 - fopen 36
 - free 36
 - getcontext 36
 - gethostbyname 36
 - gethostent 36
 - getitimer 36
 - getpeername 36
 - getpid 36
 - getsockname 36
 - getsockopt 36
 - gettimeofday 36
 - gmtime 36
 - htonl 36
 - htons 36
 - inet_addr 36
 - ioctl 36
 - listen 36
 - malloc 36
 - open 36
 - perror 36
 - poll 36
 - read 36
 - realloc 36
 - recommended 35
 - recv 36
 - recvfrom 37
 - rewind 37
 - send 37
 - sendto 37
 - setcontext 37
 - setitimer 37
 - setsockopt 37
 - shutdown 37
 - sigaction 37
 - sigaddset 37
 - sigdelset 37
 - sigemptyset 37
 - sigfillset 37
 - sigismember 37
 - sigprocmask 37
 - socket 37
 - socketpair 37
 - sscanf 37
 - write 37
- system debugging 44

U

UNIX

- signals 33

W

write 40

using 43

X

XRAY