LynxOS-178 User's Guide

LynxOS-178

DOC-2204-00

Table of Contents

Preface

Typographical Conventions

The typefaces used in this manual, summarized below, emphasize important concepts. All references to filenames and commands are case-sensitive and should be typed accurately.

Kind of Text	Examples
Body text; *italicized* for emphasis, new terms, and book titles	Refer to the *LynxOS-178 User's Guide*
Environment variables, filenames, functions, methods, options, parameter names, path names, commands, and computer data	`ls -l myprog.c /dev/null`
Commands that need to be highlighted within body text or commands that must be typed as is by the user are **bolded**.	`login:` **`myname`** `#` **`cd /usr/home`**
Text that represents a variable, such as a filename or a value that must be entered by the user, is *italicized*.	`cat` *`<filename>`* `mv` *`<file1>`* *`<file2>`*

Blocks of text that appear on the display screen after entering instructions or command	`Loading file /tftpboot/shell.kdi into 0x4000`
	`.....................................`
	`File loaded. Size is 1314816`
	`© 2015 Lynx Software Technologies, Inc. All rights reserved.`

Keyboard options, button names, and menu sequences	**`Enter, Ctrl-C`**

Technical Support

Lynx Software Technologies handles support requests from current support subscribers. For questions regarding Lynx Software Technologies products, evaluation CDs, or to become a support subscriber; our knowledgeable sales staff will be pleased to help you. Please visit us at:
`http://www.lynx.com/training-support/contact-support/`

How to Submit a Support Request

When you are ready to submit a support request, please include *all* of the following information:

- First name, last name, your job title
- Phone number, e-mail address
- Company name, address
- Product version number
- Target platform (for example, PowerPC)
- Board Support Package (BSP), Current Service Pack Revision, Development Host OS version
- Detailed description of the problem that you are experiencing:
- Is there a requirement for a US Citizen or Green Card holder to work on this issue?
- Priority of the problem - Critical, High, Medium, or Low?

Where to Submit a Support Request

Support, Europe	tech_europe@lynx.com +33 1 30 85 93 96
Support, worldwide except Europe	support@lynx.com +1 800-327-5969 or +1 408-979-3940 +81 33 449 3131 [for Japan]
Training and Courses	USA: training-usa@lynx.com Europe: training-europe@lynx.com USA: +1 408-979-4353 Europe: +33 1 30 85 06 00

CHAPTER 1 Introduction

What is LynxOS-178?

LynxOS-178 is Lynx Software Technologies Inc.'s Real-Time Operating System (RTOS) for safety-critical systems. Lynx Software Technologies, Inc. is the premier developer of POSIX conformant real-time operating systems. Our flagship product, called LynxOS, is in use in hundreds of thousands of installations where high reliability and hard real-time determinism are essential. LynxOS-178 is based on LynxOS and has the features necessary for safety-critical applications such as aviation, defense, medicine, along with other business-critical fields. Along with the operating system and the development tools, Lynx Software Technologies can optionally provide the necessary artifacts to permit LynxOS-178 to be used in systems that are certifiable up to level A of the RTCA DO-178C standard. In addition, LynxOS-178 provides the ability to run multiple levels of DO-178C criticality on the same platform.

LynxOS-178 Development Environment

The LynxOS-178 product uses the following environment:

- Host platform: As described in the release notes.
- Target system: Usually purpose-built computers running a custom-configured Board Support Package (BSP) for that board. When the actual target systems are not yet available, development can be done using "reference platforms" (that is, commercially available computers for which a BSP already exists). Contact a Lynx Software Technologies representative for information about target platforms that are currently available.

Features of LynxOS-178

LynxOS-178 provides the following features:

- UNIX-LIKE ENVIRONMENT — The LynxOS-178 operating system is similar to UNIX. Applications use processes and threads, make system calls, and use device drivers. The product can run a shell on a serial port for a developer to interact directly with the target machine. It also has device drivers to permit mounting an external disk drive to facilitate testing and data capture.
- POSIX-CONFORMANT INTERFACES — LynxOS-178 offers POSIX.1 conformance including real-time and thread extensions which have extensive applicability for real-time and embedded systems.

  The real-time extensions include priority scheduling, real-time signals, clocks and timers, semaphores, message passing, shared memory, memory locking, synchronous and asynchronous I/O. The threads extensions include specifications for thread creation, control, and cleanup, thread scheduling, thread synchronization, and signal handling.
- DO-178C LEVEL A CERTIFIABLE ARTIFACTS — LynxOS-178 provides certifiable software and artifacts that allow developers to speed safety-critical systems to market. The release includes a complete artifacts package for the kernel and system services, including full DO-178C traceability through requirements, design, code, test, and test results. These artifacts have been approved by FAA under the Advisory Circular AC 20-148, *Reusable Software Components* to *DO-178 Level A*. System Integrators can use these approved artifacts without need for any additional approval.
- MATURE, STABLE, AND FULLY CERTIFIABLE — LynxOS, on which LynxOS-178 is based, is an embedded real-time operating system that has been rigorously exercised through millions of deployments. LynxOS-178 is the foundation of multiple safety-critical systems that have been certifiable to DO-178C.
- HARD PARTITIONING OF TIME, MEMORY, AND RESOURCES — LynxOS-178 implements a time-slice scheduling algorithm that gives each partition fixed execution time so that the system can be deterministically safe. Additionally, the system allows multiple applications of differing criticality levels within partitions to execute, completely isolated, on the same hardware resource. With LynxOS-178,

each task runs protected in its own space for uncompromising reliability within an ARINC 653 partition, enabling easier application certification.

- UPGRADABLE WITHOUT INVALIDATING CERTIFICATION — Mountable file systems and dynamically installed device drivers ease the certification of upgrades and enhancements. Applications and drivers are not required to be linked to the operating system and can, therefore, be isolated, limiting recertification efforts for the full operating system when only an application or driver needs modification.
- ARINC 653-1 SUPPORT — LynxOS-178 conforms to the ARINC 653-1 APEX Interface defined by the ARINC 653-1 standard, ensuring application portability, software reuse, and interoperability between embedded systems. LynxOS-178 provides the following system service groups in accordance with the ARINC 653-1 standard: Partition Management, Process Management, Time Management, Inter-partition Communication (Sampling and Queuing Ports), Intra-partition Communication (Buffers, Blackboards, Semaphores, and Events), and Health Monitoring.
- ARINC 653-2 SUPPORT — LynxOS-178 conforms to the ARINC 653-2 APEX Interface defined by the ARINC 653-2 standard, ensuring application portability, software reuse, and interoperability between embedded systems. LynxOS-178 provides the following system service groups in accordance with the ARINC 653-2 standard: File System, Sampling Port Extensions, and Memory Blocks.
- NETWORKING SUPPORT IN THE DEVELOPMENT ENVIRONMENT — The Development Environment of LynxOS-178 includes a default TCP/IP stack to enable development and debugging. The default TCP/IP stack includes support for the following utilities and services: `ping`, `rcp`, `rlogin`, `route`, `NFS client`, `ftpd`, `irshd`, `rlogind`, `rshd`, and `telnetd`.

NOTE: The default TCP/IP stack is intended for use during software development. It is not to be used in certifiable systems and is not partition-aware.

- USER-SUPPLIED NETWORK STACK SUPPORT IN THE PRODUCTION ENVIRONMENT — The user may supply his or her own driver that will implement the network stack as well as UNIX domain datagram-based socket functionality. The Production Environment Kernel has hooks for user-supplied functions.

- Production Mode LCS TCP/IP Stack – Lynx offers as an add-on product to LynxOS-178, the Lynx Certifiable Stack (LCS). LCS is a certifiable TCP/IP IPv4 network stack that is partition aware. The reader is referred to the *Lynx Certifiable Stack User's Guide* for more information.
- INTRA-PARTITION COMMUNICATION USING UNIX DOMAIN SOCKETS — In the Development Environment, both stream- and datagram-based sockets are supported. In the Production Environment, stream-based sockets are supported by default and support for the rest depends on a user-supplied network stack or LCS (which is Lynx's implementation of a user-supplied network stack).
- HEALTH MONITORING & SIGNAL CATCHING — The Health Monitor allows a system to catch, process, and isolate faults in applications and their containing partitions, and to prevent failures from propagating.
- THREAD SAFE BSD LIBRARY — The `libbsd` library that provides network-related thread-safe functions such as `gethostbyname`, `gethostbyaddr`, and `inet_ntoa`. The BSD-based library is not intended for production use, and is not partition-aware. For a production and partition-aware network library, users should a user-supplied network stack or the Lynx Certifiable Stack (LCS).
- CONFIGURABLE TICK TIMER FREQUENCY — The user may adjust Tick Timer Frequency when the kernel is created.

LynxOS-178 Differences

LynxOS-178 is a real-time operating system that uses a priority-preemptive scheduler to schedule tasks. Each process runs in its own virtual address space, so that processes cannot normally affect each other's memory. LynxOS-178 also provides an environment in which various applications are partitioned from each other. This partitioning is done with sufficient rigor that it can be proven with formal methods that applications in different partitions cannot interfere with each other.

In LynxOS-178, each partition runs as a distinct entity from the other partitions. A partition is sometimes referred to as a Virtual Machine (VM) because of the separation partitioning provides. LynxOS-178 does not, however, incorporate any hypervisor functionality within the OS itself.

Processes running in one partition are aware of only events in their own partition, just as a process running on distinct operating systems are aware only of events on that computer. The exception is partition 0 (P0), which has special privileges. These privileges are like the root privileges in UNIX systems. For example, P0 can override protections set in other partitions and can reboot the computer. In addition, P0 knows the state of the processes and threads of the other partitions.

Within a non-root partition (partitions other than P0), the processes cannot affect, or be affected by, processes in other non-root partitions. In fact, processes in a non-root partition have no way of knowing whether other partitions exist, except through managed interfaces that work like I/O.

The partitioning involves exclusive access of three kinds: time, memory, and resources. Time partitioning is done through a time-slice scheduler, which allocates periods of time to each partition. During each time slice, only processes in the assigned partition are permitted to execute. Figure 1-1 is a timeline that illustrates the temporal partitioning that LynxOS-178 does.

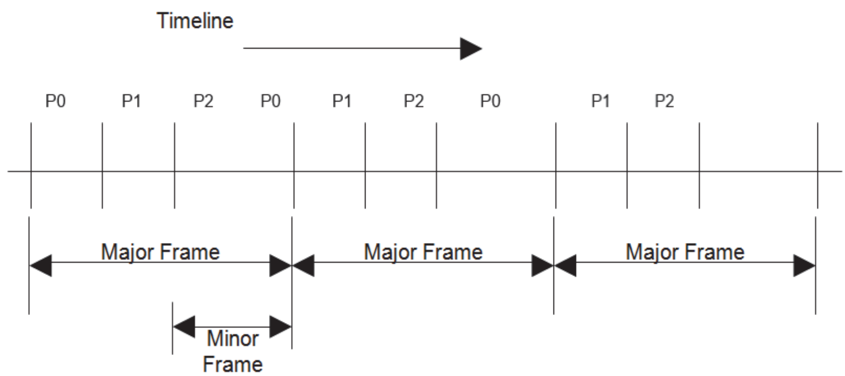

Figure 1-1: Time is Strictly Allocated to VMs

Memory partitioning is achieved by dividing RAM into discrete blocks of physical address space. Each partition is assigned one and only one block of memory.

Within the partition, the virtual address spaces of the various processes are mapped to memory from the assigned memory block.

Resource partitioning means that each device can be assigned to only one partition and that a fault in a device or its driver will be contained within a single partition. Each partition mounts a RAM-based file system for data storage.

Read/write file systems are private to the partitions and are never shared with other partitions.

Development Environment vs. Production Environment

LynxOS-178 contains two environments: The Production Environment and the Development Environment.

The Production version of LynxOS-178 has a feature set which supports certification to DO-178C Design Assurance Level A.

The Development Environment (a superset of the Production Environment) has additional features that assist in application development and debugging on LynxOS-178:

- MORE KERNEL FEATURES - `TTY`, `ptrace` and `skdb` for example
- SHELLS AND UTILITIES - `bash`, `tftp`, `ps`, `gzip`, and additional file system utilities such as `ls`, `cat`, `mkdir`, `cp`, and `rm`, for example
- DEBUGGERS - Standard `gdbserver` for working with GDB
- ADDITIONAL DEVICE DRIVERS - USB, SCSI and network interface drivers

These additional development mode features do not have supporting DO-178C artifacts and are intended for use only during development.

CHAPTER 2 Building and Booting LynxOS- 178 KDIs

A KDI is a Kernel Downloadable Image containing the LynxOS-178 Kernel and its associated root file system. The contents of the KDI are described in a spec file which is used by the mkimage utility to create the KDI. Refer to the mkimage and mkimage.spec man pages for detailed information.

LynxOS-178 comes with a sample spec file in the build tree:

```
sys/bsp.<bsp_name>/lynxos-178.spec.in
```

which is pre-processed to create the `lynxos-178.spec` file.

How to Create a KDI

To create a sample KDI, perform the following steps:

1. Source the SETUP`.bash` script in the root of the source tree:

   ```
   # cd /usr/los178/<release_num>/<cpu>_dev
   # . SETUP.bash
   ```

2. Change to the BSP directory:

   ```
   # cd sys/bsp.<bsp_name>
   ```

3. Rebuild the Lynx OS-178 Development Kernel and KDI:

   ```
   # make kdi
   ```

   This will create the file: `lynxos-178.kdi`.

The method of loading a KDI on a target board and the commands used to boot a KDI on a target board may vary from board to board. As such, they are documented in the *Board Support Guide* for each specific board.

How to Create a KDI using a demo template

The LynxOS-178 CDK/ODE CD-ROM contains templates that a user may use to build a KDI for execution on the target board.

The following table lists the KDI templates that are provided with LynxOS-178. Each template provides pre-defined specification files that can be used as a starting point to customize the KDI for a user application.

Table 2-1: KDI Template

`arinc653`	This KDI contains ARINC 653 applications demonstrating the configuration and use of ARINC 653 queuing ports.
`developer`	This KDI contains development and networking utilities that provide a minimal configuration for development.

ARINC653 KDI

The `arinc653` demo contains ARINC 653 applications demonstrating the configuration and use of ARINC 653 queuing ports for inter-VM communication. This demo creates two VMs running a producer/consumer application using ARINC 653 queuing ports to transmit data between the two processes running in different VMs.

Developer KDI

By default, the `developer` demo contains development and networking utilities that provide a minimal configuration for development. The `developer` demo can be used to work with the LynxOS-178 Development Tools, specifically:

- Luminosity
- SpyKer

The procedure described below assumes that the LynxOS-178 cross-development environment with the KDI build templates and the required tools are installed on the cross-development host. Please refer to the user documentation for Luminosity and SpyKer for the detailed installation procedure. It is also assumed that the LynxOS-178 cross-development environment is set up.

1. From the LynxOS-178 cross-development environment in development mode, execute the `PROJECT.sh` script by entering the following command:

```
$ cd <installation_dir>
$ . SETUP.bash
```

where *<installation_dir>* is the directory path where LynxOS-178 environment were extracted (e.g. `/usr/los178/2.2.5/`).

```
$ cd $ENV_PREFIX/usr/demo
$ . PROJECT.sh
```

The following screen output is displayed:

```
$ ./PROJECT.sh

===============================================================
| Project set up script                                       |
|                                                             |
| This script will create a customized directory for building |
| and experimenting with Kernel Downloadable Images (KDIs).   |
===============================================================

Note: 10MB of available disk space is needed for minimal project
      directory. 30MB is recommended.

Project directory location?                   [/tmp/newproj] /tmp/demo
BSP_FILE is zc702
Enter network name/IP address for target now? [n] y
What is the network name of the target board? [lynxdemo] zc702
What is IP address name of the target board?  [1.1.1.1] 192.168.1.50
What is the network device of the target board? [gem0]

+==============================================================
The following variables have been set for this project:

PROJECT_DIR          = /tmp/demo
BSP_NAME             = bsp.zc702
BSP_LOC              = /usr/los178/2.2.5/arm_dev/sys
BSP_DIR              = /usr/los178/2.2.5/arm_dev/sys/bsp.zc702
DEMO_PREFIX          = /usr/los178/2.2.5/arm_dev/usr/demo
ENV_PREFIX           = /usr/los178/2.2.5/arm_dev
LYNX_TARGET_NAME     = zc702
LYNX_TARGET_IP       = 192.168.1.50

+==============================================================

press enter to continue:

Step 1: Make Project Directory               /tmp/demo

Step 2: Copy /usr/los178/2.2.5/arm_dev/sys/bsp.zc702 to $PROJECT_DIR

    cp -r /usr/los178/2.2.5/arm_dev/sys/bsp.zc702 $PROJECT_DIR
    chmod u+w $PROJECT_DIR/bsp.zc702/*
    copy $PROJECT_DIR/bsp.zc702/config.tbl.orig
```

```
    copy $PROJECT_DIR/bsp.zc702/uparam.h.orig


Step 3: Create KDI.sh

    Copying Makefile to $PROJECT_DIR
    Copying OBJ_RULES to $PROJECT_DIR/common
    Creating $PROJECT_DIR/KDI.sh
    Appending [zc702] bsp_tests.cfg FILE to $PROJECT_DIR/KDI.sh

Step 4: Copy supporting files, scripts and demo dirs.

    developer arinc653

Step 5: Modify
templates

    Updating $PROJECT_DIR/KDI.sh: my_name=lynxdemo
    Making $PROJECT_DIR_PORT/rc.network not interactive:
    INTERACTIVE_RCNETWORK=false

Step 6: Creating tty link

    Linking /usr/los178/2.2.5/arm_dev/etc/ttys ttys

Setup is complete!

***************************************************************
**  To begin building KDI demos:

    $ cd /tmp/demo/developer
    $ make all

*******************************************************************
```

2. After `PROJECT.sh` is executed, a new demo directory (`/tmp/demo` is the default) is created on the host. Modify the configuration of the `developer` demo (the default location is `/tmp/demo/developer`) by editing the `developer.spec` file and `rc.local` file (if needed) to add or remove functionality in the KDI.

   The configuration of the `arinc653` demo can be modified (The default location is `/tmp/demo/arinc653`) by editing the `arinc-demo.spec` file and the `example.vct.sample` file.

   The amount of RAM can be adjusted by editing the `demo.config` file.

3. After updating the KDI configuration, build the KDI by running the following command from the demo directory (the default location is `/tmp/demo/`):

   bash# **make all**

By default, the `gem0` interface is used for bringing up the network functionality. If the target board is connected to the network via a different interface, then, after the LynxOS-178 KDI is booted on the target board, the user has to execute the following command at the bash prompt to configure the correct interface:

```
bash# ifconfig <interface-name> <target_IP_address>
```

To start the `arinc653` demo, enter the following command:

```
bash# cinit cnc
```

To stop the `arinc653` demo, hit **CTRL-C**.

How to Enable Boot Profiling

By enabling boot profiling, the `cinit` utility will display the total boot time from the initial kernel execution through the start of the `cinit` utility.

It will also display how much time was spent in various boot stages.

For example:

```
cinit:VM0:     15.790 msecs  -- Before and during rk_move()
cinit:VM0:    306.815 msecs  -- Before and during clear HW page tbls
cinit:VM0:      7.327 msecs  -- Before and during bsp_map_io()
cinit:VM0:      1.220 msecs  -- During map_kernel()
cinit:VM0:      0.607 msecs  -- Before and during csp_post_init()
cinit:VM0:     16.765 msecs  -- Before and during init_sysbrk()
cinit:VM0:     18.441 msecs  -- During initmem()
cinit:VM0:      0.682 msecs  -- Before and during fast_info_init()
cinit:VM0:      0.868 msecs  -- Allocate and clear st table
cinit:VM0:      0.618 msecs  -- Before and during fp initialize()
cinit:VM0:      0.841 msecs  -- Before and during init_drivers()
cinit:VM0:      0.213 msecs  -- Before and during attach_skdb()
cinit:VM0:      0.413 msecs  -- Before cinit [creatinit()]
cinit:VM0:    195.927 msecs  -- Launching cinit
cinit:VM0:    566.526 msecs  -- Total boot time
```

More information on what impacts each boot stage time can be found in the `$ENV_PREFIX/usr/include/boot_profile.h` header file.

In development mode, boot profiling is enabled by changing the value of the `BOOT_PROFILE` macro to 1 in `/sys/bsp.`*`<bsp_name>`*`/uparam.h`. Then rebuild the BSP and KDI.

```
#define BOOT_PROFILE    1 // Set to 1 to enable Boot Profiling.
```

In the Production Environment, it is a little more complicated as the /dev/mem device is required by cinit to get the boot time data. This device is not normally configured into the production mode version of the kernel. In addition to enabling boot profiling in the uparam.h file, the following must also be done before building the BSP and KDI.

```
# cd $ENV_PREFIX/sys/drivers/mem
# make install
# cd $ENV_PREFIX/sys/bsp.<bsp_name>
```

Enable the inclusion of mem.cfg in the $ENV_PREFIX/sys/bsp.*<bsp_name>/config.tbl* by inserting comment "#" to disable the "ifdef"

```
#ifdef("VMOS_DEV","
I:mem.cfg
#")
```

Rebuild the kernel

```
# make kdi
```

CHAPTER 3 LynxOS-178 Architecture

The software that runs on the target system is generally classified as either System Software (that is, the LynxOS-178 Operating System) or Application Software.

System Software includes parts of the operating system, such as device drivers, which execute in processor supervisor mode. It also includes System Applications, such as `cinit`, which executes with operating system “root” privileges.

Application Software is the functional software that executes within a partition on the target system. Application Software always executes with operating system “user” privileges and is verified to the DO-178 criticality level appropriate for the intended function.

LynxOS-178 Operating System

Figure 3-1 displays the LynxOS-178 software architecture. LynxOS-178 is a UNIX-style operating system designed to allow multiple real-time applications with different criticality levels to execute concurrently on the same processor. LynxOS-178 provides time, memory, and resource partitioning to applications by isolating and preventing running applications from affecting (or being affected by) any other application.

The LynxOS-178 operating system is designed to be independent of its underlying hardware platform. A unique Board Support Package (BSP) and CPU Support Package (CSP) provide the hardware-specific services to LynxOS-178. The application's only interaction with LynxOS-178 is through its documented Application Programming Interface (API).

LynxOS-178 also handles errors and exception conditions that applications do not or cannot trap.

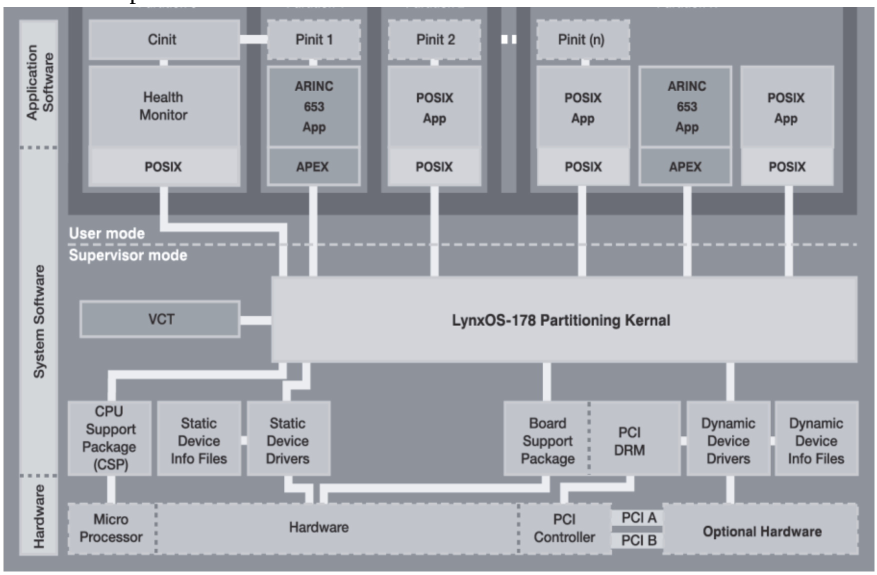

Figure 3-1: LynxOS-178 Architecture

NOTE: LynxOS-178 comprises all the components that appear in the region labeled System Software in this diagram.

The components labeled System Services and LOS-178 Kernel are the reusable software components.

The CSP, device drivers, BSP, and configuration tables may be different on different boards or microprocessors.

The Application Software is usually supplied by the system integrator.

CSP

The CSP contains all the processor family-specific routines, including the MMU, Floating Point, and processor exception handlers. The CSP routines are linked with the LynxOS-178 Kernel.

BSP and DRM

The BSP contains routines for initializing and controlling hardware on the target system. The primary responsibilities of the BSP are:

- Interface with Boot and Shutdown software
- Establish virtual address map for onboard I/O
- Interface with the interrupt controller
- Provide default handlers for error-signaling interrupts
- Interface with the PCI controller
- Interface with the system time (tick timer)

The PCI Device Resource Manager (DRM), shown with the BSP above, is platform-independent.

The primary responsibilities of the PCI DRM are:

- Locate the PCI devices
- Manage ownership of PCI devices
- Map devices into virtual address space
- Provide access to the PCI configuration space
- Manage interrupts of PCI devices

The BSP and the DRM are linked with the LynxOS-178 Kernel.

Tick Rate

The tick rate can be configured by adjusting the `TICKSPERSEC` macro in the `usr/include/conf.h` file. By default, the rate is set to 1000 ticks per second. After changing the value, the kernel must be rebuilt.

Static Device Drivers

The Static Device Drivers are software components that isolate specific details of hardware devices from Application Software components. Items such as hardware dependent interrupt handlers (for example, power warn and load shed) and kernel threads are added to the kernel with device drivers. Static device drivers are linked with the kernel.

Static Device Info Files

The Static Device Info Files are used to configure the Static Device Drivers for devices available in the target system. There are one or more info files per device driver. The static device info files are linked with the LynxOS-178 Kernel.

Dynamic Device Drivers

The Dynamic Device Drivers are hardware access routines for optional devices on the target system. These device drivers are stored in the file system and installed after the LynxOS-178 Kernel is booted.

Dynamic Device Info Files

The Dynamic Device Information Files are used to configure the Dynamic Device Drivers for optional devices on the target system. There can be one or more information files per device driver. These device info files are stored in the file system and installed after the LynxOS-178 Kernel is booted.

System Services

The system services are linked with the application code (C or C++) and run in processor user mode.

Application Programming Interfaces (API) include:

POSIX API	LynxOS-178 provides a subset of POSIX 1 operating system services
File System	LynxOS-178 provides a file system with a POSIX API and ARINC653 API.
IEEE Floating Point Services	LynxOS-178 provides services to configure floating point responses
ARINC653 API	LynxOS-178 provides a subset of ARINC653 operating system services

System Admin Services, available to P0 (that is, partition “zero”) only, include:

File System Admin services	`mount, ffsck, mkffs`
Scheduler service	`create_pinit()`

High Water Mark services	`_get_resource_entry()`
LynxOS-178 Library	`dr_install(), setgroups()`

Kernel

The LynxOS-178 Kernel is statically linked with the CSP, BSP, and Static Device Drivers to create the LynxOS-178 Operating System. During initialization, Dynamic Device Drivers are dynamically linked with LynxOS-178 and effectively become part of the operating system.

Common Initialization (cinit)

`cinit` is the first POSIX process to run after the LynxOS-178 Kernel has been initialized. `cinit` executes with the Operating System's root privileges. It reads the Virtual Machine Configuration Table (VCT) and creates the partitions within LynxOS-178.

The primary responsibilities of `cinit` is to perform the following:

- Validate and read the VCT (see Chapter 7, "Understanding the Virtual Machine Configuration Table")
- Load Dynamic Device Drivers
- Initialize system wide environment variables
- Mount the file systems
- Initialize the scheduler
- Start the user application
- Respond to partition fatal errors as defined within the VCT

Partition Initialization (pinit)

Just subsequent to the LynxOS-178 initialization process, the Operating System is able to then allocate partitions. At this point, `cinit` migrates into a unique `pinit` process within each partition. Each `pinit` acts in tandem with the system's Application Software for effective operation within the partition. `pinit` begins execution with the Operating System's root privileges.

Components Created by the User

In addition to the Application Software itself, the Systems Integrator creates files and binary images that are part of a fully functional software configuration.

Virtual Machine Configuration Table

The Virtual Machine Configuration Table contains configuration information to create partitions within LynxOS-178. The VCT also contains a Virtual Machine/Partition configuration profile for each partition. This information is used to allocate system resources to the application software, defining a valid configuration of the target system as determined by the user.

The VCT contains information based on the set of functionality loaded on the target system. For LynxOS-178, the VCT should be placed in both `/usr/local/etc` and `/usr/etc` directories.

Chapter 7, “Understanding the Virtual Machine Configuration Table” describes the syntax of the VCT.

KDI

The KDI is a single downloadable image containing LynxOS-178 and a root file system. The file system is a UNIX-style root file system. It should contain the minimum system software necessary for `cinit` to complete its initialization tasks. It also contains file system mount points for all other file systems used by any partition. A RAM disk is created on `/tmp` and mounted for each partition. All file systems specified in the VCT are mounted in directories as described in the <FS> section.

Table 3-1: Root File System Structure

Default LynxOS-178 Root File System Structure	
`/`	Root directory
`/bin`	Binary files
`/dev`	Statically installed device nodes
`/dev/ddev`	Dynamically installed device nodes
`/etc`	Configuration files
`/mnt`	File system mount point

Table 3-1: Root File System Structure

Default LynxOS-178 Root File System Structure	
`/tmp`	Temporary file system
`/usr`	
`/usr/etc`	A copy of the VCT is located here
`/usr/local`	
`/usr/local/etc`	A copy of the VCT is located here

CHAPTER 4 *Virtual Memory Map*

LynxOS-178 Virtual Memory Map

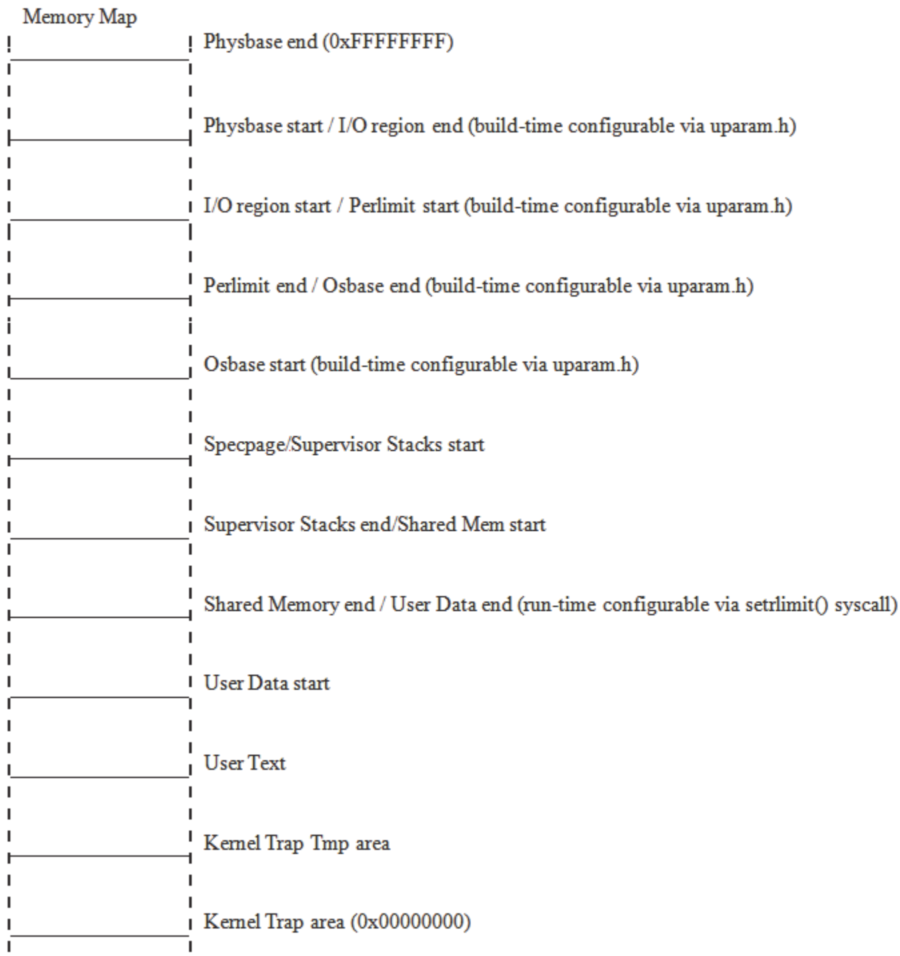

Figure 4-1: LynxOS-178 Virtual Memory Map

Configuring LynxOS-178 Virtual Memory Map

Configuring PHYSBASE Region Size

Size of the system RAM memory (PHYSBASE region) is configured by the KAS_PHYSBASE_SIZE macro in the `uparam.h` file. The following values are valid:

- 512 MB
- 1 GB
- 2 GB

By default, size of the system RAM memory is set to 2 GB.

Configuring I/O Region Size

Refer to the appropriate Board Support Guide for details of the I/O Region on a particular BSP.

Configuring Perlimit Size

Size of the Perlimit region is controlled by the `KAS_PERLIMIT_SIZE` macro in the `uparam.h` file. It shall be set to the 4 KB aligned non-zero value. By default, size of the Perlimit region is set to 512 MB or 256 MB.

Configuring OSBASE Size

Size of the OSBASE region is controlled by the `KAS_OSBASE_SIZE` macro in the `uparam.h` file. Note that the OSBASE region start address is automatically aligned to the BSP-specific boundary.

Refer to the appropriate *Board Support Guide* for details of the OSBASE alignment boundary on a particular BSP.

By default, size of the OSBASE region is set to 256 MB.

OSBASE Region Sections

The OSBASE region consists of the following sections:

Table 4-1: OSBASE Region Sections Information

Region Name	Region Size
Unused space	16Kb
OS Text	lmexe.sztext, page aligned
OS Data	lmexe.szdata +
OS BSS	lmexe.szbss +
OS Symbol Table	lmexe.szsymtab, page aligned
Root File System	lmexe.szrootfs, page aligned
Resident Text	lmexe.szsalt, page aligned
Kernel Environment Page	4Kb
Kernel Memory Heap	configurable

NOTE: Sizes of all sections except Kernel Memory Heap are pre-defined, size of the Kernel Memory Heap is configurable.

Calculating Size of OSBASE Region (Except Kernel Memory Heap)

To determine sizes of the OS Text, OS Data, OS BSS, OS Symbol Table, Root File System, Resident Text, and User File System regions, build KDI and UserFS:

```
$ cd $ENV_PREFIX/sys/bsp.<bsp_name>
$ make kdi
```

Check output produced by the `make kdi` command. Specifically, check the following values:

```
lmexe.sztext
lmexe.szdata
lmexe.szbss
lmexe.szsymtab
lmexe.szrootfs
lmexe.szsalt
```

For Example:

```
-----------------------------------------------------------
lmexe.sztext:   00078000
lmexe.szdata:   0000cbf8
lmexe.szbss:    0002846c
lmexe.szsymtab: 0000cd00
lmexe.szrootfs: 00032000
lmexe.szsalt: 00037000
-----------------------------------------------------------
```

For the example described above, size of the OSBASE region (except Kernel Memory Heap) will be equal to 0x00123000.

Configuring Size of Kernel Memory Heap

By default, `cinit` allocates unused kernel memory heap to all partitions equally. This default behavior is the preferred. However, if there is a real need to change this, the use can configure kernel memory heap by setting the `KernHeapMemLim` variable for each partition in the VCT file.

Specpage

The Specpage size is not configurable and is equal to 4 KB.

Supervisors Stacks Region

The Supervisors Stacks region size is not configurable and is calculated automatically.

Configuring User Stacks Region End Address

The User Stacks region is part of the User Shared memory. The end address of the User Stacks region is set at run-time via the `setrlimit()` system call.

Configuring User Data Region End Address

The end address of the User Data region is configured at run-time via the `setrlimit()` system call.

CHAPTER 5 Support for ARINC 653 in LynxOS-178

What is ARINC 653?

ARINC is an acronym for “Aeronautical Radio, Incorporated” which is an organization that issues specifications for airline electronic equipment.

ARINC 653 (Avionics Application Standard Software Interface) is a software specification for space and time partitioning in safety-critical avionics real-time operating systems. It allows the hosting of multiple applications of different software levels on the same hardware in the context of an Integrated Modular Avionics architecture.

In order to decouple the RTOS platform from the application software, ARINC 653 defines an API called APplication EXecutive (APEX). LynxOS-178 conforms to the ARINC 653 APEX.

ARINC 653 System Services in LynxOS-178

LynxOS-178 provides the following system service groups in accordance with the ARINC 653 standard:

- Partition Management
- Process Management
- Time Management
- Inter-partition Communication
  - Sampling Port Services
  - Queuing Port Services
- Intra-partition Communication
  - Buffer Services
  - Blackboard Services
  - Semaphore Services
  - Event Services
  - Memory Block Services
- Health Monitoring
- File System Services

Refer to the *ARINC 653 APEX Interface Conformance Document* for detailed descriptions of these services.

Installing ARINC 653 to LynxOS-178

By default, the ARINC 653 support is statically installed in a dummy configuration. The ARINC 653 support device driver can be installed either dynamically or statically.

Static Installation of ARINC 653 Support

The `$ENV_PREFIX/sys/devices/arinc653info.c` file describes the ARINC 653 configuration being linked into the kernel.

1. Edit the `$ENV_PREFIX/sys/devices/arinc653info.c` file as described in "Configuring the ARINC 653 Support Device Driver" on page 26.
2. Rebuild the ARINC 653 device info file.

```
# cd $ENV_PREFIX/sys/devices
# make install
```

3. Enable ARINC653 support in the `$ENV_PREFIX/sys/bsp.<bsp_name>/config.tbl` file by uncommenting the ARINC driver configuration files as follows:

```
# ARINC653 device driver
I:arinc653.cfg
```

4. Rebuild the kernel.

```
# cd $ENV_PREFIX/sys/bsp.<bsp_name>
# make clean all
```

Dynamic Installation of ARINC 653 Support

The `$ENV_PREFIX/sys/devices/arinc653info.c` file describes the ARINC 653 configuration being linked into the kernel.

1. Edit the `$ENV_PREFIX/sys/devices/arinc653info.c` file as described in "Configuring the ARINC 653 Support Device Driver" on page 26.
2. Rebuild the ARINC 653 device info file:

```
# cd $ENV_PREFIX/sys/drivers.rsc/arinc653
# make install
```

3. Disable ARINC653 support in the `$ENV_PREFIX/sys/bsp.<bsp_name>/config.tbl` file by commenting out the ARINC driver configuration file as follows:

```
#ARINC653 device driver
#I:arinc653.cfg
```

4. Rebuild the kernel:

```
# cd $ENV_PREFIX/sys/bsp.<bsp_name>
# make kdi
```

Use the `arinc653.dldd` file as a character device driver and the `arinc653.info` file as the device driver information file.

Configuring LynxOS-178 for ARINC 653 Support

The support for ARINC 653 APEX interfaces is implemented on top of the existing LynxOS-178 services. The configuration for a certain target is split in several places:

- Configuring the kernel
- Configuring the VCT
- Configuring the ARINC 653 support device driver

Refer to subsections below for more information.

NOTE: The resource limits configured for a certain LynxOS-178 partition should not be more restrictive than the limits specified in the ARINC 653 device configuration file. This applies to the resources set in both VCT and `uparam.h` files.

Configuring the Kernel

Kernel configuration is adjusted by editing the `uparam.h` file and rebuilding the kernel. The following parameters may need to be configured to support a specific ARINC 653 configuration:

Table 5-1: Parameters

Setting	Description
`NPROC`	Total number of processes in the system. In addition to the explicitly configured ARINC 653 processes, the implementation needs 2 extra processes per partition.
`NTHREADS`	Total number of threads in the system. Each process needs one thread, and there needs to be 3 extra threads per partition.

Table 5-2: Settings for Partition 0

Setting	Description
`NUMTOUTS`	The number of kernel timeouts for partition 0. Refer to "Number of Timeouts" on page 30.
`VMZERO_SEMAPHORES`	The number of semaphores for partition 0. Each `CREATE_SEMAPHORE` request uses 1 semaphore.
`VMZERO_MSGQS`	The number of message queues for partition 0. Each `CREATE_BUFFER` request uses 1 message queue.
`NSHM`	The number of shared memory segments for partition 0. Each `CREATE_BLACKBOARD`/ `CREATE_BUFFER` request uses 1 shared memory segment.

Table 5-2: Settings for Partition 0 (Continued)

NUM_NAMES	The number of all POSIX IPC objects in partition 0. Each CREATE_BLACKBOARD/ CREATE_SEMAPHORE request uses 1 POSIX IPC object and each CREATE_BUFFER request uses 2 POSIX IPC objects.
VMZERO_NPROC VMZERO_TOT_THREADS	The number of processes and threads reserved for partition 0. If the partition 0 is used, this numbers should be set as follows: • VMZERO_NPROC: (number of processes in partition 0) + 4 • VMZERO_TOT_THREADS: VMZERO_NPROC + 1 + (number of kernel threads created by the installed drivers). Two of the ARINC 653 per-VM threads are created from VM0 before the partitioning is started, so the VM0 thread ID pool is used for additional threads.

Enabling Health Monitor Signal Catching

Fatal signal catching by the Health Monitor is configurable in Development Mode. By default, the fatal signals are not processed by the Health Monitor and the reaction of the system to such signals conforms to POSIX.

By setting the HM_CATCH_FAULTS macro to 1 in the uparam.h file, the system may be configured to catch the fatal signals using the Health Monitor.

In Production Mode, fatal signals are always processed by the Health Monitor.

Configuring the VCT

For partitions other than partition 0, resource limits are configured using the VCT (settings for partition 0 are ignored). The following parameters may need to be configured:

Table 5-3: VCT Configuration Parameters

Setting	Description
NumOfProcessesLim	The number of processes in a partition. Should be at least 2 more than the number of ARINC 653 processes.
NumOfThreadsLim	The number of threads in a partition. Should be 1 more than the NumOfProcessesLim value.
NumOfMsgQueuesLim NumOfSharedMemObjsLim NumOfSemaphoresLim	The number of the POSIX IPC objects. The same as VMZERO_MSGQS, NSHM, VMZERO_SEMAPHORES parameters described in Table 5-2.
NumOfUnixStreamSockLim	The maximum number of a UNIX domain stream sockets for the VM. The value of NumOfUnixStreamSockLim can be between 0 and 2^{31}, inclusive.

Configuring the ARINC 653 Support Device Driver

The ARINC 653 support device driver is configured using the device information file (sys/devices/arinc653info.c). Type definitions for structures mentioned below are described in the sys/dheaders/arinc653info.h file.

Table 5-4: Configuring Device Driver

Structure /Array	Contents
arinc653_info structure	• A pointer to the memory blocks info table • The number of defined memory blocks • A pointer to the channel info table • The number of defined channels • A pointer to the ports info table • The number of defined ports

Table 5-4: Configuring Device Driver (Continued)

`arinc653_info_mblock array`	For each memory block, the following information is specified: • Memory block name • Memory block address • Memory block size • Read-only mask – bitmask of partitions that have read-only access to the memory block • Read-write mask – bitmask of partitions that have read-write access to the memory block
`arinc653_info_channel array`	For each inter-partition communication channel, the following information is specified: • Channel name • Maximum message size for that communication channel • Maximum number of messages in the channel FIFO • The channel's flow control mode (sampling or queueing) • The channel's flow control mode (block, drop, or overwrite) • Whether the channel is unicast or broadcast
`arinc653_info_port array`	For each port (sampling or queuing), the following information is specified: • The partition to which the port belongs • Port name • Port direction (`SOURCE/DESTINATION`) • Maximum number of messages in the port's FIFO

Example

Below is an example of how to configure queuing ports. It sets up 2 channels for queuing ports between partitions 0 and 1. `QUEUING_PORT1` and `QUEUING_PORT3` are connected using `CHANNEL-0`, and `QUEUING_PORT2` and `QUEUING_PORT4` are connected using `CHANNEL-1`. `QUEUING_PORT1` and `QUEUING_PORT4` are the sources from partitions 0 and 1 respectively. `QUEUING_PORT2` and `QUEUING_PORT3` are the destinations for partition 0 and 1 respectively.

Data can be sent from partition 0 to partition 1 by writing to QUEUING_PORT1, and received at partition 1 by reading from QUEUING_PORT3.

Data can be sent from partition 1 to partition 0 by writing to QUEUING_PORT4, and received at partition 0 by reading from QUEUING_PORT2.

```
static const struct arinc653_info_mblock arinc653_info_mblocks[] = {
};

static const struct arinc653_info_channel arinc653_info_channels[] = {
  {
    .name = "CHANNEL-0",
    .max_message_size = 40,
    .max_nb_messages = 5,
    .mode = ARINC653_INFO_CHANNEL_QUEUING,
    .flow_control = ARINC653_INFO_CHANNEL_FLOW_CONTROL_BLOCK,
    .broadcast = ARINC653_INFO_CHANNEL_UNICAST
  }, {
    .name = "CHANNEL-1",
    .max_message_size = 40,
    .max_nb_messages = 5,
    .mode = ARINC653_INFO_CHANNEL_QUEUING,
    .flow_control = ARINC653_INFO_CHANNEL_FLOW_CONTROL_BLOCK,
    .broadcast = ARINC653_INFO_CHANNEL_UNICAST
  }
};

static const struct arinc653_info_port arinc653_info_ports[] = {
  {
    .vm = 0,
    .name = "QUEUING_PORT1",
    .direction = SOURCE,
    .channel = "CHANNEL-0"
  }, {
    .vm = 0,
    .name = "QUEUING_PORT2",
    .direction = DESTINATION,
    .channel = "CHANNEL-1"
  }, {
    .vm = 1,
    .name = "QUEUING_PORT3",
    .direction = DESTINATION,
    .channel = "CHANNEL-0"
  }, {
    .vm = 1,
    .name = "QUEUING_PORT4",
    .direction = SOURCE,
    .channel = "CHANNEL-1"
  }
};

const struct arinc653_info arinc653_info = {
  .mblocks = arinc653_info_mblocks,
  .nmblocks = sizeof(arinc653_info_mblocks)/sizeof(arinc653_info_mblocks[0]),
  .channels = arinc653_info_channels,
  .nchannels =
sizeof(arinc653_info_channels)/sizeof(arinc653_info_channels[0]),
  .ports = arinc653_info_ports,
  .nports = sizeof(arinc653_info_ports) / sizeof(arinc653_info_ports[0])
};
```

Implementation-specific Details

The ARINC 653 standard defines the following constants as implementation-dependent:

Constant Name	Constant Value	Scope
SYSTEM_LIMIT_NUMBER_OF_PARTITIONS	16	Module Scope
SYSTEM_LIMIT_NUMBER_OF_MESSAGES	512	Module Scope
SYSTEM_LIMIT_MESSAGE_SIZE	8192	Module Scope
SYSTEM_LIMIT_NUMBER_OF_PROCESSES	63	Partition Scope
SYSTEM_LIMIT_NUMBER_OF_SAMPLING_PORTS	512	Partition Scope
SYSTEM_LIMIT_NUMBER_OF_QUEUING_PORTS	512	Partition Scope
SYSTEM_LIMIT_NUMBER_OF_BUFFERS	256	Partition Scope
SYSTEM_LIMIT_NUMBER_OF_BLACKBOARDS	256	Partition Scope
SYSTEM_LIMIT_NUMBER_OF_SEMAPHORES	256	Partition Scope
SYSTEM_LIMIT_NUMBER_OF_EVENTS	256	Partition Scope

In LynxOS-178, the number of partitions is limited to a total of 16 partitions, including the system partition (partition 0).

Therefore, the maximum number of partitions available to the ARINC application (`SYSTEM_LIMIT_NUMBER_OF_PARTITIONS`) is 16.

Maximum number of processes that a single partition can handle is 64, including the partition initialization process. Therefore, `SYSTEM_LIMIT_NUMBER_OF_PROCESSES` is set to 63.

Partition 0 Issues

Due to the special nature of partition 0 in LynxOS-178, the resource limits for it cannot be set in the VCT. Instead, its resource limits are set in the `uparam.h` file. By default, there are only 10 processes reserved for partition 0. To allow more ARINC 653 processes to be created, either any other partition should be used, or the default values should be increased.

Number of Threads and Processes

Concerning the limits on the number of threads and processes for a certain partition, please keep in mind that each partition needs 2 extra processes and corresponding threads allocated to each partition's pool plus 2 extra threads allocated to partition 0's pool (in addition to those needed for the resources specified for the ARINC 653 configuration). The extra resources are needed because the error handler process constitutes a separate process. Hence, its resources need to be accounted for in the configuration.

Number of Timeouts

The timeout is a kernel resource representing a delayed action. The LynxOS-178 Kernel allocates 1 timeout for each process and 1 timeout for each thread in every partition. For partition 0, the number of timeouts is configured in the `uparam.h` file. The ARINC 653 uses timeouts for the following actions:

- Delayed process start (the `DELAYED_START` service request)
- Sleeping (the `TIMED_WAIT` service request)
- Suspending a process (the S`USPEND` service request)
- Process deadline tracking (Tracking release points of periodic processes) The following timeout usage should be noted:
- A process that has not yet started does not have deadlines or release points set.
- If a process has a `DELAYED_START` timeout, it cannot have any other timeouts. It would therefore have 1 timeout associated with it.
- If a process is non-periodic, it can't sleep and suspend at the same time, so a deadline plus a sleep (or suspend) timeout can be associated with it. This means that 2 timeouts (deadline + sleep or suspend = 2 timeouts) can be associated with a non-periodic process.
- A periodic process cannot be suspended, but can be put to sleep with `TIMED_WAIT.`

This means that the worst case timeout demand would be when a periodic process with a deadline calls `TIMED_WAIT` to sleep, thus using 3 timeouts (one for the process being periodic, one for its deadline, and one for the timed wait).

If multiple ARINC processes try to use timeouts, and a sufficient number have not been allocated to the partition, it is possible for the pool of timeouts to be exhausted. The number of timeouts allocated to the partition should be increased accordingly to prevent this from happening.

Thread Priorities

The implementation of the ARINC 653 support uses internal priorities in the range of 0-247, which are used as follows:

0	The priority of a per-partition idle thread
1-120	Priorities reserved for use by the ARINC 653 processes
121-247	Temporarily boosted priorities

Worth noting is that the preemption lockout is implemented by raising the priorities of the idle thread and the current process to 121 and 122, respectively. A deadlock can occur if the process invokes services of a driver that passes a request to a kernel thread with lower priority (and priority inheritance is not used).

The `tty` driver is known to have this problem.

Compiling an ARINC 653 Application

LynxOS-178 provides the API specified by the ARINC 653 standard in the `libarinc653.a` library. Therefore, an ARINC 653 application is compiled in the same way as a regular LynxOS-178 application and linked against the `libarinc653.a` library, for example:

```
$ gcc -o myapp myapp.c -larinc653
```

NOTE: The implementation of ARINC 653 service requests uses POSIX interfaces wherever appropriate. Therefore, the following warning is issued:

```
MIXING ARINC653 AND POSIX INTERFACES IN THE SAME APPLICATION MAY LEAD TO
UNDEFINED BEHAVIOR.
```

Also, the process must exit only as a result of the `STOP/STOP_SELF` quests. Exit by performing `return` from the `main`() function will cause undefined behavior.

Debugging an ARINC 653 Application

Since an ARINC 653 application is a regular LynxOS-178 executable, the same debugging techniques are applicable to debug an ARINC 653 application that are used for POSIX applications.

CHAPTER 6 Overview of Reusable Software

The LynxOS-178 reusable software architecture is shown in Figure 6-1. The basic components of this architecture consist of the LynxOS-178 Partitioning Kernel & System Services (API). The sections below describe each of these components in more detail.

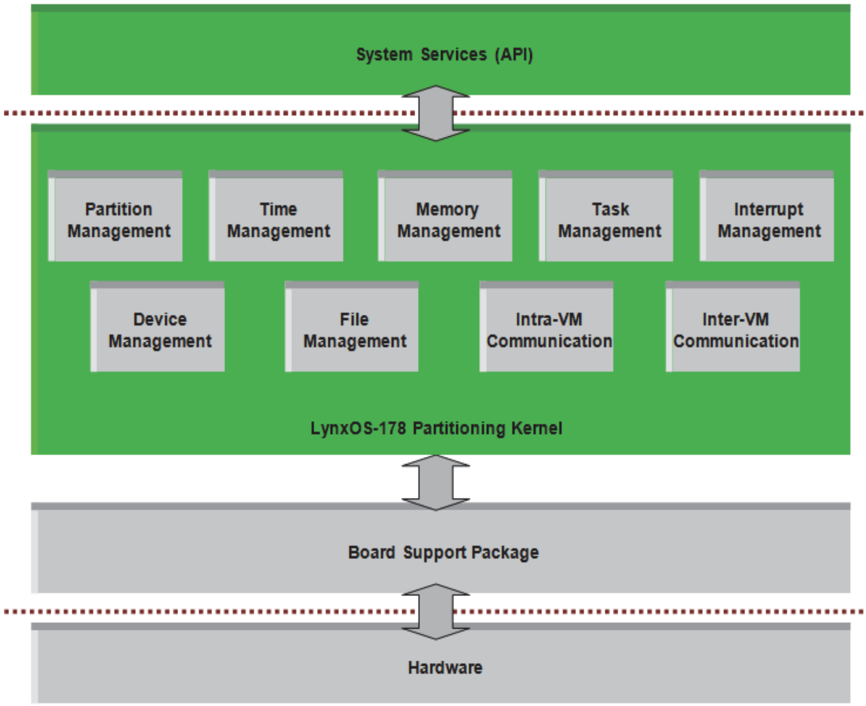

Figure 6-1: LynxOS-178 Software Architecture

LynxOS-178 Partitioning Kernel

The LynxOS-178 Partitioning Kernel provides deterministic time, memory, and resource partitioning in accordance with the ARINC 653 specification as well as hard real-time multitasking capabilities.

Partition Management

The partitioning management subcomponent handles creation and management of time and space requirements of a partition. It consists of time partitioning, resource partitioning, and memory partitioning.

Time Partitioning

Figure 6-2 shows a two-stage scheduling technique that involves scheduling of VMs as the primary stage and scheduling of individual processes/threads within a VM as the second stage. The specific methodology used to implement time partitioned scheduling involves the following:

- The LynxOS-178 Kernel determines which VM should be given the processor based on a fixed, preset schedule. A hardware timer interrupt signals the end of a time period for a VM and initiates a context switch.
- The VM's threads are then scheduled based on their priorities using standard POSIX scheduling policies. Priorities are based on extended rate monotonic analysis, where faster threads have higher priority to ensure completion of specific deadlines

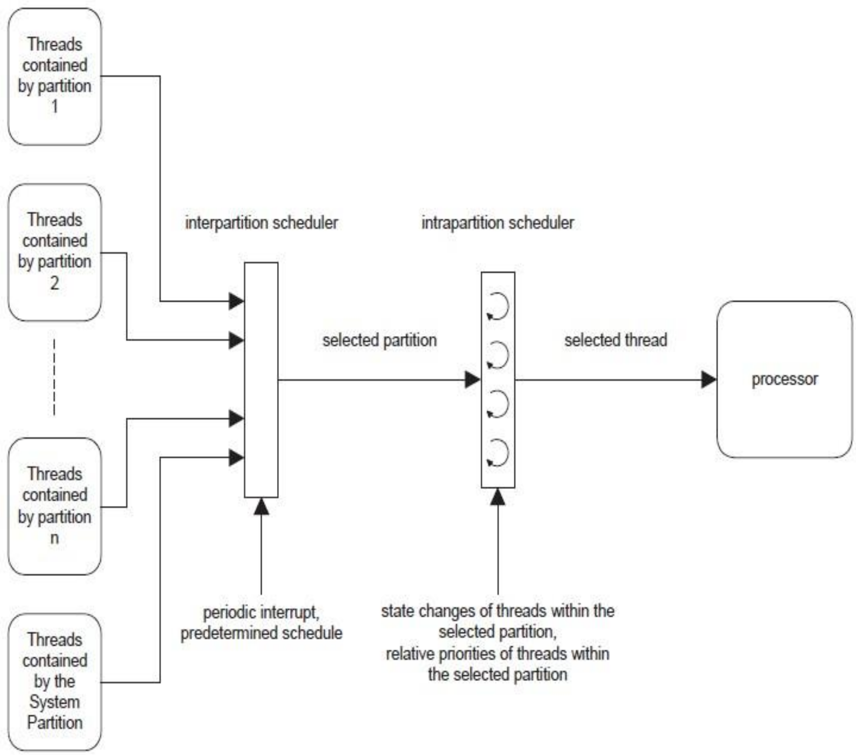

Figure 6-2: LynxOS-178 Scheduler

Execution time durations are defined for each partition based on their real-time deadlines. Also, each partition includes its processing time required to interact with the LynxOS-178 software. For example, the execution time duration for a partition with intensive input/output functionality includes the access time to the device driver for sending and retrieving data.

The LynxOS-178 scheduler is driven by a 1 KHz hardware interrupt. The first level, or interpartition scheduler, is an ARINC 653 like fixed cyclic, time- slicing scheduler. The System Integrator allocates processor time to partitions by specifying a schedule in the VCT, in integer units of timer ticks (1 millisecond default period), with each VM included in the schedule at least once. This schedule represents one repeating major frame. The second level, or intra-partition scheduler, is a priority-preemptive scheduler that is qualified for hard real-time determinism.

Within a partition, the scheduler ensures that the thread with the highest priority that is ready to run will execute. If threads in the minor frame are given the same priority, then they will run sequentially in first-in first-out order. When a thread blocks or terminates, the scheduler will chose another thread to run, favoring the highest priority. Thus, each partition is allocated a repeating time slice called a minor frame, and within the minor frame, threads are scheduled by priority.

For example, a schedule of `0[5] 1[5] 2[20] 1[5] 3[15]` in the VCT specifies a 50 ms repeating major frame, with P0 getting the first 5 ms, P1 getting the next 5 ms, P2 getting the next 20 ms, P1 again getting 5 ms, and finally P3 getting the last 15 ms. Since the sum of the minor intervals is 50ms, the time-slicing scheduler runs at a period of 50 ms per major frame, or 20 major frames per second. P0, P2, and P3 likewise all run at the rate of 20 executions per second. Since P1 occurs twice in the major frame, and its two occurrences begin 25 ms apart, P1 runs at a period of 25 ms per minor frame, or 40 executions per second (20 major frames per second * 2 execution times per major frame).

Memory Partitioning

The hardware MMU provides physical memory partitioning for each partition and logical address partitioning for each process. The MMU maps the logical address space onto the physical memory address space for each partition. The memory resources for each partition are assigned at configuration time, allocated at initialization time, and the limits are enforced at run time. The MMU hardware limits access to the partition's memory by other partitions, preventing errant partitions from corrupting the code and data space of other partitions or the LynxOS-178 Kernel. The LynxOS-178 Kernel thus uses the MMU of the processor to enforce strict memory partitioning.

Resource Partitioning

Shared resources are allocated to partitions statically. Shared Resources include items such as kernel data structures, memory, I/O, or file systems. Partitions are prevented from using more than their defined resources. Partitions are also prevented from writing to any other partition's resources. The LynxOS-178 Kernel thus supports resource partitioning for the system as a whole.

Memory Management

The memory management subcomponent of LynxOS-178 provides management of user and kernel memory. This consists of allocation and deallocation of memory to components as well as memory reclamation of unused memory.

Task Management

LynxOS-178 provides the capability to create processes and threads (collectively referred to as tasks) within the context of a partition. These constitute the primary application programs that execute within the partition. The LynxOS-178 Kernel provides mechanisms to create, execute, and terminate processes and threads in the context of a specific partition.

Interrupt Management

LynxOS-178 provides the capability to handle interrupts to the Operating System by external events. The Interrupt Management subcomponent provides the generic mechanisms to enable handling of interrupts and exceptions to the Operating System.

Device Management

LynxOS-178 provides a hardware-independent mechanism for applications to access peripheral devices. The Device Management subcomponent of the kernel provides the ability for applications to read and write to peripheral devices as well as control their operational configuration.

File Management

LynxOS-178 provides a hardware-independent mechanism for applications to access file systems. The File Management subcomponent of the kernel provides the ability for applications to read and write to files on a file system on any persistent storage medium.

Intra-partition communication

The LynxOS-178 Kernel provides several service functions for intra-partition communication between distinct processes or threads, or both. Applications can request these services from the kernel through the System Services (API).

Inter-VM communication

Inter-VM communication is provided through the ARINC 653 API. These services can be requested by applications through the System Services (API).

Time Management

The time management subcomponent of LynxOS-178 manages the timers and clocks to allow applications access to real-time timers within the context of an individual partition. These services can be requested by applications through the System Services (API).

System Services (API)

LynxOS-178 provides basic system services in accordance with Portable Operating System Interface (POSIX) standard 1003.1, 1003.1b, and 1003.1c, including:

- Scheduling
- Message queues, pipes, and sockets
- Memory management
- Exception handling
- Shared memory management
- Rendezvous of process threads
- Real-time POSIX queued signals
- Clocks and timers
- Semaphores
- Mutexes
- File system services

LynxOS-178 also provides services according to the RTAA ARINC 653 APEX, including:

- Partition management
- Process management
- Time management
- Inter-partition communications
- Intra-partition communications
- Blackboard services
- Semaphore services

- Event services
- Health monitoring
- File system services
- Memory blocks

These system services constitute the mechanism for applications to communicate with other applications within a partition. The system services component is also the sole mechanism for applications to request services from the LynxOS-178 Kernel. The application calls for system services are converted to software traps that provide entrance into the kernel.

The System Services (API) also provides mathematical functions to perform floating point, trigonometric, and logarithmic functions.

BSP

The BSP is not considered a LynxOS-178 reusable software component since it must be created for each board. LynxOS-178 does provide defined interface requirements that must be satisfied to interface properly with a new board. BSP components are a collection of hardware-specific routines necessary to perform functions necessary for the kernel on the underlying hardware platform. The platform-independent kernel software calls routines in the BSP to perform operations that require a hardware-specific implementation. A unique BSP implementation is created for each board that the LynxOS-178 architecture executes on. The integrator is responsible for obtaining approval of the BSP from the certification authority and demonstrating that it complies with the LynxOS-178 BSP requirements document.

The System Integrator and Partitions

From the System Integrator’s point of view, a partition is the smallest unit of resource allocation and fault containment, isolation, and recovery. The System Integrator sets the partitioning policies, and in many cases, configures the resources and fault handling as well.

Resources that are not allocated to the target system as a whole are allocated to specific partitions by the user. Most of the information the System Integrator uses to allocate resources to partitions is based on information supplied by the application developers.

CHAPTER 7 *Understanding the Virtual Machine Configuration Table*

Purpose

This chapter defines the syntax and format of the Virtual Configuration Table.

Virtual Machine Configuration Table

System Overview

The VCT contains configuration information for the target system and the partitions. The information is used to configure LynxOS-178 and platform applications. In addition, it contains limited configuration information for each partition (for example, partition-specific environment variables).

Access to the VCTs is controlled through file permissions and is intended to be limited to partition 0. When the system is booted, `init` is the first process to execute. It uses the VCT to configure LynxOS-178, load device drivers, and mount file systems. It also passes the corresponding configuration information to each partition via environment variables and parameter lists.

The VCT is an ASCII file that is loaded into the target system with the KDI. The file is marked in the file system on the target as Read-Only. All accesses to the VCT are via the file system services. Currently, the user needs to put the same VCT into `/usr/etc`, and `/usr/local/etc`.

Format and Syntax

This section describes the organization of the VCT. The subsequent sections contain detailed information for each field in the table.

The Contents of the VCT

The VCT consists of an ASCII text file containing two types of configuration objects: scalars and tables. Scalars are objects for which there is at most one instance. Table objects contain one or more entries, where each row defines one instance of a partition, a dynamic device driver, or a file system. The following formal description shows the organization of the VCT.

The VCT is organized into four areas. The first area contains information relating to the target system or all partitions (scalar objects). The second area contains a table of information specific to each partition (table objects). The third area contains a table of information relating to each dynamic device driver or device to install (table objects). The last area contains information relating to each configurable file system to mount (table objects).

The Syntax of the VCT

The method used to describe the syntax of the VCT is Backus-Naur Form (BNF). This formal method is commonly used in defining the syntax of programming languages. The VCT can be thought of as a simple program language that configures the platform software.

The following is the BNF definition of the VCT. Notice that terminal symbols, whether literal digits, letters, or names not in angle brackets, are shown in boldface. Nonterminal symbols are enclosed between < and >. If “<” or “>” are required in the syntax then they will be enclosed in single quotes ('<' and '>').

```
'<'VM < table id > '>' <line delimiter>
   <vct table> ::= <scalar> <vm table> | <ddd table> | <fs table>
   <scalar> := VctCrc   = <hexadecimal> | Null ; <line delimiter>
   VctCpn               = <string> ; <line delimiter>
   VctVersion           = <decimal> ; <line delimiter>
   NumOfVms             = <decimal> ; <line delimiter>
   NumOfDdds            = <decimal> ; <line delimiter>
   NumOfFss             = <decimal> ; <line delimiter>
   ActionOnModuleErr    = <decimal> ; <line delimiter>
   ColdStartSchedule    = <string> ; <line delimiter>
   WarmStartSchedule    = <string> ; <line delimiter>
   RunTimeSchedule      = <string> ; <line delimiter>
   ColdStartDuration    = <decimal> ; <line delimiter>
   WarmStartDuration    = <decimal> ; <line delimiter>
   IbitDuration         = <decimal> ; <line delimiter>
   GroupIds             = <string> ; <line delimiter>
   LogicalName          = <string> ; <line delimiter>
   CommandLine          = <string> ; <line delimiter>
   EnvironmentVars      = <string> ; <line delimiter>
   StdInNodeFname       = <unix fname> ; <line delimiter>
   StdOutNodeFname      = <unix fname> ; <line delimiter>
   StdErrNodeFname      = <unix fname> ; <line delimiter>
   WorkingDir           = <unix path> ; <line delimiter>
   RamFsMount           = <unix path> ; <line delimiter>
```

```
    RamFsLim                 = <decimal> ; <line delimiter>
    RamFsNumOfInodes         = <decimal> ; <line delimiter>
    ActionOnVmErr            = <decimal> ; <line delimiter>
    ActionOnSigillExcp       = Default | Override ; <line delimiter>
    ActionOnSigfpeExcp       = Default | Override ; <line delimiter>
    ActionOnSigsegvExcp  = Default | Override | POSIX ; <line delimiter>
    ActionOnSigbusExcp       = Default | Override ; <line delimiter>
    SysRamMemLim             = <decimal> ; <line delimiter>
    KernHeapMemLim           = <decimal> ; <line delimiter>
    NumOfProcessesLim        = <decimal> ; <line delimiter>
    NumOfThreadsLim          = <decimal> ; <line delimiter>
    NumOfTimersLim           = <decimal> ; <line delimiter>
    FsCacheLim               = <decimal> ; <line delimiter>
    FsCacheAttr            = WriteThrough | WriteBack ; <line delimiter>
    NumOfOpenFdsPerVmLim     = <decimal> ; <line delimiter>
    NumOfMsgQueuesLim        = <decimal> ; <line delimiter>
    NumOfPipesLim            = <decimal> ; <line delimiter>
    NumOfSharedMemObjsLim    = <decimal> ; <line delimiter>
    NumOfSemaphoresLim       = <decimal> ; <line delimiter>
    NumOfUnixStreamSockLim   = <decimal> ; <line delimiter>
'<'/VM < table id > '>' <line delimiter>

<ddd table> ::=
'<'DDD < table id > '>' <line delimiter>
   Type                    = c | b ; <line delimiter>
   DriverId                = <decimal> ; <line delimiter>
   ObjectFname             = <unix fname> ; <line delimiter>
   InfoFname               = <unix fname> ; <line delimiter>
   NumOfMinorDevs          = <decimal> ; <line delimiter>
   BaseCharNodeFname       = <unix fname> ; <line delimiter>
   BaseBlockNodeFname      = <unix fname> ; <line delimiter>
   OwnerId                 = <decimal> ; <line delimiter>
   GroupId                 = <decimal> ; <line delimiter>
   Permissions             = <octal> ; <line delimiter>
   '<'/DDD < table id > '>' <line delimiter>

<fs table> ::=
'<'FS < table id > '>' <line delimiter>
   Mount                   = <unix path> ; <line delimiter>
   NodeFname               = <unix fname> ; <line delimiter>
   NumOfInodes             = <decimal> ; <line delimiter>
   MkffsArgs               = <string> ; <line delimiter>
   FfsckArgs               = <string> ; <line delimiter>
   OwnerId                 = <decimal> ; <line delimiter>
   GroupId                 = <decimal> ; <line delimiter>
   Permissions             = <octal> ; <line delimiter>
   '<'/FS < table id > '>' <line delimiter>

<table id>::= <digit> | <digit>

<unix path>            ::= / | / <path string> /
<unix fname>           ::= / <path string>
<path string>          ::= <path char> | <path char> <path string>
<path char>            ::= <letter> | <digit> | <special char>
<string>               ::= <character> | <character> <string>
<character>            ::= <letter> | <digit> | <special char> | ' '
<special char>         ::= . | - | _ | / |
<letter>               ::= a | b | c | … z | A | B | C | … Z
<hexadecimal>          ::= 0x <hex string>
<hex string>           ::= <hex char> | <hex char> <hex string>
```

```
<hex char>      ::= A | B | C | D | E | F | a | b | c | d | e | f | <digit>
<octal>               ::= 0 <octal sting>
<octal string>        ::= <octal char> | <octal char> <octal string>
<octal char>          ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7
<decimal>             ::= <pos digit> | <pos digit> < decimal string>
<decimal string>      ::= <digit> | <digit> < decimal string>
<digit>               ::= 0 | <pos digit>
<pos digit>           ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<line delimiter>      ::= LF | EOF
```

Data Value Definition

The BNF formal method is a context-free grammar that is ideal for describing the syntax of programming languages. It cannot adequately describe the semantics of the language. In addition, it cannot define context-sensitive conditions that are illegal. Formal methods for describing the context-sensitive syntax are not very well known and are difficult to understand. Because of this, the following is the informal context-sensitive conditions of the VCT. In addition, the following three sections define the semantics of the VCT.

1. For each table instance (entry), the <table id> tag names should be identical (for example, <VM0> … </VM0> to indicate partition 0's table data).
2. The <table id> number of the first table instance should be 0 and incremented by one for each subsequent entry (that is, table IDs must be consecutive and ascending).
3. The following strings should be reserved and cannot be used in a <string>: "//" and ";". The "//" defines a comments line or string (as in C++) where all characters after the "//" until LF or EOF are considered comments and are ignored when parsing. Because strings are not enclosed in quotes, the ";" is used to terminate <string>.
4. The size of the <hexadecimal> format should be 1 to 8 characters. The range of the hexadecimal number is specified with the description of the data values.
5. The size of the <octal> format should be 1 to 3 characters. The range of the octal number is specified with description of the data values.
6. The size of the <decimal> format should be 1 to 10 characters. The range of the decimal number is specified with description of the data values.

7. A <unix path> is a string of subdirectory names separated and enclosed by forward slashes (/). <unix path> should be absolute, starting with a forward slash (indicating the root directory) and terminated by a forward slash.
8. A <unix fname> should follow the <unix path> definition with the filename appended.
9. The maximum length of <unix path> and <unix fname> is 255 characters.
10. The definition of <string> is ambiguous with respect to <unix path> and <unix fname>. This was done intentionally to increase the readability of the grammar. In the future, it is recommended that <string> be enclosed in quotes to resolve this problem.
11. The maximum length of <string> is 1023 characters.
12. The VCT should use the UNIX line-end convention (that is, new line character 0xA versus the DOS convention of 0xD 0xA pair).

Notice that hexadecimal strings must begin with “0x,” octal strings must begin with a leading zero, decimal numbers cannot have leading zeros, and enumerated values must start with a character. This allows the `atoi()` or `catoi()` C library functions to be used to convert the strings to integer values.

Module Information Scalars

VctCrc

The VctCrc contains the Cyclic Redundancy Check (CRC) value for the VCT. The CRC should be created starting with the first byte after the `VctCrc` ending delimiter character (“;”) and ending with the last byte in the file.

The VctCrc should be created using a 32-bit CRC defined by the following formula.

$$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^{8} + x^{7} + x^{5} + x^{4} + x^{2} + x + 1$$

The VctCrc data field is optional. If the data field does not exist (that is, `VctCrc=`;), then CRC checking of VCT is disabled. This allows the file to be easily modified during development. In addition, the Health Monitor application uses this field to disable CRC checking of the data load header file, located in the `/usr/etc/` directory, during Continuous Built-In Tests and during Software Validation mode.

VctCpn

Not used.

VctVersion

The `VctVersion` contains the version of the VCT. It allows the software to be backward compatible when the format of the VCT changes. The value should be a number between 0 and 65535.

The `VctVersion` should be incremented when the format of the VCT changes.

The value of `VctVersion` should be set to “`220`.”

NumOfVms

The NumOfVms is the number of partitions in the partition (VM) table. The range should be between 1 and 16, inclusive.

NumOfDdds

The `NumOfDdds` is the number of dynamic device drivers and devices in the DDD table. The range should be between 0 and 48, inclusive. A maximum of 48 device drivers, 16 block devices, and 32 character devices can be installed in this section.

NumOfFs

The `NumOfFs` is the number of File Systems in the FS table. The range should be between 0 and 16, inclusive.

ActionOnModuleErr

The `ActionOnModuleErr` is the action to perform on module fatal errors. The actions are either reset or halt after N errors from power up (cold and warm).

The `ActionOnModuleErr` should be set to 0 if the target system is to be reset on every occurrence of a module fatal error.

The `ActionOnModuleErr` should be set to N, where N is greater than 0 and less than 255, if the target system is be halted after N module fatal errors.

ColdStartSchedule

The `ColdStartSchedule` defines the cold start schedule for the target system. The value specifies the number of system ticks (for example, 1 millisecond) each VM executes during a major (repeating) frame. After the major frame is completed, the schedule is repeated until the `ColdStartDuration` is reached. At this point, the schedule switches to the `RunTimeSchedule`.

The `ColdStartSchedule` value should be a string consisting of zero and one or more "`v [ t ]`" sets separated by white space, where *v* is the partition (VM) number, *t* is the number of system ticks, and `[]` are delimiters. For example, a value of "`0[5] 1[30] 0[10] 2[5]`" defines a major of 50 system ticks, where P0 runs for 5 ticks, followed by P1 for 30 ticks, followed by P0 for 10 ticks, followed by P2 for 5 ticks.

The partition (VM) number, *v*, should be greater than or equal to 0 and less than `NumOfVms`. The system tick, *t*, should be between 0 and $2^{32} - 1$, inclusive.

The number of "`v [ t ]`" sets should be between 0 and $2^{32} - 1$, inclusive. Note, refer to "Data Value Definition" for the maximum string length requirement.

The `ColdStartSchedule` should be the *null* string when the `ColdStartDuration` is *zero*.

NOTE: When the `ColdStartDuration` is zero, the `ColdStartSchedule` is ignored. The upper bound on system ticks and "`v [ t ]`" directly impacts the `SystemRamLim` for P0. The `cinit` process parses the `ColdStartSchedule` string and uses `malloc()` to create an array of bytes equal to the size of the major frame. For example, if the major frame is 100, then a 100 byte array is allocated. If the major frame is 10,000, then a 10,000 byte array is allocated.

WarmStartSchedule

The `WarmStartSchedule` defines the warm start schedule for the target system. The value specifies the number of system ticks (for example, 1 millisecond) each partition executes during a major (repeating) frame. After the major frame is completed, the schedule is repeated until the `WarmStartDuration` is reached. At this point, the schedule switches to the `RunTimeSchedule`.

The same requirements for `ColdStartSchedule` and `ColdStartDuration` apply to the `WarmStartSchedule` and `WarmStartDuration`, with the exception of not allowing a *null* string.

RunTimeSchedule

The `RunTimeSchedule` defines the run-time schedule for the target system. The value specifies the number of system ticks (for example, 1 millisecond) each partition executes during a major (repeating) frame. After the major frame is completed, the schedule is repeated indefinitely.

The same requirements for `ColdStartSchedule` apply to the `RunTimeSchedule`, with the exception of the Start Duration.

ColdStartDuration

The `ColdStartDuration` is the length of time the cold start schedule executes before switching over to the run-time schedule. The value of `ColdStartDuration` is in microseconds, ranging from 0 to $2^{32} - 1$. A value of 0 causes the cold start schedule not to execute.

The value of `ColdStartDuration` should be “15000000” (15 seconds) for a target system.

WarmStartDuration

The `WarmStartDuration` is the length of time the warm start schedule executes before switching over to the run-time schedule. The value of `WarmStartDuration` is in microseconds, ranging from *0 to 2**32 - 1*. A value of 0 causes the warm start scheduler not to execute.

The value of `WarmStartDuration` should be “1000000” (1 second) for a target system.

Virtual Machine Table

GroupIds

The `GroupIds` entry is a string that specifies the supplementary group membership, if any, of the partition. Its purpose is to permit I/O device nodes to be shared among partitions. The string should consist of zero to seven numbers separated by spaces.

NOTE: The target system software allows processes to be members of up to eight groups. One of these is the primary group, leaving seven available for supplementary groups. Multiple memberships may be needed if multiple devices are shared.

The supplementary group IDs, if any, should be listed in numerical order and should not contain duplicates.

NOTE: This is for readability and to prevent `cinit` from having to sort the list.

LogicalName

The `LogicalName` entry is the name of the partition. It is used to logically identify the partition. The `LogicalName` should be a 1- to 63-character string.

CommandLine

The `CommandLine` entry is used to start the master process of the partition. It consists of a sequence of words separated by space characters. All words are passed to the master program in the `argv[]` array. The `argc` parameter is calculated and set appropriately. The `CommandLine` should be a 1- to 1023-character string.

Words should be separated by white space.

A word should be a sequence of characters containing anything except a space, semicolon, or null character. Tabs and new-lines are not separators. They will be considered part of a word.

The first word in the `CommandLine` (process name) should be the full UNIX path name of the binary image to execute (as the partition's master process) and the remaining words are the run-time arguments.

EnvironmentVars

The `EnvironmentVars` entry defines environment variables that are partition-specific. These variables, along with the system-wide variables, are passed to the master process on startup. A list of environment variables is passed to the master program in the `envp[]` array. The `EnvironmentVars` should be a 0- to 1023-character string.

The environment variables should be specified as a list of entries separated by new-line characters.

Each entry should consist of a name and a value separated by an equal sign.

A name should be a sequence of ASCII letters, digits or underscores, beginning with a letter or an underscore.

A value should be a sequence of any ASCII characters except new-line or semicolon.

NOTE: An example of the format can be obtained by logging into any UNIX system and running `/bin/env`.

StdInNodeFname

The `StdInNodeFname` entry specifies the directory path and filename of the Standard In character device node. During initialization, `cinit` opens this device node and redirects Standard In to this device. The device may reference a static or dynamic device driver. The `StdInNodeFname` should be the absolute path and filename of a readable character device node.

NOTE: The `runshell` utility in the Development KDI is used to start a `bash` shell in a partition. This utility opens and redirects the Standard In, Out, and Error devices to the `/dev/tty0` or `/dev/tty1` device, overriding the VCT entries. Setting `StdInNodeFname`, `StdOutNodeFname`, and `StdErrNodeFname` to the `/dev/tty0` or `/dev/tty1` device and starting the `bash` utility will *not* start up a `bash` shell correctly because the `tty` manager requires a special I/O control command to be sent to the driver.

StdOutNodeFname

The `StdOutNodeFname` entry specifies the directory path and filename of the Standard Out character device node. During initialization, `cinit` opens this device node and redirects Standard Out to this device. The device may reference a static or dynamic device driver. The `StdoutNodeFname` should be the absolute path and filename of a writable character device node.

StdErrNodeFname

The `StdErrNodeFname` entry specifies the directory path and filename of the Standard Error character device node. During initialization, `cinit` opens this device node and redirects Standard Error to this device. The device may reference a static or dynamic device driver. The `StdErrNodeFname` should be the absolute path and filename of a writable character device node.

WorkingDir

The `WorkingDir` entry specifies the path name of the partition's working directory (that is, home directory). When `cinit` forks and execs the master process, it sets the current directory to this path. The directory path should be the absolute path of an existing directory. If the directory does not exist or the partition does not have read+execute (r+x) permission for the directory, a partition fatal error will be logged.

RamFsMount

The `RamFsMount` entry specifies the path name of a directory in the root file system where the partition's RAM file system will be mounted. The directory path should be the absolute path or an empty string if there is no RAM file system for this partition.

For partitions with RAM file systems, the `RamFsMount` should be set to `/tmp`.

RamFsLim

The `RamFsLim` entry defines the size of the partition's RAM file system. The value of `RamFsLim` is in bytes and should be from 0 to `SysRamMemLim`.

The `RamFsLim` should be on a page boundary (for example, 4096 byte increments for PPC CPUs).

If a RAM file system is not needed, then the value should be set to 0.

RamFsNumOfInodes

The `RamFsNumOfInodes` entry defines the maximum number of inodes allowed in the file system. One inode is required for every file and directory. The value of `RamFsNumOfInodes` should be between 4 and the maximum number of data blocks, where the number of data block is equal to `RamFsLim` / 512. The target system software will truncate the value to a multiple of 4.

NOTE: If the `RamFsLim` entry is set to zero, then the target system's system software ignores this entry.

ActionOnVmErr

The `ActionOnVmErr` is the action to perform on partition (VM) fatal errors. The actions are either reset or halt after *N* errors from the current power cycle (cold and warm).

The `ActionOnVmErr` should be set to 0 if the partition (VM) should be reset on every occurrence of a partition fatal error.

The `ActionOnVmErr` should be set to *N*, where *N* is greater than 0 and less than 2**32 - 1, if the partition (VM) should be halted after *N* partition fatal errors.

NOTE: If the master process exits with code `0` (via the `exit()` function) the process will be restarted by `cinit` without an error being logged. If the application exits with code `254`, the partition will not be restarted and a partition fatal error will be printed on screen. If the partition exits with error codes `1-253`, `255`, then a partition fatal error will appear on screen, and `cinit` will use the `ActionOnErr` value to determine whether to restart or halt the partition.

ActionOnSigillExcp

The `ActionOnSigillExcp` entry identifies how the target system software will handle a SIGILL exception when it occurs. This processor exception occurs when the processor executes an illegal instruction.

The value of `ActionOnSigillExcp` should be either `Default` or `Override`. Default forces a partition fatal error, and `Override` allows the partition to handle the exception.

The `ActionOnSigillExcp` should be set to `Default` for all partitions except for the first partition (the system partition, with `UserId=0`).

The `ActionOnSigillExcp` should be set to `Override` for the first partition (the system partition, with `UserId=0)`.

ActionOnSigfpeExcp

The `ActionOnSigfpeExcp` entry identifies how the target system software will handle a SIGFPE exception when it occurs. This processor exception occurs on floating point errors.

The value of `ActionOnSigfpeExcp` should be either `Default` or `Override`. Default forces a partition fatal error, and `Override` allows the partition to handle the exception.

The `ActionOnSigfpeExcp` should be set to `Override` for the first partition (the system partition, with `UserId=0`).

ActionOnSigsegvExcp

The `ActionOnSigsegvExcp` entry identifies how the target system's system software will handle a SIGSEGV exception when it occurs. This processor exception occurs when Memory Management Unit (MMU) detects a segment violation.

The value of `ActionOnSigsegvExcp` should be either `Default`, `Override`, or `POSIX`. Default forces a partition fatal error, `Override` and `POSIX` allows the partition to handle the exception.

The `ActionOnSigsegvExcp` should be set to `Default` for all partitions except for the first partition (the system partition, with `UserId=0`).

The `ActionOnSigsegvExcp` could be set to `Override` for the first partition (the system partition, with `UserId=0`).

The `ActionOnSigsegvExcp` could be set to `POSIX` for the first partition (the system partition, with `UserId=0`).

The `POSIX` value is required to generate core files. Core files are written to `/rwfs/CoRE`, where `rwfs` is mounted writable file system.

CoreLimit

When `ActionOnSigsegvExcp` is set to `POSIX` value `CoreLimit` parameter shall be large enough to hold all of the data from the dumped process.

ActionOnSigbusExcp

The `ActionOnSigbusExcp` entry identifies how the target system’s system software will handle a SIGBUS exception when it occurs. This processor exception occurs when an unauthorized access to memory occurs.

The value of ActionOnSigbusExcp should be either Default or Override. Default forces a partition fatal error, and Override allows the partition to handle the exception.

The ActionOnSigbusExcp should be set to Default for all partitions except for the first partition (the system partition, with UserId=0).

The ActionOnSigbusExcp should be set to Override for the first partition (the system partition, with UserId=0).

SymRamMemLim

The SysRamMemLim entry specifies the total amount of system RAM allocated to this partition. This includes memory used by the partition's application(s) (for example, text, data, stack, and heap segments) and internal kernel resources required to support a partition. The RAM file system size entry (RamFsLim) and the kernel resource limit entries (for example, NumOfProcessesLim and NumOfThreadsLim) directly impact the SysRamMemLim value.

The value of the SysRamMemLim is specified in bytes with a maximum value equal to the total amount of system RAM on the target system (for example, 32MB - 0x2000000, 64mb - 0x4000000). The target system software will truncate the value down to the nearest page boundary (for example, the nearest 4096-byte boundary on PPC CPUs).

The sum of the SysRamMemLim values for all partitions should not exceed the total amount of system RAM on the target system.

KernHeapMemLim

The KernHeapMemLim entry specifies the kernel heap memory limit. The minimum limit for KernHeapMemLim = 1048576.

NumOfProcessesLim

The NumOfProcessesLim entry defines the maximum number of processes on the target system allocated to this partition. Any attempts by the partition to exceed this allocation will be rejected. The value of NumOfProcessesLim should be between 2 and 64, inclusive.

NOTE: The `cinit` process forks and execs other processes to accomplish certain tasks (as well as to create the user application), therefore there must be at least two processes allocated.

The sum of the `NumOfProcessesLim` values for all partitions should be less than or equal to 64.

NOTE: This entry is not configurable for partition 0.

NumOfThreadsLim

The `NumOfThreadsLim` entry defines the maximum number of threads on the target system allocated to this partition. Any attempts by the partition to exceed this allocation will be rejected. The value of `NumOfThreadsLim` should be between 2 and 256, inclusive. This value must be equal to or greater than `NumOfProcessesLim` because a process always has a main thread.

The sum of the `NumOfThreadsLim` values for all partitions should be less than or equal to 256.

NOTE: When setting thread and process limits in the VCT, you need to keep the following in mind:

LynxOS-178 has undergone partitioning analysis, for up to 64 processes and 256 threads running on the system. Although it is possible to set larger values, systems that need to be certified should not use more than these upper limits.

If the sum of processes or threads in all partitions exceeds the system limit, one or more partitions will be denied their quota as set in the VCT, causing the partition to terminate.

The number of threads in the VCT does not distinguish between threads used in the kernel or threads spawned by a process. For example, if you are given 50 threads and 20 processes in a partition, you can have any of the following combinations (keep in mind that a process is created with one thread for `main()`).

20 processes that spawn no more than 30 threads total

50 kernel threads and no processes

1 process with 49 spawned threads

In LynxOS-178, every process has a minimum of one thread, the main thread. It is possible to exceed the thread limit by creating processes. For example, assume you are given 50 threads and 20 processes in a VM. If a process uses 40 threads (including the main thread), then at most 10 more processes can be created. This means that even though the partition has 20 processes allocated to it, only 11 processes can run as the partition has used up its quota of threads. A similar problem can occur when dealing with pipes and file descriptors.

This entry is not configurable for partition 0.

NumOfTimersLim

The `NumOfTimersLim` entry defines the maximum number of POSIX timers available through `timer_create()` on the target system, allocated to this partition. Any attempts by the partition to exceed this allocation will be rejected. The value of `NumOfTimerLim` should be between `1` and `0x7fff` inclusive.

NOTE: This entry is not configurable for partition 0.

FsCacheLim

The `FsCacheLim` entry defines the size of the file system cache located in system RAM. One cache area is used for all file systems accessible by this partition. The sizing of the cache will directly affect the performance of the file systems, since all read and write data goes through the cache.

The value of the `FsCacheLim` is specified in bytes, ranging from *`X`* to `SysRamMemLim` where *`X`* equals 4 times the largest block size of all file systems. The value of the `FsCacheLim` should be a multiple of 2048.

NOTE: This entry is not configurable for partition 0.

FsCacheAttr

The `FsCacheAttr` entry specifies the file cache attribute for the file system cache. One cache area is used for all file systems accessible by each partition. The `FsCacheAttr` should be set to `WriteThrough` or `WriteBack`. For write-through, writes to a file system will be written to cache and to the physical media during the `write()` system call. For write-back, writes to a file system will be written to the file cache. The file system cache policy determines when the data is actually written to the physical media (for example, least recently accessed). In addition, the system call `sync()` may be used to flush the file cache to the physical media.

NOTE: In an embedded environment, the operating system has a very short time (perhaps 2–3 ms) to perform a “proper” shutdown. This is not enough time to flush the file cache to the physical media. If the file cache attribute is set to write-back, then the user/owner of a read/write file system is responsible for flushing the file system cache. This can be accomplished directly by using the `sync()` system call or indirectly by using the `O_SYNC` attribute when opening the file.

This entry is not configurable for partition 0.

NumOfOpenFdsPerVmLim

The `NumOfOpenFdsPerVmLim` entry specifies the maximum number of file descriptors that can be in use at one time. This entry doesn’t specify just the number of concurrent open files. A file descriptor is used by many other resources in the system, such as message queues, shared memory, and pipes. As a general rule, a file descriptor will be used for each type of `open` call within an application. Unnamed pipes use two FDs per `open` because the call requires one for read and one for write access.

The value of `NumOfOpenFdsPerVmLim` should be between `3` and `32767`, inclusive. A minimum of `3` is required for `stdin`, `stderr`, and `stdout`.

NOTE: This entry is not configurable for partition 0.

NumOfMsgQueuesLim

The `NumOfMsgQueuesLim` entry limits the number of different named message queues a partition can open. Opening the same message queue name doesn't affect this total. All opens of message queues affect the usage of other resources. An open of a new message queue will consume a file descriptor, a shared memory object, and three semaphores tracked internally with `NumOfSemaphoresLim`. The value of `NumOfMsgQueuesLim` should be between `1` and `32767`, inclusive.

NOTE: This entry is not configurable for partition 0.

NumOfPipesLim

The `NumOfPipesLim` entry specifies the maximum number of pipes that can be opened. There are named and unnamed pipes. Pipes consume other resources as well. An open to a named pipe consumes one file descriptor while an open of an unnamed pipe consumes two file descriptors.

The value of `NumOfPipesLim` should be between `1` and `32767`, inclusive.

NOTE: This entry is not configurable for partition 0.

NumOfSharedMemObjsLim

The `NumOfSharedMemObjsLim` entry specifies the maximum number of shared memory objects available. Shared memory objects can be used directly through the `shm_open()` POSIX call. They are also used indirectly by POSIX message queues, so add one for each message queue created. Shared memory objects also use a file descriptor.

The value of `NumOfSharedMemObjsLim` should be between `1` and `32767`, inclusive.

NOTE: This entry is not configurable for partition 0.

NumOfSemaphoresLim

The `NumOfSemaphoresLim` entry specifies the maximum number of POSIX named and unnamed semaphores (POSIX calls `sem_open()` and `sem_init()`).

The value of `NumOfSemaphoresLim` should be between `1` and `32767`, inclusive.

NOTE: This entry is not configurable for partition 0.

NumOfUnixStreamSockLim

The `NumOfUnixStreamSockLim` entry specifies the maximum number of UNIX domain stream sockets for the partition.

The value of `NumOfUnixStreamSockLim` can be between 0 and 2^{31}, inclusive. However, sockets consume other resources as well. An open of a socket consumes one file descriptor.

NOTE: This entry is not configurable for partition 0.

Dynamic Device Driver Sub-Table

Type

The `Type` entry specifies the type of the dynamic device driver, character or block (used by `drinstall()` and `mknod()`). The value of `Type` should be either “`c`” or “`b`”. The value “`c`” indicates character and “`b`” indicates block.

DriverId

The `DriverId` entry is the driver ID for statically installed drivers (used by `devinstall()`). The value is required to install dynamic devices for drivers that were installed statically by the BSP. The value to use as `DriverId` can be determined by calling the `drivers` utility and taking the value from the `id` field from the first column of output for the specific driver associated with this device.

The value of `DriverId` should be an empty string (`DriverId=;`) if the driver is installed dynamically.

ObjectFname

The `ObjectFname` entry is the full directory path and filename of the driver object module (used by `drinstall()`). The path name should be the absolute path and filename or an empty string if installing only a device.

InfoFname

The `InfoFname` entry is the full directory path and filename of the device info driver's configuration file (used by `devinstall()`). The path name should be the absolute path and filename of the file.

NOTE: Refer to the read-only file system directory structure for the absolute path.

NumOfMinorDevs

The `NumOfMinorDevs` entry specifies the number of minor devices supported by this driver (used by `mknod()`). The value of `NumOfMinorDevs` should be from `0` to `256`, inclusive. A value of `0` or `1` will create one minor device. Refer to `BaseCharNodeFname` for details on the values.

BaseCharNodeFname

The `BaseCharNodeFname` entry specifies the absolute directory and base filename of the character device node or nodes, as appropriate. If the `NumOfMinorDevs` is equal to 0, then the `BaseCharNodeFname` will be directly used. If the `NumOfMinorDevs` is greater than 0, then a number will be appended to base name, starting with 0. The number will be incremented by one for each minor device. The `BaseCharNodeFname` should be the absolute path and base filename of the character device node or nodes.

The absolute path for all dynamic device nodes should be `/dev/ddev/`.

NOTE: In LynxOS-178, all device drivers must have a character device node, even block device drives.

NOTE: The LynxOS-178 naming convention for character device nodes is the block node name prepended with "r" (for example, `rd0` and `rrd0` for block and character device nodes, respectively).

BaseBlockNodeFname

The `BaseBlockNodeFname` entry specifies the absolute directory and base filename of the block device node or nodes, as appropriate. If the `NumOfMinorDevs` is equal to 0, then the `BaseBlockNodeFname` will be directly used. If the `NumOfMinorDevs` is greater than 0, then a number will be appended to the base name, starting with 0. The number will be incremented by one for each minor device. The `BaseBlockNodeFname` should be the absolute path and base filename of the block device node or nodes.

The absolute path for nonblock device nodes should be the empty string. The absolute path for all dynamic device nodes should be `/dev/ddev/`.

OwnerId

The `OwnerId` entry contains the `UserId` (uid) of the owner of the device node. The value of `OwnerId` should be from `0` to `NumOfVms-1`, inclusive.

GroupId

The `GroupId` entry contains the `GroupId` (gid) of the owner of the device node. The value of `GroupId` should be from `0` to `32765`, inclusive.

Permissions

The `Permissions` entry contains the UNIX file permission to be assigned to the device node file. The value should be an octal string that defines the file permissions.

Refresher: The UNIX file permission string is of the form “UGO,” where U is the user (owner), G is the group, and O is others (non-owner users). The bit values for each the octal characters are “rwx”, where r is read access, w is write access, and x is execute. For example, the permission string “760” gives the user read/write/execute permission, while the group has read/write permission and others have no access to the file.

File System Sub-Table

Mount

The `Mount` entry specifies the path name of a directory in the root file system where the file system will be mounted. The directory path should be an absolute path.

All configurable file systems should be mounted off the `/mnt/` directory branch (for example, `/mnt/vm1/` or `/mnt/vm2_data/`).

NodeFname

The `NodeFname` entry specifies the directory path and filename of the block device node that is used to access the file system. The `NodeFname` should be the absolute path and filename of the file system device node.

NumOfInodes

The `NumOfInodes` entry defines the maximum number of inodes allowed in the file system. One inode is required for every file and directory. The value of `NumOfInodes` should be between `2` and the maximum number of data blocks, or `Null`.

If the value is `Null` (empty), then one inode in every 16 data blocks is reserved for inodes.

MkffsArgs

The `MkffsArgs` entry defines command line parameters that are passed to the `mkffs` utility when the file system is created or formatted. The block device node and inode parameters are omitted, since they are already specified in the file system table. For a Flash file system, the format of the `MkffsArgs` command line string should be as follows:

```
[-F <%_spare_blocks>]
```

where the brackets (“[” and “]”) indicate optional parameters.

The <*%_spare_blocks*> entry should be an integer between `0` and `99` that specifies the percent of the file system reserved for part wear. For non-Flash file systems, there are no arguments.

The percentage is rounded up to the nearest block size. For example, given an 8 MB file system with a block size of 8 KB (1024 possible data blocks), if *<%_spare_blocks>* was set to 10 (10 percent), then 103 blocks would be reserved for part wear and 921 blocks would be available for data and inodes. The minimum number of blocks reserved for part wear is two, even if *<%_spare_blocks>* is 0.

NOTE: The verbose (`-v`) option is not supported in the Production Version of the `mkffs` utility. `cinit` expects to find the `mkffs` utility at `/usr/bin/mkffs`.

FfsckArgs

The `FfsckArgs` entry defines command line parameters that are passed to the `ffsck` utility when the file system is verified or checked. The block device node is omitted, since it is already specified in the file system table.

The format of the `FfsckArgs` command line string is as follows.

```
[-f] [-F] [-t | -r]
```

where the brackets (“[” and “]”) indicate optional parameters and bar (“|”) indicates either/or.

The `-F` parameter should be specified for Flash file systems. The `-r` parameter should be specified to remove corrupted files.

The `-t` parameter should be specified to truncate a corrupted file at the last valid data pointer.

The `-f` parameter should be specified to force a check of the file system. Note, if the “dirty” bit is not set on a file system, the `ffsck` utility will not check the file system unless this argument is used. The file system’s “dirty” bit indicates that the file system device was mounted read/write or was not successfully unmounted prior to shutdown.

NOTE: The purpose of the `ffsck` utility is to verify and repair a file system on a block device prior to mounting. Due to the fast shutdown design characteristic of LynxOS-178, a write to a RWFS may be interrupted during a shutdown. Only files that are written to during this time are at risk for potential corruption. This is to be considered normal operation for a RWFS.

The verbose (`-v`) option is not supported in the Production Version of `ffsck`. `cinit` expects to find the `ffsck` utility in `/usr/bin/ffsck`.

OwnerId

The `OwnerId` entry contains the `UserId` (uid) of the owner of the file system mount point. The value of `OwnerId` should be from `0` to `NumOfVms - 1`, inclusive.

GroupId

The `GroupId` entry contains the `GroupId` (gid) of the owner of the file system mount point. The value of `GroupId` should be from `0` to `32765`, inclusive.

Permissions

The `Permissions` entry contains the UNIX file permission to be assigned to the file system mount point. The value should be an octal string that defines the file permissions.

Refresher: The UNIX file permission string is of the form “UGO,” where U is the user (owner), G is the group, and O is others (non-owner users). The bit values for each the octal character are “rwx”, where r is read access, w is write access, and x is execute. For example, the string “760” gives the user read/write/execute permission, while the group has read/write permission and others have no access to the file.

VCT Target Filename and Location

The absolute path and filename of the VCT on the target should be `/usr/etc/VCT` and `/usr/local/etc/VCT`. Please note that for LynxOS-178, the same VCT should be placed in both locations mentioned above.

Example VCT File

The following is an example of a VCT with two partitions (sometimes referred to as VMs in the VCT), with comments explaining each field's permitted values. Users can modify the values shown here to match their target configuration.

LynxOS-178 comes with this sample VCT file in `/etc/vct.sample`. This VCT sets up two partitions running an interactive `bash` command shell. Partition 0 uses the `com1` serial port for input/output, and partition 1 uses `com2`. The sample VCT is shown below. It contains one dynamic driver section and two file system sections. These sections are provided as examples only and can be effectively disabled in the sample VCT by having the `NumOfDdds` and `NumOfFs` values set to 0.

```
////////////////////////////////////////
// Example VCT file
////////////////////////////////////////

// Cyclical Redundancy Check (CRC) value for the VCT.
// If this field is empty, then CRC checking of VCT is disabled.
VctCrc=;

// Version of the VCT. It allows the software to be backwards
// compatible when the format of the VCT changes.
// The value shall be a number between 0 and 65535.
VctVersion=2;

// The number of virtual machines in the VM table.
// The range is between 1 and 16, inclusive.
NumOfVms=2;

// The number of dynamic device drivers and devices in the DDD table.
// The range shall be between 0 and 48, inclusive. A maximum of 48
// device driver drivers, 16 block devices, and 32 character devices
// can be installed in this section.
NumOfDdds=1;

// The number of File Systems in the FS table.
// The range shall be between 0 and 16, inclusive.
NumOfFs=2;

// The action to perform on module fatal errors. The actions are
// either reset or halt after N errors from power up (cold and warm).
// Set to 0 if the system is to be reset on every occurrence of
// a module fatal error.
// Set to N, where N is greater than 0 and less than 255, if the
// system is to be halted after N module fatal errors.
ActionOnModuleErr=3;

// On power up, four possible start conditions can occur; factory,
// field, cold, and warm. The first three conditions result in the
// ColdStartSchedule being executed. The last condition results in
// the WarmStartSchedule being executed.

// The cold start schedule for the system. The value specifies the
// number of system ticks (e.g. 1 millisecond) each VM executes
```

```
// during a major (repeating) frame.
// For example, a value of "0[5] 1[30] 0[10] 1[5]" defines a major
// of 50 system ticks where VM0 runs for 5 ticks, followed by VM 1
// for 30 ticks, followed by VM0 for 10 ticks, followed by VM 2 for
// 5 ticks.
ColdStartSchedule=0[40] 1[10];

// The warm start schedule for the system.
// After the major frame is completed, the schedule is repeated until
// the WarmStartDuration is reached. At this point, the schedule
// switches to the RunTimeSchedule.
WarmStartSchedule=0[40] 1[10];

// The run-time schedule for the system.
RunTimeSchedule=0[40] 1[10];

// The length of time the cold start schedule executes before
// switching over to the run-time schedule.
ColdStartDuration=15000000;//15 seconds

// The length of time the warm start schedule executes before
// switching over to the run-time schedule.
WarmStartDuration=1000000;//1 second

////////////////////////////////////////
// Virtual Machines
////////////////////////////////////////
<VM0>
// The supplementary group membership, if any, of the virtual machine.
GroupIds=2 1 0;

// The name of the virtual machine. It is used to logically identify
// the virtual machine.
LogicalName=System;

// The CommandLine is used to start the master process of the VM.
// It consists of a sequence of words separated by space characters.
// All words are passed to the master program in the argv[] array.
// The argc parameter is calculated and set appropriately.
CommandLine=/bin/runshell /dev/com1 /bin/bash;

// Environment variables that are VM-specific.
EnvironmentVars=VAR1=var1 VAR2=var2;

// The total amount of system RAM allocated to this VM. This
// includes memory used by the VM's application(s) (text, data, stack,
// heap segments, etc.) and internal kernel resources required to
// support a VM.
SysRamMemLim=20971520;

// The directory path and file name of the Standard In character
// device node. During initialization, cinit opens this device node
// and re-directs Standard In to this device.
// The device may reference a static or dynamic device driver.
StdInNodeFname=/dev/com1;

// The directory path and file name of the Standard Out character
// device node. During initialization, cinit opens this device node
// and re-directs Standard Out to this device.
// The device may reference a static or dynamic device driver.
StdOutNodeFname=/dev/com1;

// The directory path and file name of the Standard Error character
```

```
// device node. During initialization, cinit opens this device node
// and re-directs Standard Error to this device.
// The device may reference a static or dynamic device driver.
StdErrNodeFname=/dev/com1;

// The path name of the VM's working directory (i.e. home directory).
WorkingDir=/;

// The path name of a directory in the root file system where the VM's
// RAM file system will be mounted. The directory path is the absolute
// path or an empty string if there is no RAM file system for this VM.
// For VMs with RAM file systems, the RamFsMount should be set to /tmp.
RamFsMount=/tmp;

// The size of the VM's RAM file system. The value of RamFsLim is in
// bytes ranging from 0 to SysRamMemLim. The RamFsLim shall be on
// a PPC page boundary (e.g. 4096 byte increments).
// If a RAM File system is not needed then the value shall be set to 0.
RamFsLim=3145728;

// The maximum number of Inodes allowed in the file system. One Inode
// is required for every file and directory.
RamFsNumOfInodes=200;

// The action to perform on VM fatal errors. The actions are either
// reset or halt after N errors from the current power cycle (cold
// and warm).
ActionOnVmErr=0;

// How the system software will handle a SIGILL exception when it
// occurs. This processor exception occurs when the processor
// executes an illegal instruction.
// The value of ActionOnSigillExcp is either "Default" or "Override".
// Default forces a VM fatal error and Override allows the VM to
// handle the exception.
// Should be set to "Default" for all VMs except for the first VM.
// Set to "Override" for the first VM (system partition, UserId=0).
ActionOnSigillExcp=Override;

// How the system software will handle a SIGFPE exception when it
// occurs. This processor exception occurs on floating point
// errors.
ActionOnSigfpeExcp=Override;

// How the system software will handle a SIGSEGV exception when it
// occurs. This processor exception occurs when Memory Management
// Unit (MMU) detects a segment violation.
ActionOnSigsegvExcp=POSIX;

// The core file size
CoreLimit=1048576;

// How the system software will handle a SIGBUS exception when it
// occurs. This processor exception occurs when an unauthorized
// access to memory occurs.
ActionOnSigbusExcp=POSIX;

// NB: The following per-VM resource limits are ignored for VM0.
// VM0 resource limits are specified in the uparam.h header file.

// The maximum number of processes on the system allocated to this VM.
// Any attempts by the VM to exceed this allocation will be rejected.
// The value of NumOfProcessesLim shall be between 2 and 64, inclusive.
```

```
NumOfProcessesLim=15;

// The maximum number of threads on the system allocated to this VM.
// Any attempts by the VM to exceed this allocation will be rejected.
// The value NumOfThreadsLim of shall be between 2 and 256, inclusive.
// This value must be equal to or greater than NumOfProcessesLim
// because a process always has a main thread.
NumOfThreadsLim=30;

// The maximum number of POSIX timers, available through
// timer_create(), on the system allocated to this VM. Any attempts
// by the VM to exceed this allocation will be rejected. The value
// of NumOfTimerLim shall be between 1 and 0x7fff, inclusive.
NumOfTimersLim=4;

// The size of the file system cache located in system Ram. One
// cache area is used for all file systems accessible by this VM.
// The sizing of the cache will directly affect the performance
// of the file systems since all read and write data goes through
// the cache.
FsCacheLim=4194304;

// The file cache attribute for the VM file system cache.
// The FsCacheAttr shall be set to WriteThrough or WriteBack.
// For write-through, writes to a file system will be written to
// cache and to the physical media during the write() system call.
// For write-back, writes to a file system will be written to the
// file cache.
FsCacheAttr=WriteThrough;

// TBD
FsNumOfInodesLim=150;
FsNumOfRecordLocksLim=60;

// The maximum number of file descriptors which can be in use at
// one time.
// The value of NumOfOpenFdsPerVmLim shall be between 3 and 32767,
// inclusive. A minimum of 3 is required for stdin, stderr, and
// stdout.
NumOfOpenFdsPerVmLim=150;

// The number of different named message queues a VM can open.
NumOfMsgQueuesLim=10;

// The maximum number of pipes which can be opened. There are named
// and unnamed pipes. Pipes consume other resources as well. An open
// to a named pipe consumes one file descriptor while an open of an
// unnamed pipe consumes two file descriptors.
NumOfPipesLim=80;

// TBD
NumOfSignalsLim=30;

// The maximum number of shared memory objects available. Shared
// memory objects can be used directly through the shm_open() POSIX
// call. They are also used indirectly by POSIX message queues so
// add one for each message queue created. Shared memory objects
// also use a file descriptor.
NumOfSharedMemObjsLim=10;

// The maximum number of POSIX named and unnamed semaphores (POSIX
// calls sem_open() and sem_init()).
NumOfSemaphoresLim=30;
```

```
// The maximum number of UNIX domain sockets per VM.

NumOfUnixStreamSockLim=5;

</VM0>

<VM1>
GroupIds=;
LogicalName=Virtual Machine 1;
CommandLine=/bin/runshell /dev/com2 /bin/bash;
EnvironmentVars=;
StdInNodeFname=/dev/com2;
StdOutNodeFname=/dev/com2;
StdErrNodeFname=/dev/com2;
WorkingDir=/;
ActionOnVmErr=3;
ActionOnSigillExcp=Default;
ActionOnSigfpeExcp=Default;
ActionOnSigsegvExcp=Default;
ActionOnSigbusExcp=Default;
RamFsLim=1048576;
RamFsNumOfInodes=200;
RamFsMount=/tmp;
PersStorOnLocalLim=0;
PersStorOnPmcLim=0;
RamHoldUpStorLim=0;
SysRamMemLim=20971520;
NumOfOpenFdsPerVmLim=150;
FsNumOfInodesLim=150;
FsCacheLim=4194304;
FsNumOfRecordLocksLim=60;
FsCacheAttr=WriteThrough;
NumOfMsgQueuesLim=10;
NumOfPipesLim=80;
NumOfSignalsLim=30;
NumOfSharedMemObjsLim=10;
NumOfSemaphoresLim=30;
NumOfTimersLim=4;
NumOfProcessesLim=10;
NumOfThreadsLim=30;
</VM1>

///////////////////////////////////////////
// Start of dynamic devices and drivers
///////////////////////////////////////////
<DDD0>

// The type of the dynamic device driver, character or block (used
// by drinstall() and mknod()). The value of Type shall be either:
// "c" or "b".
Type=c;

// The driver id for statically installed drivers (used by
// devinstall()). The value is required to install dynamic devices
// for drivers that were installed statically by the BSP.
DriverId=;

// The full directory path and file name of the driver object module
// (used by drinstall()). The path name shall be the absolute path
// and file name or an empty string if installing only a device.
ObjectFname=/bin/sample.dldd;

// The full directory path and file name of the device info (driver's
```

```
// configuration file, used by devinstall()). The path name shall be
// the absolute path and file name of the file.
InfoFname =/etc/sample.info;

// The number of minor devices supported by this driver (used by
// mknod()). The value of NumOfMinorDevs shall be from 0 to 255,
// inclusive. A value of 0 or 1 will create one minor device.
NumOfMinorDevs=0;

// The absolute directory and base filename of the character device
// node or nodes, as appropriate. If the NumOfMinorDevs is
// equal to 0 then the BaseCharNodeFname will be directly used. If
// the NumOfMinorDevs is greater than 0 then a number will be
// appended to base name starting with 0. The number will be
// incremented by one for each minor device. The BaseCharNodeFname
// shall be the absolute path and base filename of the character
// device node or nodes.
// The absolute path for all dynamic device nodes shall be /dev/ddev/.
// In LynxOS-178, all device drivers must have a character device node,
// even block device devices.
BaseCharNodeFname=;

// Same for the block device nodes.
BaseBlockNodeFname=/dev/ddev/sample;

// The UserId (Uid) of the owner of the device node.
// The value of OwnerId shall be from 0 to NumOfVms-1, inclusive.
OwnerId=0;

// The GroupId (Gid) of the owner of the device node.
// The value of GroupId shall be from 0 to 32765, inclusive.
GroupId=0;

// The Unix file permission to be assigned to the device node file.
Permissions=0666;

</DDD0>
///////////////////////////////////////////
// Start of filesystems
///////////////////////////////////////////
<FS0>

// The path name of a directory in the root file system where the
// file system will be mounted. The directory path shall be an
// absolute path.
// All configurable file systems shall be mounted off the /mnt
// directory branch.
Mount=/mnt/a;

// The directory path and file name of the block device node that
// is used to access the file system.

// The NodeFname shall be the absolute path and filename of the file
// system device node.
NodeFname=/dev/sdusb.0a;

// The maximum number of Inodes allowed in the file system. One
// Inode is required for every file and directory.
// If the value is Null then 1 inode in every 16 data blocks shall
// be reserved for inodes.
NumOfInodes=;

// Command line parameters that are passed to the ffsck utility
```

```
// when the file system is verified or checked.
// Warning: cinit expects to find the ffsck utility in /usr/bin.
FfsckArgs=;

// Command line parameters that are passed to the mkffs utility
// when the file system is created.
// If ffsck fails to find a filesystem on a device, mkffs utility
// will be called to create a new filesystem on that device.
// Warning: cinit expects to find the mkffs utility in /usr/bin.
MkffsArgs=;

// The UserId (Uid) of the owner of the File System mount point.
OwnerId=0;

// The GroupId (Gid) of the owner of the File System mount point.
GroupId=0;

// The Unix file permission to be assigned to the File System mount
// point. This value also describes the filesystem access mode.
// Filesystems with read-write access mode are only mounted by the VM
// identified by the OwnerId. Read-only filesystems are mounted by
// all VM's.
Permissions=0777;

</FS0>

<FS1>
Mount=/mnt/b;
NodeFname=/dev/sdusb.0b;
MkffsArgs=;
FfsckArgs=;
OwnerId=0;
GroupId=0;
Permissions=0777;
</FS1>
```

Calculated Limits (P0/Pn)

Table 7-1 describes limits derived from those listed above. The “Comments” column gives the formulas for these values.

Table 7-1: Calculated Limits (P0 / Pn)

Resource	Location	P0 Value	Limit	Comments
Max Fast Semaphores	uparam.h/cinit.c	512	180	(NumOfSemaphoresLim + NumOfMsgQueuesLim) * 3
Kernel Timers	uparam.h/fork.c	60	72	NumOfThreadsLim

Table 7-1: Calculated Limits (P0 / Pn) (Continued)

NameServer Name	uparam.h/cinit.c	200	86	NumOfSemaphoresLim + (2 * NumOfMsgQueuesLim) + NumOfSharedMemObjsLim
NameServer Open	uparam.h/cinit.c	300	129	NameServerName * 150%
NameServer Memory	uparam.h/cinit.c	25,600	11,008	NameServerName * Max string length (128)
File System Mounts	uparam.h/cinit.c	10	16	Total Num Of Block Devices (VM0) or File Systems (VM1-n)
File System Inodes	uparam.h/cinit.c	256	256	NumOfOpenFdsPerVmLim
Message Queue Signals	uparam.h/cinit.c	50	30	NumOfMsgQueuesLim * 3

Acronyms/Abbreviations

Table 7-2 explains the acronyms and abbreviations used in field names.

Table 7-2: Acronyms and Abbreviations

Acronym / Abbreviation	Definition
Args	Arguments
CRC	Cyclic Redundancy Check
DDD	Dynamic Device Driver
Dev	Device
Err	Error
Excp	Exception
Fd	File Descriptor
Fname	Filename
Fs	File system
Id	Identifier

Table 7-2: Acronyms and Abbreviations (Continued)

Lim	Limit
Mem	Memory
Msg	Message
Num	Number
Pers	Persistent
PMC	Peripheral Mezzanine Card
RO	Read only
RW	Read / write
SigBus	Signal bus error (LynxOS-178)
SigFPE	Signal floating point error (LynxOS-178)
SigIll	Signal illegal instruction (LynxOS-178)
SigSegv	Signal segmentation violation (LynxOS-178)
Sys	System
Stor	Storage
Vars	Variables
VCT	Virtual Machine Configuration Table
VM	Virtual machine

CHAPTER 8 LynxOS-178 Debugging Tools

LynxOS-178 provides a rich variety of tools to help users develop user-level applications and device drivers. Users can work with either the Luminosity IDE or with the GNU-based command line utilities. These cross-development tools run on a host system that is usually different from the target board on which the OS will run. This chapter describes specific features of the GNU debugger (GDB) ported and adapted to the LynxOS-178 Operating System, as well as the Simple Kernel DeBugger (SKDB) tool that is specially developed for debugging a kernel or device driver.

Lynx GDB debugger

The Lynx GDB Debugger provided in LynxOS-178 cross-development environment supports all main commands and features that are supported by GNU debugger, except for the following features:

- Debugging across `exec()` system call is not supported at this time.
- Reverse debugging is not supported at this time.
- Debugging of multi-threaded applications is provided in the “all stop” mode. In other words, when a thread stops due to a breakpoint or watchpoint, then all other threads are also stopped. This behavior differs from Linux with the “non-stop” mode support where only one thread is stopped.

The Lynx GDB Debugger provides a set of enhancements specially designed for the Lynx Operating Systems, such as:

- Extended Thread Information
- POSIX Awareness Feature
- Target System Information
- Kernel-Level Debugging Support

User-Level Debugging

The Cross-Development nature of the Lynx debugging process uses the following model of User-Level Debugging:

1. The `gdbserver` server is launched on the target.
2. The `gdb` client is launched on the Development Host.

3. The connection between the two is established using the `target remote` or `target extended-remote` commands. As a means of communication, either a network or a serial line can be used. Refer to the GNU document *Debugging with GDB* for details of command usage.

Information about Threads

The Lynx GNU Debugger (GDB) displays information about application threads printed by the info threads command. It shows process and thread IDs, thread name, address and function where each thread stopped.

For example:

```
* 3 process 9 thread 68 (print_thread1) 0x00011b04 in printf()
  2 process 9 thread 71 (print_thread2) 0x0001cd08 in _trap_()
  1 process 9 thread 67 (<noname>) 0x0001cd08 in _trap_ ()
```

If a thread was not given a name with the `st_name()` system call, then *`<noname>`* is printed for it.

POSIX Awareness

POSIX awareness support allows the Lynx GDB to display information about the POSIX synchronization objects (such as mutexes, condition variables, semaphores, and message queues) for the Lynx remote target via Machine Interface (MI) commands.

The following MI commands are supported:

- `-data-cv-info`
- `-data-mutex-info`
- `-data-sem-info`
- `-data-mq-info`
- `-data-queue-list`
- `-thread-list-blocked`

The -data-cv-info Command

Syntax	`-data-cv-info`
Synopsis	This command returns a list of the condition variables defined in the debugged process with their attributes.
GDB Command	No equivalent

Example:

```
(gdb)
-data-cv-info
^done,cv-
info=[{addr="0x10020748",cv={magic="0xace0cafe",wait={b_head="47",b_va
l="0"},mutex="0x100206f4",refcnt="1",clockid="2",lock={flags="0xcafe00
22",wait={b_head="0",b_val="0"},prio_c="0",count="1",referenced="0"}}}
,{addr="0x10020718",cv={magic="0xace0cafe",wait={b_head="0",b_val="1"}
,mutex="0x00000000",refcnt="0",clockid="2",lock={flags="0xcafe0022",wa
it={b_head="0",b_val="0"},prio_c="0",count="1",referenced="0"}}}]
(gdb)
```

The -data-mutex-info Command

Syntax	`-data-mutex-info`
Synopsis	This command returns a list of the mutexes defined in the debugged process with their attributes.
GDB Command	No equivalent

Example:

```
(gdb)
-data-mutex-info
^done,mutex-
info=[{addr="0x1001fe6c",mut={flags="0xcafe0022",wait={b_head="0",b_
val="0"},prio_c="0",count="0",referenced="0"}},{addr="0x1001fe54",mu
t={flags="0xcafe0022",wait={b_head="47",b_val="46"},prio_c="0",count
="1",referenced="0"}}]
(gdb)
```

The -data-sem-info Command

Syntax	`-data-sem-info`
Synopsis	This command returns a list of the POSIX semaphores defined in the debugged process with their attributes.
GDB Command	No equivalent

Example:

```
(gdb)
-data-sem-info
^done,sem-
info=[{addr="0x40003000",open_count="1",name="/sem_test_mult_semaphore
",sem={nsid="171639488",valid="0x0000736d",val="0",wait={b_head="0",b_
val="0"}}},{addr="0x1002028c",open_count="1",name="Unnamed",sem={nsid=
"0",valid="0x0000736d",val="2",wait={b_head="46",b_val="0"}}}]
(gdb)
```

The -data-mq-info Command

Syntax	`-data-mq-info`
Synopsis	This command returns a list of the POSIX message queues defined in the debugged process with their attributes.
GDB Command	No equivalent

Example:

```
(gdb)
-data-mq-info
^done,mq-info=[{addr="0x10025858",mq={mqh={mq_msg_list="-
1",mq_free_msgs="0",mq_shm_size="11488",mq_name="/tmp/mqtest0",mq_ns
id="171573952",mq_attrib={mq_flags="0x00000000",mq_maxmsg="2",mq_msg
size="4096",mq_curmsgs="0"}}}}]
(gdb)
```

The -data-queue-list Command

Syntax	`-data-queue-list addr`
Synopsis	This command returns a list of the messages in the message queue specified by the addr address in the debugged process address space.
GDB Command	No equivalent

Example:

```
(gdb)
-data-queue-list 0x10025858
^done,queue-list=[{addr="0x40007cc8",msg={cb={next="-1",prev="-
1",msg_prio="255",msg_size="4096"}, data="This is 1 message from
parent."}}]
(gdb)
```

The -thread-list-blocked-ids Command

Syntax	`-thread-list-blocked-ids addr`
Synopsis	This command returns a list of threads blocked on a sync object defined by the address addr in the debugged process address space. Additionally, for the message queues, this command returns the type of locked threads: receiving/transmitting.
GDB Command	No equivalent

Example:

```
(gdb)
-thread-list-blocked-ids 0x1001fe54
^done,thread-list-blocked-ids=[{pid="17",tid="47"}]
(gdb)
```

Target System Information

The Lynx GDB allows displaying the general parameters of the Target System, SysV, and POSIX IPC information for the Lynx remote target via Machine Interface (MI) commands.

The following MI commands are supported:

- `-data-system-info`
- `-data-sysv-ipc-info`
- `-data-posixipc-info`

The -data-system-info Command

Syntax	`-data-system-info`
Synopsis	This command returns the following information from the Target System: • Target system name and version. • Target file system limits such as maximum file length, maximum path length, maximum number of hard links per file. • Maximum number of processes and threads available on the target. • Current number of processes and threads existing on the target. • Memory usage information, such as size of physical and virtual memory available on the target and size of free physical and virtual memory.
GDB Command	No equivalent

Example:

```
(gdb)
-data-system-info
^done,system-
info=[osname="LynxOS",osrelease="7.0.0",osversion="20150226",osrevis
ion="Tue May 6 12:48:47 MSK
2015",max_file_len="4294967295",max_path_len="1024",max_name_len="25
5",max_hard_links="32767",max_open_files="128",max_procs="128",cur_p
rocs="8",max_threads="336",cur_threads="22",mem_phys_free="106172416
"]
(gdb)
```

The -data-sysvipc-info Command

Syntax	`-data-sysvipc-info`
Synopsis	This command returns information about all active System V-compatible interprocess communication objects.
GDB Command	No equivalent

Example:

```
(gdb)
-data-sysvipc-info
^done,sysvipc-info=[ ]
(gdb)
```

The -data-posixipc-info Command

Syntax	`-data-posixipc-info`
Synopsis	This command returns information about all active POSIX.1-compatible interprocess communication objects.
GDB Command	No equivalent

Example:

```
(gdb)
-data-posixipc-info
^done,posixipc-
info=[{name="/dev/mem",type="SHM",mode=0644,uid="0000",gid="0000",li
nks="1"}]
(gdb)
```

Kernel-Level Debugging

To perform Kernel-Level Debugging, the following model is used:

1. SKDB (Simple Kernel Debugger) runs in the kernel as a replacement for `gdbserver`.

2. The GDB client runs on the Development Host using a kernel binary `a.out` as a program to debug.
3. The connection between the GDB client and SKDB is established using a new `target skdb <port>` command where *<port>* is a serial port. Only a serial line is supported as a communication means between the host and target.

Starting a Debug Session

The following steps are required to create KDI with SKDB support and start a Kernel Debugging Session from the development host using `gdb`:

1. Enter the environment sourcing the `SETUP.bash` script.
2. Change to the BSP directory `sys/bsp.<name>` where *<name>* is the BSP name.
3. Enable SKDB by editing the kernel parameter file `uparam.h` and setting `INSTALL_SKDB` macro to 1:

   ```
   #define INSTALL_SKDB 1
   ```

4. Set the SKDB port to the serial line intended to be used for communication between GDB and SKDB:

   ```
   #define SKDB_PORT SKDB_COM<n>
   ```

   where *<n>* is a port number.

   Though it is possible to share the target serial port between regular terminal usage and Kernel Debugging, a dedicated serial port for Kernel Debugging is strongly recommended for reliable communication. If the target system has only one serial port which is used for console, release it before starting kernel debug session and use remote login for executing commands on the target.

   The target serial port must have matching parameters with the host's bit rate speed and parity bit. These parameters are usually configured with the target's `ttyinfo.c` file. To change the communication speed of the GDB session, use the `set remotebaud` GDB command.

5. Set `osstrip=none` and `resident=false` in the `$ENV_PREFIX/sys/bsp.<name>/lynxos-178.spec` file, where *<name>* is the BSP name.

6. Comment out the kernel binary stripping in the file `$ENV_PREFIX/sys/soc.<name>/Makefile.common` as:

```
# @$(ECHO)
# @$(ECHO) Stripping debug symbols...
# @$(ECHO)
# strip --strip-debug a.out
# @$(ECHO)
```

7. Create KDI using the following command:

```
$ make clean all kdi
```

The KDI named `lynxos-178.kdi` will then be created.

8. Boot the KDI onto the target and open the port that will be used for Kernel Debugging if it does not coincide with console port as:

```
$ cat /dev/com<n>&
```

where *<n>* is the port number.

9. Launch GDB on the host:

```
$ gdb a.out
```

10. Set the connection between GDB and SKDB in the GDB session:

```
gdb> target skdb <device>
```

where *<device>* is the device used on the host.

For example: `/dev/ttyS0`

NOTE: On the Windows Host, the serial ports are specified in the same way as on the Linux Host. For example, specify `/dev/ttyS0` for `COM1` port, or `/dev/ttyS1` for `COM2` port, etc.

Once connection between GDB and SKDB is established, the GDB interrupts the kernel and reports the location where the kernel stopped, usually in the null process.

Breakpoints can now be set at desired locations within the kernel and the `continue` command can be used to resume normal kernel operations.

When the kernel hits a breakpoint and stops, it is then possible to examine kernel variables as well as the call stack chain, and execute single-step, continue, and other related operations similar to the ones used for application debugging.

If the target is connected via serial port to a host different from the development host where Lynx cross-development environment is installed, the `socat` host utility can be used to make serial port available from the development host. For this, the following steps should be performed:

- On the remote host connected with the target, perform the following command:

  ```
  # socat tcp-l:<listen_tcp_socket>, reuseaddr,fork
  \ file:/dev/<serial_device>,nonblock,\
  waitlock=/var/run/lock/LCK..<serial_device>
  ```

  where *<listen_tcp_socket>* is an arbitrary free TCP/IP port, *<serial_device>* is a device on the host connected with the target serial port. For example:

  ```
  # socat tcp-l:4444, reuseaddr,fork \
  file:/dev/ttyS0,nonblock,\
  waitlock=/var/run/lock/LCK..ttyS0
  ```

- On the development host, perform the commands:

  ```
  # socat pty,link=/tmp/virt_serial \
  tcp:<host_ip>:<listen_tcp_socket>
  ```

  ```
  # ls -l /tmp/virt_serial
  ```

  where *<host_ip>* is an IP address of the remote host and *<listen_tcp_socket>* is the TCP/IP port specified on it. For example:

  ```
  # socat pty,link=/tmp/virt_serial tcp:192.168.4.41:4444
  ```

  The `ls` command will show the name of a dynamically assigned PTS device (e.g., `/dev/pts/46`) which should be used to connect to SKDB from the debugging session started on the development host.

NOTE: On the Windows Host, the `socat` utility cannot be used. So to start kernel debugging session, the target must be directly connected to the development host.

Working with Breakpoints

In a Kernel Debugging session, GDB appears virtually disconnected from any process. Though it still reports the current process ID, every thread running on the target is visible to GDB, unlike user process debugging where only the target process threads are visible. Therefore, a breakpoint can be hit by any thread of any process on the target, including kernel threads. The info threads command may display a long list of threads.

CAUTION! Since Kernel Debugging stops the entire Operating System, the Operating System will not respond when the kernel is at a breakpoint or is stopped by the debugger. In some circumstances, even if the system is in the running state, it may respond considerably slower or have no response at all because the debugger internally keeps single-stepping the kernel.

Kernel Debugging is a powerful tool. GDB allows the user to manipulate any and all memory contents in the target system, both in the kernel and user process spaces. However it is not advisable to set breakpoints in the user process space (a.k.a user program).

CAUTION! SKDB handles Kernel Debugging breakpoints in a completely different manner from that of user-mode breakpoints. A user-mode breakpoint set by user-level GDB will not be captured by GDB running in kernel debug mode, and may therefore cause an unexpected termination of the process.

CAUTION! It is not recommended to attempt to trace (setting breakpoints in and/or single-stepping) the core portions of the Lynx Kernel, such as those handling context switching and interrupt control. Any attempt may severely interfere with the LynxOS-178 Kernel operation. In addition, the instructions that manipulate the processor's status register may not be safely single-stepped. Although Lynx GDB detects, warns, and prevents such attempts, casual tracing of this critical code may result in an unexpected system freeze.

Interrupting the Kernel

To interrupt a Kernel Debug session, press `Ctrl-C` while the GDB is waiting for the target to stop.

Exiting the Debug Session

To finish a debug session and to let the target kernel resume freely, kill the target at the GDB prompt or quit GDB. Despite the command name, the target's kernel is not actually killed, but is resumed freely.

Example:

```
(gdb) kill
Kill the program being debugged? (y or n) y
(gdb)
```

CAUTION! If a kernel debugging session is terminated due to a communication error or for an unexpected reason, GDB may not be able to remove breakpoints before the termination. If this happens and a new kernel debug session is started, the kernel may be trapped at a breakpoint indefinitely or until the original instruction at the breakpoint location is restored manually using a command such as print or until the kernel is reloaded (restarted). GDB currently provides no other way to restore the original instructions. To reload the kernel and restart the system, use the R SKDB command (refer to "Raw SKDB Command" on page 88) or reset the target's hardware with the reset switch or a power cycle.

Debugging Dynamically Loaded Drivers

To debug a dynamically loaded driver, perform the following steps:

1. Create a KDI as described in the section entitled "Starting a Debug Session" on page 83 and boot it onto the target.
2. Compile the driver with the -g option to enable the debugger information in the binary and place it on the target.
3. Install the driver. For example:

   ```
   [/tmp] $ drinstall -c sample.dldd
   sample.dldd 23
   [/tmp] $ devinstall -c -d 23 sample.info
   sample.info0 c 30 0
   [/tmp] $ mknod /tmp/ss c 30 0
   [/tmp] $
   ```

4. Issue the `drivers` command to verify the driver load address. The following is an example of output information that will be displayed:

```
23 char 1 537ab55c 55627 sample.dldd
```

where `537ab55c` is the hexadecimal driver load address.

5. Start the Kernel Debugging session on the host:

```
$ gdb a.out
(gdb) target skdb /dev/ttyS0
```

6. Add symbol information for the dynamically loaded device driver:

```
(gdb) add-symbol-file /home/user/sample/Debug/sample.dldd \
0x537ab55c

add symbol table from file
    "/home/user/sample/Debug/sample.dldd" at text_addr =
    0x537ab55c (y or n) y

Reading symbols from
    /home/user/sample/Debug/sample.dldd...done.

(gdb)
```

NOTE: The path to the driver binary on the host is entered in this command. On the Windows Host, the path should be entered as Cygwin-style path.

7. Add the driver source directory onto the host to the source search path to perform source-level debugging:

```
(gdb) dir /home/user/sample
```

On the Windows Host, the path should be entered as Cygwin-style path.

8. Set a breakpoint as for example:

```
(gdb) br sample.c:sampleopen
```

In the above command `sample.c` is a name of file which the device driver consists of and `sampleopen` is a function name inside the file.

Raw SKDB Commands

GDB is a generic debugger. It knows little about the Lynx kernel. To explore the Lynx kernel in greater details, raw SKDB commands can be issued from kernel GDB session using the skdb command, followed by the desired SKDB command string. Please refer to “Appendix C. LynxOS-178 SKDB Commands” on page 173 for the list of available commands.

For example, to print the first 5 processes running on the target, issue the following command:

```
(gdb) skdb p 5
* p 5
    pid ppid prio pgroup signals mask        sem VM S name
    0    0    0      0        0  ffffffff         0 C nullpr
    1    1   79      0        0  800000  5366d30c 0 W /init
    2    1   16      2        0  800000  53609bb4 0 W ./lwsrvr
    3    1   16      3        0  800000  536f4474 0 W /bin/bash
    4    3   16      4        0  800000  536f506c 0 W /bin/cat
    tid pid  prio stklen signals mask    sem     VM S name
(gdb)
```

Simple Kernel Debugger (SKDB)

The Simple Kernel Debugger (SKDB) is a machine-level symbolic debugger. This section provides an overview of SKDB, instructions on how to start it should a kernel crashes, and lists details of SKDB commands.

SKDB is designed to support debugging of Lynx kernel internals that primarily include device drivers. It allows you to perform the following operations interactively in the Lynx kernel space:

- Setting Breakpoints
- Examining Memory and Registers
- Changing Memory Contents
- Displaying Kernel Data Structures

In addition, SKDB can be used to determine the cause of a kernel crash or a kernel panic. Lynx GDB uses SKDB as the target agent for kernel/device driver debugging. Refer to “Appendix C. LynxOS-178 SKDB Commands” on page 173 for the list of all available SKDB commands and common rules for their usage.

Using SKDB

Using SKDB should be reserved for advanced users who have a firm understanding of Lynx Software Technologies concepts such as memory model, scheduling, interrupt handling, etc. SKDB is not designed for user process debugging. Instead, use GDB for user process debugging.

SKDB is a tool designed to debug new device drivers and similar components once the kernel has started and is running stable. It is not intended to be used for porting the Lynx Kernel to a new platform.

There may be times when SKDB may be useful for LynxOS-178 Kernel porting; however, it requires a very stable kernel and it is not effective during start-up until such time that the kernel internally installs and SKDB is initialized.

While SKDB is in operation (at the prompt), the entire Operating System is paused and no kernel services are available.

The SKDB Prompt

Whenever the Operating System is using SKDB, an asterisk (*) will appear as the prompt. The entire Operating System is then paused and no kernel services are available while the Operating System is in SKDB.

Starting SKDB Automatically after a Kernel Crash or Panic

Once installed, SKDB automatically starts when a kernel crashes or when a kernel has a memory access fault and/or panic situation. In this case, the kernel is usually unable to resume operation, but it is possible to determine the cause and the location of the kernel fault or the panic. The following commands may be useful for analyzing the cause:

- `p` process/thread table display
- `t` stack trace display
- `r` register contents display
- `m` memory contents display

Breaking into SKDB with Hot Key

Once SKDB has been installed, it can be started by pressing `Shift-Ctrl-Minus` (the “hot key”) on the keyboard of an SKDB-ready port while the Operating System is up and running. Some keyboards (typically video consoles) may use `Ctrl-Minus` instead.

NOTE: To break into SKDB with a hot key from a serial port, the port must be explicitly opened by a process. Having a login process or a dummy process (such as `cat`) running on the port will be sufficient.

To change the hot key combination for SKDB, perform the following steps:

1. At the SKDB prompt, enter the following command:

   * **z**

   SKDB will then prompt for a new key combination.

2. Press the desired key combination.

   SKDB will then prompt for the same key combination in order to confirm.

3. Press the same key combination again. To cancel the change, press any other key.

The hot key combination is set per port; therefore, different key combinations can be set for different ports.

The new hot key combination is not preserved across operating system reboots; it returns to the default combination after each reboot; whereby, the hot key combination will require to be performed again.

NOTE: To use SKDB for remote kernel debugging with GDB, do not change the hot key combination on the serial port. GDB has the hot key combination hard coded.

Kernel Status Display

At each invocation, SKBD prints a line like the following one:

```
DELOK:<pid>.<tid>@<trapcode>,<slevel>,<econtext>,<PSW>,<PC>
<checksum>
```

The following table describes the fields in the string above:

Table 8-1: Kernel Status Display Fields

Field	Description
`DEL`	The ASCII Delete character (usually invisible)
`OK`	The literal string
`pid`	Current process ID
`tid`	Current thread ID
`trapcode`	*`trapcode`* is the same as the architecture's exception code with the addition of -1 for invocation from keyboard and -2 for a panic situation
`slevel`	*`slevel`* is the kernel preemption level (0: user, 1: kernel, 2: no context switching, 3: no interrupts)
`econtext`	*`econtext`* is the per-thread register stack into which the kernel saves registers

Field	Description
PSW	Processor Status Word; called flags or status register in some architectures
PC	Last program counter
checksum	String checksum

As an example, breaking into SKDB with the “hot key” would display something similar to the following:

```
OK:0.0@-1,1,DB0AB19C,000199A0,00000207 C961
```

The above is interpreted that the operating system was running the `null` process code (`process 0, thread 0`) at address `0x207` with context switching enabled when the break-in occurred.

Kernel Status Redisplay

To redisplay the kernel status information, press `Ctrl-B` then `?` and `Enter`. This option is currently available on serial terminals only, not on video consoles.

Stack Trace Display

The `t` command displays a traceback or the “history” of nested function calls of a thread within the kernel. One can determine the “path” to the current breakpoint, panic location, or kernel fault location where the kernel entered SKDB. Tracing then stops as soon as the stack frame appears to be out of the valid kernel address range.

By default, the `t` command displays information about the current thread. However it can be used with an argument. If the argument equals to the process ID, information about the main thread of the specified process is printed. If the argument equals to the thread ID with a preceding `-` (minus sign) information about the specified thread is displayed.

Verbose Trace Mode

Turning on the verbose trace mode with the `v` command allows the `t` command to display the contents of each stack frame as well as the offset values from the frame pointer (or the stack pointer in the case of PowerPC).

Process, Thread, and Other Displays

The `p` command displays the contents of the kernel process table and thread table.

The `s` command with options displays the contents of a variety of kernel internal data structures.

Resuming the Kernel

To exit SKDB and resume the operating system, press the `Esc` key; the kernel will continue running until any of the following events occurs:

- a kernel breakpoint is hit
- kernel run is interrupted by a hot key
- kernel has crashed or got in panic

NOTE: It may be impossible to resume the kernel if it is in SKDB due to a kernel crash or a kernel panic.

Setting Breakpoints

SKDB can set up to 15 breakpoints in the kernel including device drivers. When the CPU reaches the instruction at a breakpoint, the control is trapped into SKDB. The breakpoints remain set until explicitly unset by the `u` command. SKDB may refuse to set a breakpoint on some instructions that are critical to its operation. Such instructions include those that work with the processor status word (PSW) register.

CAUTION! Do not set a breakpoint in the user process space from SKDB. Such a breakpoint will not be recognized by SKDB and will cause an unexpected termination of the user process.

Single-Stepping

Pressing `x` and the `Enter` key single-steps the current thread (the thread that was the initial cause of entering SKDB.) It is not possible to single-step the following:

- The thread that was "broken-in" with the hot key.
- Some machine instructions that are critical for SKDB operation (generally those that change the processor status word [PSW] register).
- A crashed kernel

CAUTION! It is not recommended to attempt to trace (setting breakpoints in and/or single-stepping) the "core" portions of the Lynx kernel, such as those that provide context switch or interrupt control, because such an attempt may severely interfere with the Lynx kernel operation. Also, the instructions that manipulate the processor status register cannot be safely single-stepped. Though SKDB detects, warns about, and prevents such an attempt, casual tracing of such critical code may result in an unexpected system freeze.

Disassembly

The `d` command disassembles 10 instructions from either the specified address or the current address. The current address is updated to the next text location after each disassembly. The current address is also updated to the breakpoint or the fault location whenever the kernel stops and re-enters SKDB.

NOTE: The current x86 version of SKDB uses the Intel-style syntax of disassembly, not the GNU (AT&T) style.

Setting Watchpoints

Some CPU architectures support hardware debug registers to implement watchpoints. The following LynxOS-178 platforms support SKDB watchpoints:

- x86 up to 4 watchpoints
- PowerPC up to 1 watchpoint

NOTE: The availability of the watchpoint operation for the PowerPC depends on the type of CPU.

NOTE: Watchpoints are not supported for ARM targets yet.

To set watchpoints, the following command is used:

`B` *<num> <addr>* `[`*<mode>*`] [!` *<ignore_addr>*`]`

where

<num> is a watchpoint number. It is a mandatory argument.

<addr> is a watchpoint location address. It is a mandatory argument.

<mode> is an access mode. It can be `r` for read access and `w` for write access. Default is `w`.

<ignore_addr> is an address to ignore. It should be preceded by the exclamation mark and space. The argument can be presented by a string of up to 10 symbols and means an address that should be ignored when the watchpoint is hit. This is useful to avoid stopping at known kernel locations where the watchpoint is accessed.

The `B` command can set as many watchpoints as the target CPU architecture allows.

For example, to catch all write accesses to `currtptr`, but not stopping at `resched+0x24` which is considered normal access, use the following command:

```
* B 1 currtptr w ! resched+0x24
```

CAUTION! "Ignore PC addresses" feature is implemented by software: all accesses to a watchpoint memory location cause CPU exception that is captured by SKDB. SKDB then examines the cause of the exception and the program counter value to determine whether to resume the kernel passively or stop and report the hit to the user. This may result in a significant speed penalty if the watchpoint is frequently accessed but ignored.

SKDB uses virtual addresses when setting the debug register. Depending upon the CPU (MMU) architecture, this may result in watchpoint misses if a real page is aliased (mapped to a different address location) and the alias addresses are accessed.

To remove a watchpoint, use the `U` command with the watchpoint number.

CHAPTER 9 Luminosity Open Communication Interface

The LOCI (Luminosity Open Communication Interface) toolkit is intended for user applications to work in tandem with Luminosity IDE on the host development systems to easily access services on a LynxOS-178 target. Services include running and debugging applications, running remote shell sessions, uploading and downloading files from the target, and obtaining information regarding processes, threads and other relevant events on the targets. Refer to *Luminosity User's Guide* for detailed description of Luminosity IDE functionality.

The LOCI package consists of the following components:

- LOCI Proxy
- LOCI Server
- LOCI Driver.

LOCI Proxy

The LOCI Proxy is a server that supports and controls all communications between Luminosity IDE and LynxOS-178 targets. It runs on the host and communicates with target systems via TCP/IP or a serial line. Connection between Luminosity IDE and the LOCI Proxy is always network-based.

One instance of the LOCI Proxy can communicate with one or more Luminosity IDE instances and one or more target systems.

The LOCI Proxy supports up to 2 serial line connections per target.

The current version of the LOCI Proxy supports the following features:

- LOCI Version

This feature allows Luminosity IDE to detect the current version of the LOCI Proxy. If the LOCI Server is running on a LynxOS-178 target, its version should match the LOCI Proxy version.

- Target Session Support

This feature allows Luminosity to create LOCI sessions with the LOCI Server on a LynxOS-178 target if there is network connection between the LOCI Proxy host and the target. The following sessions are available in the LynxOS-178 Development mode:

  - Remote Login

- Remote Filesystem
- Target System Viewer
- Remote Run
- Remote debug
- SpyKer trace collection.

The following sessions are available in the LynxOS-178 Production mode:

- Target System Viewer
- SpyKer trace collection.

- Serial Line Support

This feature allows Luminosity to create LOCI sessions with the LOCI Server on LynxOS-178 targets if the LOCI Proxy host is connected with the target via a serial line. If LynxOS-178 is running in the Development mode, all sessions that are available using network connection are available via serial line. The LynxOS-178 running in the Production mode does not support any sessions via serial line.

- SKDB Session Support

This feature allows Luminosity IDE to create SKDB (Simple Kernel Debugger) session with LynxOS-178 Kernel Debugger. The session does not require the LOCI Server on the target. It requires serial connection between the LOCI Proxy host and the target. The session is supported only if LynxOS-178 runs in the Development mode.

- Console Session Support

This feature allows Luminosity IDE to provide the System Console if the console is configured to a serial line. The session does not require the LOCI Server on the Target. It requires serial connection between the LOCI Proxy host and the target. The session is supported only if LynxOS-178 runs in the Development mode.

- SpyKer Trace Transfer Support

This feature allows Luminosity IDE to obtain a SpyKer trace collected on a LynxOS-178 Target running in Production Mode. The trace is transferred to the LOCI Proxy via a serial line connecting the Target with the LOCI Proxy host. The feature does not require the LOCI server to be running on the target. Refer to the “SpyKer Trace Tool” on page 105 for description of the SpyKer tool. This feature is not supported for LynxOS-178 running in the Development mode.

The LOCI Proxy host can be the same as a host with Luminosity IDE or a separate host.

The LOCI Proxy should be started before Luminosity IDE creates a LOCI Server-based session with the target. Luminosity IDE is able to start the LOCI Proxy automatically on the same host when access to a particular target is attempted for the first time. Refer to the *Luminosity User's Guide* for details of launching LOCI Proxy. Otherwise, the LOCI Proxy should be started manually or during the LOCI Proxy host booting procedure.

LOCI Server

The LOCI Server is a LOCI Target Agent that runs on the LynxOS-178 targets and implements services required by Luminosity IDE. Luminosity IDE does not communicate with the LOCI Server directly. Instead, it sends service requests to the LOCI proxy which passes them to the LOCI Server providing a communication channel between Luminosity IDE and the LOCI Server.

Only one LOCI Server can run in each VM of LynxOS-178 target system. Each LOCI Server can connect to different Proxy hosts simultaneously if the connection is performed via the network. If the connection is performed via a serial line, only one LOCI Proxy host can be connected by each LOCI server.

The LOCI server is located in the `$ENV_PREFIX/usr/loci/bin` directory of the Cross-Development environment together with the `loci_install` installation script which can be used to install the LOCI dynamic driver on the target. The LOCI server is named as `lwsrvr` in the development environment and `lwsrvr_pdn` in the production environment.

LOCI Driver

LynxOS-178 includes LOCI driver that can be installed on the Target statically or dynamically. It is used to provide such information that cannot be obtained at the user level. The dynamic driver is located in the `$ENV_PREFIX/sys/dldd` directory of the Cross-Development environment and named `loci.dldd`.

Running LOCI Proxy

The LOCI Proxy can be launched either automatically from the Luminosity IDE (refer to the *Luminosity User's Guide* for details), or manually. It must be launched manually if the LOCI Proxy is running on a host that is different from the Luminosity host.

One user can launch only one instance of the LOCI Proxy on a particular host. Different users can have different LOCI Proxy processes; however, one serial port cannot be shared between different instances. Different users should use different ports for communication with different LOCI Proxy instances running on the same host.

Launching LOCI Proxy on the Linux Host

To launch the LOCI Proxy on the Linux host, change to the `/opt/Lynx/LynxOS-178-<losver>/Luminosity/<lumver>/loci/lwproxy` directory where the LOCI Proxy is installed and run the following command:

```
$ ./lwproxy [-port <port_number>]
```

where `<losver>` is a LynxOS-178 version, `<lumver>` is a Luminosity version and *`<port_number>`* is a TCP/IP port number . Port number `8356` is used if the port option is not specified.

Launching LOCI Proxy on the Windows Host

To launch the LOCI Proxy on the Windows host, open the Command Prompt window, change to the `C:\Lynx\LynxOS-178-<losver>\Luminosity\<lumver>\loci\lwproxy` directory where the LOCI Proxy is installed and run the following command:

$ **lwproxy** [**-port** <port_number>]

The options have the same meaning as described above.

NOTE: To run the LOCI Proxy in the background, launch the lwproxy command from the Cygwin prompt as:

```
# ./lwproxy [-p <port_number>] &
```

NOTE: In case if LynxOS-178 is installed as a part of LynxSecure Bundle, lwproxy location is **`/opt/luminosity/<lumver>-0/loci/lwproxy/`**

Running LOCI Server

To run LOCI Server, the LOCI driver should be present in the target system. The driver can be statically linked into the kernel in which case there is no need to install it, or it can be installed automatically using `VCT` file on the target booting.. Otherwise it should be explicitly installed.

To install LOCI dynamic driver into the kernel, perform the following commands:

```
# cd /usr/loci/bin
# ./loci_install
```

The LOCI Server can be run as either a network or a serial line utility depending on the type of connection between the LOCI Proxy host and the target.

Running LOCI Server as a Network Utility

This mode requires the network components on the target initialized and properly set up. To start the LOCI Server in the TCP/IP mode, execute the following commands from the command prompt:

```
$ PATH=$PATH:/usr/loci/bin
$ lwsrvr -D
```

By default LOCI Server listens to the TCP connection at port `8355`.

Running LOCI Server as a Serial Line Utility

To start the LOCI Server in the serial mode, execute the following commands from the command prompt:

```
$ PATH=$PATH:/usr/loci/bin
$ lwsrvr -d <serial device> -D
```

where *`<serial device>`* is a name of the serial device available on the target, for example, `/dev/com2`. The default baud rate is `9600`.

The LOCI Server Command Line

The LOCI Server `lwsrvr` utility supports the following command line arguments:

`-D`

Runs `lwsrvr` in the background as a daemon.

`-p` *`<port_number>`*

Sets `lwsrvr` port as *`<port_number>`* to listen to the TCP connection. The default port is `8355`.

`-s` *`<baudrate>`*

Sets the serial baud rate. The default baud rate is `9600`.

`-M` *`<number>`*

Sets the maximum number of simultaneous sessions. The default number is `12`.

`-d` *`<device>`*

Sets the serial device to use for connection with the LOCI Proxy. Only one proxy can be connected.

NOTE: The serial device should not be used by the system, for example, as a console device.

`-l` *`<path>`*

Sets the path to the LOCI device (the LOCI node). The default path is inherited from the `LOCI_DRIVER_PATH` environment variable. If environment variable is not set, the default path is `/tmp/loci` which is the device node for the dynamic LOCI driver installed manually. If the dynamic LOCI driver is installed via VCT, the name is `/dev/ddev/loci0`. If the LOCI driver is statically linked to the kernel, the device name is `/dev/loci`.

`-L` *`<flag>`*

Toggles the delay between new connections accepting. The following values can be used to set the delay *`<flag>`*:

`0` - delay is off, `1` - delay is on

The default value is `0`.

`-u`

Configures `lwsrvr` to force suspending a new connection in case if the session limit is reached.

Creating KDI with LOCI

To create KDI with LOCI, the LOCI Server binary, LOCI driver and, optionally, the LOCI installation script should be present on the target.

Creating a Target Image with Statically Linked LOCI Driver

To create a target image with the statically linked LOCI driver, perform the following actions:

1. Enter the Cross-Development Environment.
2. Change to the `$ENV_PREFIX/sys/bsp.`*`<bsp_name>`* directory where *`<bsp_name>`* is the name of a BSP.
3. Add the following line to the end of the `config.tbl` file:

   ```
   I:loci.cfg
   ```
4. Change the `lynxos-178.spec` file adding the following line to the end of the file:

   ```
   #LOCI
   directory=/usr/loci/bin owner=0 group=0 \
   mode=dr-xr-xr-x \
   file=lwsrvr<mode> \
   source=$(ENV_PREFIX)/usr/loci/bin/lwsrvr<mode> \
   owner=0 group=0 mode=-r-xr-xr-x
   ```

   where *`<mode>`* is empty for the development mode and `_pdn` for the production mode.
5. If the debug session is required from Luminosity IDE, `gdbserver` is also should be added as:

   ```
   directory=bin file=gdbserver \
   source=$(ENV_PREFIX)/bin/gdbserver \
   owner=0 group=0 mode=-r-xr-xr-x
   ```
6. Rebuild the target image as:

   ```
   $ make clean all kdi
   ```

In the Development mode, after the resultant image is booted on the target, the LOCI server should be launched using the `-l` option, for example:

```
# PATH=$PATH:/usr/loci/bin
# lwsrvr -l /dev/loci –D
```

Creating a Target Image with Dynamically Linked LOCI Driver

To create a target image with the dynamically linked LOCI driver, perform the following actions:

1. Enter the Cross-Development Environment.
2. Change to `the $ENV_PREFIX/sys/bsp.<bsp_name>` directory where `<bsp_name>` is the name of a BSP.
3. Change the `lynxos-178.spec` file as follows:
    - Set the `osstrip` variable value as `none`.
    - Set the `strip` variable value as `false`.
    - Add the following lines to the end of the file:

    ```
    #LOCI
    directory=/usr/loci/bin owner=0 group=0 \
    mode=dr-xr-xr-x
    file=lwsrvr<mode> \
    source=$(ENV_PREFIX)/usr/loci/bin/lwsrvr<mode> \
    owner=0 group=0 mode=-r-xr-xr-x
    file=loci_install \
    source=$(ENV_PREFIX)/usr/loci/bin/loci_install \
    owner=0 group=0 mode=-r-xr-xr-x
    dfile=loci.info \
    source=$(ENV_PREFIX)/sys/dldd/loci.info \
    owner=0 group=0 mode=-r--r--r–
    file=loci_uninstall \
    source=$(ENV_PREFIX)/usr/loci/bin/loci_uninstall \
    owner=0 group=0 mode=-r-xr-xr-x
    dfile=loci.dldd \
    source=$(ENV_PREFIX)/sys/dldd/loci.dldd \
    owner=0 group=0 mode=-r-xr-xr-x
    ```

    where <*mode*> is empty for the development mode and `_pdn` for the production mode.
4. If the debug session is required from Luminosity IDE, `gdbserver` is also should be added as:

    ```
    directory=bin
    file=gdbserver source=$(ENV_PREFIX)/bin/gdbserver \
    owner=0 group=0 mode=-r-xr-xr-x
    ```

5. Rebuild the target image as:

    `$` **`make clean all kdi`**

In the Development mode, after the resultant image is booted on the target, the LOCI driver should be installed and the lwsrvr server launched, for example:

`#` **`cd /usr/loci/bin`**

```
# ./loci_install
```

```
$ PATH=$PATH:/usr/loci/bin
$ lwsrvr -D
```

NOTE: The dynamically linked driver can be installed using the `loci_install` script in the development mode only and in VM0 only. Use VCT to install the dynamic LOCI driver in the production mode or if the LOCI server should run in VM other than VM0 and the driver should be dynamic.

The LOCI dynamic driver can be installed using the VCT file. To perform this, update the VTC file after Step 2 listed above as follows:

- Add the following lines:

```
<DDDN>
Type=c;
DriverId=;
ObjectFname=/usr/loci/bin/loci.dldd;
InfoFname=/usr/loci/bin/loci.info;
NumOfMinorDevs=1;
BaseCharNodeFname=/dev/ddev/loci;
BaseBlockNodeFname=;
OwnerId=0;
GroupId=0;
Permissions=0777;
</DDDN>
```

where *N* is the first unused number. For example, if there are 3 `DDD` sections already, `N` should be equal to 3. Note that the numbering starts from 0.

- Set `NumOfDdds` parameter to `N+1` where *N* is the number previously used.

If the LOCI driver is installed via VCT, the `loci_install` command should not be issued because the driver will be installed on the target system booting. In the development mode, the `lwsrvr` daemon should be invoked in all required VMs as:

```
# PATH=$PATH:/usr/loci/bin
# lwsrvr -l /dev/ddev/loci0 -D
```

In the Production mode, since `bash` is not provided, use `CommandLine` parameter of the VCT file to invoke the `lwsrvr_pdn` server.

When a network connection is involved, each VM where `the` LOCI server is running should be accessible from the development host by a specific IP address. If a serial line is used for communication, the LOCI server must run in only one VM.

CHAPTER 10 The SpyKer Trace Tool

Overview

SpyKer belongs to a class of software development tools known as tracing tools. Tracing tools typically provide a system developer the ability to record, capture, and display operating system and application events on a running system. By capturing event trace data and displaying it with an easy-to-read graphical user interface (GUI), trace tools provide system developers with powerful system tuning and debugging capabilities.

Some tracing tools are based on in-circuit emulators and other hardware devices, but these tools are expensive and difficult to learn and use. SpyKer is a software-only trace tool. Unlike most software-only trace tools, SpyKer does not require manually inserting event capture code into the software being traced. It does not even require the software being traced to be recompiled and relinked. On the target system, SpyKer can be installed and executed on a running system without any prior preparation or configuration (except for required memory and disk space). SpyKer can even be installed and executed on production software in shipping products.

Event Trace Capturing

Taking advantage of LynxOS-178 dynamic device driver installation, SpyKer's event trace capture software installs itself into a running target system kernel. Once installed, SpyKer is prepared to take over certain critical function entry and exit points. During a trace, SpyKer intercepts the execution of these functions. Upon each interception, SpyKer records the event that occurred, the time it occurred, and additional data as needed. This additional data is known as an event's payload. For example, one entry point that is intercepted is the system call interface. The payload is the number of the system call that is invoked.

By default, SpyKer intercepts and captures trace data for a large number of critical functions, including system calls, interrupts, return from interrupts, context switches, and so forth. See Appendix D, "SpyKer Events and Commands" for a complete list of predefined event types.

Once the SpyKer driver is installed on the target system, it must be configured with the event types to capture. This configuration process can be done in a number of ways, but the simplest method is to use the SpyKer display tool on the development host system. This easy-to-use program provides a GUI to configure SpyKer event capture.

The simplest configuration of SpyKer is to capture all events. This approach, however, causes the greatest impact on the target system. This impact, called intrusion, represents the amount of system overhead that is incurred when capturing event trace data. There are two forms of intrusion: performance intrusion and space intrusion. Most trace tools do not provide any way to configure or tune intrusion to get the most faithful and useful event capture data. For example, most trace tools enable capturing all events or no events at all. When all events are being captured, system performance is impacted by all the data collection activity and the amount of memory required to store the trace data. This high level of intrusion may affect other system activities and may substantially alter performance characteristics and timing.

SpyKer, on the other hand, is designed to minimize intrusion in several ways. It allows the user to specifically set the amount of memory to use for captured event data. The size of this buffer determines the total amount of trace data that may be captured. Reducing the size of this buffer limits the amount of trace data that is collected, but it also reduces space intrusion.

SpyKer can also be configured to buffer captured event data to disk files or network interfaces. In this case, both performance and space intrusion are increased, but a virtually unlimited amount of event trace data can be captured.

Additionally, SpyKer can be configured to minimize performance intrusion by capturing event trace data for selected events only. The amount of intrusion for events not being captured is zero because SpyKer does not intercept those locations in the target system.

Minimizing intrusion is not something that is required every time SpyKer is used to capture event data. In many cases, the problem or tuning issue that is being pursued can be found when SpyKer is capturing all events. Since capturing all events maximizes intrusion, however, it is possible that the intrusion may mask or avoid the problem or tuning issue being pursued. For example, since system timing changes with the level of performance intrusion, any timing-related problems are affected and may be masked unless low levels of performance intrusion are selected. The overhead of a trace point is about 0.1microseconds on a 3.26 Hz Pentium system.

Once event data is captured and transmitted to the development host system, the SpyKer display tool can be used to display the information in a graphical form. The display tool allows the user to examine and measure captured events. Refer to the Luminosity User’s Guide details how to use the SpyKer display tool to configure event capture and inspect and evaluate trace data.

The SpyKer Tool Architecture

The SpyKer trace tool is designed to support high performance event tracing of the LynxOS-178 kernel and applications. SpyKer is composed of two parts:

- Development Host Display Tool
- Target System Data Capture Tools

The Development Host Display Tool is built-in into the Luminosity IDE. Please refer to *the Luminosity User's Guide* for instructions how to use it.

The Target System Data Capture Tool is provided with LynxOS-178 system. All binaries are located in the $ENV_PREFIX/usr/local/spyker/spykertarget directory. The Target System Data Capture Tool consists of two parts:

- A SpyKer driver. It can be linked in the LynxOS-178 kernel as either a static or dynamic driver. The driver performs actual event tracing. The SpyKer driver does not affect the kernel performance in any way until SpyKer daemon starts.
- A SpyKer event trace daemon. The daemon initializes the SpyKer driver and communicates with the SpyKer display tool on the development host.

NOTE: It is possible to perform event tracing without using the SpyKer display tool. This capability is described in *"Advanced Capture Features"* on page 114.

To perform an event capture, the SpyKer driver must be loaded and the SpyKer daemon must be started.

The SpyKer driver and the SpyKer daemon communicate using a device node which name is dependent on the way, the driver is linked. Before running stracerd, make sure that the user has the read-write permissions for the following SpyKer device nodes:

- /tmp/tr — if the SpyKer Device Driver is installed dynamically
- /dev/ddev/<name>0 where <name> is specified in the DLDD section of VCT — if the dynamic SpyKer Device Driver is installed using VCT Device Driver section
- /dev/spyker — if the SpyKer Device Driver is installed statically

The read-write permissions can be set by the owner of the device node by executing, for example, the chmod command:

```
$ chmod g+rw,o+rw <device_node>
```

Running SpyKer

To run SpyKer daemon, the SpyKer driver should be present in the target system. The driver can be statically linked into the kernel in which case there is no need to install it, or it can be installed automatically using VCT file on the target booting. Otherwise it should be explicitly installed.

To install SpyKer dynamic driver into the kernel, perform the following commands:

```
# cd /usr/local/spyker/spykertarget
# ./install_spyker
```

The SpyKer driver may be installed in this way only in VM0 virtual machine of LynxOS-178 target system.

In the Development mode, if trace collection is performed using Luminosity IDE, the driver installation and SpyKer daemon launching is performed automatically.

The peculiarities of using the SpyKer tool in the Production mode are described in details in “Running SpyKer in the Production Mode” on page 109.

The SpyKer Daemon Command Line

The syntax of the command to launch the SpyKer daemon `stracerd` is as follows:

`stracerd [options] <device>`

where *`<device>`* is `/tmp/tr` if SpyKer driver is dynamic and installed manually, `/dev/ddev/<name>0` where `<name>` is specified in the DLDD section of VCT if SpyKer driver is dynamic and installed via VCT file, and `/dev/spyker` if the SpyKer driver is statically linked in the kernel.

The following options are supported

`-c` *`<cmd_file>`*

Use commands from *`<cmd_file>`* file to collect a trace. Refer to “Trace Command File” on page 114 for details of *`<cmd_file>`* contents.

`-B` *`<baud_rate>`*

Set the speed as the *`<baud_rate>`* rate when transferring a collected trace via serial line. The option is used only if LynxOS-178 target system runs in Production mode and the trace is collected using **`Get trace`** operation in Luminosity IDE.

`-S <serial_device>`

Use the `<serial_device>` device to transfer a collected trace via serial line. The option is used only if LynxOS-178 target system runs in production mode and the trace is collected using **`Get trace`** operation in Luminosity IDE. Refer to "Running SpyKer in the Production Mode" on page 109 for details.

`-T <timeout>`

Set a timeout between two consecutive trace data packets sent by the daemon over the serial port. By default, the timeout is set to 1 second. The option is used only if LynxOS-178 target system runs in production mode and the trace is collected using **`Get trace`** operation in Luminosity IDE. Refer to "Running SpyKer in the Production Mode" on page 109 for details.

`-p`

Launch the SpyKer daemon in the pipe mode. This option is used by Luminosity IDE to transfer the trace to the LOCI server which will then pass it to Luminosity through the LOCI communication channel.

In the Development environment, the SpyKer daemon is named as `stracerd` while in the production environment its name is `stracerd_pdn`.

NOTE: The trace is collected in the `out.trc` file created in the directory where `stracerd` was launched. If the root file system is read-only, be sure that `stracerd` is launched from the directory which allows writing.

Running SpyKer in the Production Mode

SpyKer trace can be collected in the Production Mode also. The SpyKer daemon used in the Production Mode is named `stracerd_pdn` to distinguish it from the SpyKer daemon used in the Development Mode.

To collect a SpyKer trace in the Production Mode, the SpyKer driver should be installed either statically or using the VCT file.

There are two ways to collect a SpyKer trace in Production Mode:

- Using a Serial Line
- Using a Production version of the LOCI Server

Both ways can be used from Luminosity IDE only.

Trace Collection Using a Serial Line

To collect a SpyKer trace via a serial line, the VCT file should set an application that will launch both SpyKer daemon and the program to trace as a command to run in VM.

The SpyKer directory in the production cross-development environment `$ENV_PREFIX/usr/local/spyker/spykertarget` contains an example of such application - `run_spyker_pdn.c`. The application performs the following actions:

- Starts the SpyKer daemon in the script mode passing to it the sample command script `/usr/local/spyker/spykertarget/trace.cmd`. The script sets start and stop triggers and starts tracing. Also, the following option is passed to the daemon:

  ```
  -S /dev/rs232B_blocking
  ```

  The option forces the daemon to send a collected trace which is saved in the `/tmp/out.trc` via the specified serial port (`/dev/rs232B_blocking`). The trace can be obtained on a host connected to the port if LOCI is installed there and the LOCI Proxy is listening on the port. Refer to the *Luminosity User's Guide* for details on how to launch Lynx Proxy and receive the trace from Luminosity IDE.
- Starts a sample application `/bin/banner` to trace.

The user can change the `run_spyker_pdn` application to set another program to trace or use another serial device to deploy a trace as well as modify the `trace.cmd` script to set other SpyKer commands (refer to "SpyKer Commands" section on page 186 of this guide). To rebuild the `run_spyker_pdn.c` code if it was changed, use the following commands:

```
$ cd $ENV_PREFIX/usr/local/spyker/spykertarget
$ gcc -o run_spyker_pdn run_spyker_pdn.c
```

When selecting the serial device, please note that only the `rs232` serial driver is available in the Production mode. It supports two device nodes for each physical port in the system; blocking and non-blocking. Ports are numbered with capital letters: A, B, etc. It is preferable to use the blocking device node for the trace transfer.

For example: `/dev/rs232B_blocking`.

Usually VM0 outputs to the first port `/dev/rs232A` (this is controlled by the VCT file). Make sure to use a serial port, which is not occupied, otherwise the trace data may be corrupted or not sent at all.

Additionally, the -T <timeout> option can be added to the SpyKer daemon command line. It specifies a timeout between two consecutive trace data packets sent by the daemon over the serial port. By default, the timeout is set to 1 second. Increase this timeout if LOCI Proxy fails to collect all trace data sent, for example, the progress indicator in Luminosity stalls (refer to the *Luminosity User's Guide* for details).

The user can also specify the -B <baud_rate> option, where <baud_rate> is the serial port speed. If this option is not specified, 115200 is used.

Using a Production Version of the LOCI Server

The SpyKer daemon can be started automatically by a production version of the LOCI Server (lwsrvr_pdn) on the target upon a request from Luminosity IDE via a network. The production version of the LOCI server as well as an application to trace should be started using the CommandLine variable in the VCT file.

The SpyKer directory in the production cross-development environment $ENV_PREFIX/usr/local/spyker/spykertarget contains a sample application run_spyker_pdn_loci that can be used for this purpose. This routine achieves the following:

- Configures the network by starting the lcsd daemon and executing the ifconfig and route utilities.
- Launches the production version of the LOCI Server.
- Launches the /bin/banner application to trace.

The user can randomly change the run_spyker_pdn_loci application to set target and gateway IP addresses, LOCI server options, as well as run another program to trace. To rebuild the run_spyker_pdn_loci.c code if it was changed, use the following commands:

```
$ cd $ENV_PREFIX/usr/local/spyker/spykertarget
$ gcc -o run_spyker_pdn_loci run_spyker_pdn_loci.c
```

Refer to the *Luminosity User's Guide* for details on how to initiate a SpyKer trace collection.

NOTE: Collecting a trace using the LOCI server requires LOCI in KDI. Refer to “Creating KDI with LOCI” on page 101 for details.

NOTE: The SpyKer server should be included in the target file system under `stracerd` name (not `stracerd_pdn`). Otherwise Luminosity IDE will not be able to start it.

Creating KDI with SpyKer

To create KDI with SpyKer, the SpyKer Server binary, LOCI driver and, optionally, the SpyKer installation script and Production mode utilities should be present on the target.

Creating a Target Image with Statically Linked SpyKer Driver

To create a target image with the statically linked SpyKer driver, perform the following actions:

1. Enter the Cross-Development Environment.
2. Change to `the $ENV_PREFIX/sys/bsp.`*`<bsp_name>`* directory where *`<bsp_name>`* is the name of a BSP.
3. Add the following line to the end of the `config.tbl` file:

   `I:spyker.cfg`
4. Change the `lynxos-178.spec` file as follows:
   - Set the `osstrip` variable value as `none`.
   - Set the `strip` variable value as `false`
   - Add the following lines to the end of the file:

```
#SpyKer
directory=/usr/local/spyker/spykertarget \
owner=0 group=0 mode=drwxr-xr-x
file=stracerd \
source=$(ENV_PREFIX)/usr/local/spyker/spykertarget/stracerd<mode> \
owner=0 group=0 mode=-rwxrwxrwx
```

where *`<mode>`* is empty for the development mode and `_pdn` for the production mode.

5. If the trace will be collected using Luminosity IDE, LOCI support should be added also. Refer to “Creating KDI with LOCI” on page 101 for details.
6. Rebuild the target image as:

   `$` **`make clean all kdi`**

Creating a Target Image with Dynamically Linked SpyKer Driver

To create a target image with the dynamically linked LOCI driver, perform the following actions:

1. Enter the Cross-Development Environment.
2. Change to `the $ENV_PREFIX/sys/bsp.<bsp_name>` directory where *`<bsp_name>`* is the name of a BSP.
3. In the Production mode, increase the `VMZERO_NPROC` value in the `uparam.h` file. For example:

   ```
   #define VMZERO_NPROC 20
   ```

4. Change the `lynxos-178.spec` file as follows:
   - Set the `osstrip` variable value as `none`.
   - Set the `strip` variable value as `false`.
   - Add the following lines to the end of the file:

```
#SpyKer
directory=/usr/local/spyker/spykertarget \
owner=0 group=0 \
mode=dr-xr-xr-x
file=stracerd \
source=$(ENV_PREFIX)/usr/local/spyker/spykertarget/stracerd<mode> \
owner=0 group=0 mode=-r-xr-xr-x
file=install_spyker \
source=$(ENV_PREFIX)/usr/local/spyker/spykertarget/install_spyker \
owner=0 group=0 mode=-r-xr-xr-x
dfile=spyker.info \
source=$(ENV_PREFIX)/sys/dldd/spyker.info \
owner=0 group=0 mode=-r--r--r—
file=showtrace \
source=$(ENV_PREFIX)/usr/local/spyker/spykertarget/showtrace \
owner=0 group=0 mode=-r-xr-xr-x
dfile=spyker.dldd \
source=$(ENV_PREFIX)/sys/dldd/spyker.dldd \
owner=0 group=0 mode=-r-xr-xr-x
```

   where <*mode*> is empty for the development mode and `_pdn` for the production mode.

5. If the trace will be collected using Luminosity IDE, LOCI support should be added also. Refer to “Creating KDI with LOCI” on page 101 for details.
6. Rebuild the target image as:

   ```
   $ make clean all kdi
   ```

Advanced Features

Trace Command File

As a rule, users work in Luminosity IDE to configure and collect traces. There are circumstances, however, in which the target system is inaccessible to the host development system.

The SpyKer daemon can be executed and provided with a command file (a simple text file) to configure and start traces. The command file is a series of commands with the final :start command as follows:

```
<command>
<command >
<command>
...
<command>
:start
```

where `<command>` is a SpyKer command. Refer to “Appendix D. SpyKer Commands” on page 186 for complete list of available commands.

To start a trace with a command file `<command_file>`, execute the following commands:

```
# cd /tmp
# PATH=$PATH:/usr/local/spyker/spykertarget
# stracerd -c <command_file> <dev_node> &
```

where <dev_node> is `/tmp/tr` for dynamically installed driver, `/dev/spyker` for statically installed driver, or `/dev/ddev/<name>0` where `<name>` is specified in the DLDD section of VCT for SpyKer driver installed using VCT. If the `<command_file>` argument is specified as a dash (-), then the SpyKer daemon reads commands from its standard input. In this case, do not run the `stracerd` command in the background. For example:

```
# cd /tmp
# PATH=$PATH:/usr/local/spyker/spykertarget
# stracerd -c - <dev_node>
```

NOTE: The trace is collected in the `out.trc` file created in the directory where `stracerd` was launched. If the root file system is read-only, be sure that `stracerd` is launched from the directory which allows writing.

User Event Capture

SpyKer is designed to automatically intercept kernel functions to capture events. In addition to these events, SpyKer can be used to capture additional events referred to as user events by adding calls to the SpyKer `trace()` function (two underscores) in user's applications or in arbitrary places of the kernel sources.

To capture user events in the kernel, the SpyKer driver must be statically linked to the kernel.

To capture user events in application code, the `ustrace.o` file located in `$ENV_PREFIX/usr/local/spyker/spykertarget` directory of the Cross-Development environment must be linked to the application. To avoid changing makefiles, `ustrace.o` can be added to the standard C run-time library. The following commands add `ustrace.o` to the standard C run-time library:

```
# ar r $ENV_PREFIX/lib/libc.a ustrace.o
# ranlib $ENV_PREFIX/lib/libc.a
```

To capture user events, call the `trace()` function with the arguments shown below:

```
void __trace(int event_type_number, int short_payload,
char *long_payload, int long_payload_length);
```

where `event_type_number` is the number of event that should be generated if a program performs the `trace()` call, `short_payload` is a short payload that will be passed to the SpyKer display tool, `long_payload` is a pointer to the memory where the long payload is located, and `long_payload_length` is a long payload length in bytes. The SpyKer display tool interprets the long payload of user events to be a null terminated character string. To account for the length of the string and the null terminator, the following equation is recommended:

```
long_payload_length = strlen(long_payload)+1;
```

The maximum payload length is 252 bytes, including the null terminator. If there is no payload, the long payload length should be 0.

Creating Custom Events

SpyKer predefines event types 0 through 42, with event types 43 through 45 allocated to user events. The SpyKer driver and daemon support event types 0 through 511. Though the SpyKer display tool supports only event types 0 through 45 in its default mode, it can be configured to recognize and display event types greater than 45. Refer to the *Luminosity User's Guide* for specific information.

Intercepting Kernel Functions

In addition to user events, SpyKer can be extended to automatically intercept new kernel functions to capture new events. However, only statically linked functions may be intercepted. To add new interceptions and event captures, the custom_patches.c file located in the $ENV_PREFIX/sys/drivers/spyker directory needs to be edited and the SpyKer driver rebuilt. Before editing this file, a backup copy should be created.

The following code needs to be added to custom_patches.c to intercept new functions and add new events:

```
extern <type> <function> (<arguments...>);
static int <function>_blen = 0;
static <type> <function>_insb(<arguments...>)
{
    DUMMYINS
}
<type> proxy_<function>(<arguments...>)
{
    int short_paylod = <value>;
    char *long_payload = "The Long Payload";
    DRTOC();
    trace(<event_number>, short_payload, long_payload,
        strlen(long_payload) + 1);
    OSTOC();
    return <function>_insb(<arguments...>);
}
```

In the above code, *<function>* is the name of the function being intercepted and *<arguments>* and *<type>* must exactly match the arguments and the type of return value of the intercepted function. Refer to “User Event Capture” section on page 10 for a description of the arguments to trace().

The following code must be added to the function custom_patch() of the same file.

```
if (trace_ev_ok(s, <event_number>) )

    PATCH(<function>, proxy_<function>, <function>_insb,

        <function>_blen);
```

Finally, the following code needs to be added to the function `custom_unpatch()`:

```
UNPATCH(<function>, <function>_insb, <function>_blen );
```

After the file is updated, the SpyKer driver should be rebuilt by performing the following actions:

1. Enter the LynxOS-178 Cross-Development Environment.
2. Change to the `$ENV_PREFIX/sys/drivers/spyker` directory.
3. Perform the following command:

```
$ make clean install
```

4. Rebuild a target image as described in the section “Creating KDI with SpyKer” section on page 112.

Using showtrace Tool

The SpyKer directory provides a showtrace tool. It can be used to view text representation of a collected trace on the target. To use the tool, perform the following command on the target after the trace is collected:

```
# /usr/local/spyker/spykertarget/showtrace out.trc
```

Tuning SpyKer Driver Internal Structures

Memory Buffer Management

Upon capturing event trace data, SpyKer saves the data in kernel memory buffers managed by the SpyKer driver. SpyKer manages three separate buffers:

- Start buffer (disabled by default)
- Main buffer (always enabled)
- End buffer (disabled by default)

Main Buffer

By default, the Start and End buffers are disabled (size set to zero) and all event capture data is placed in the Main buffer. After data is placed in the Main buffer, one of three possible actions occurs:

• The SpyKer daemon empties the Main buffer after event tracing is finished and transmits the data over the network to the SpyKer display tool on the host development system. If the Main buffer is full and new event data is captured the buffer overrun occurs and the new event is lost. This is the default action.

• The SpyKer daemon empties the Main buffer while event tracing is occurring and writes the data to a file on a local file system in the target system. If event capture occurs so fast that the SpyKer daemon cannot keep up, overruns occur in the Main buffer. Buffering trace data to a local file system creates additional system overhead and intrusion during trace capturing. Captured trace data is written to the file `out.trc` on the target system in the directory where the SpyKer daemon was started. To prevent the target system's local file system from running out of space, set a maximum file size. This can be done either in the GUI (refer to the *Luminosity User's Guide* for details) or by a SpyKer command if no GUI is used. If a size is not specified, then the file size is not limited by SpyKer. If a maximum file size is entered, then no additional event captures occur after the maximum size is reached.

- The Main buffer is configured with the wrap option. New captured event data is allowed to overwrite the oldest event data in the buffer. When the trace stops, the SpyKer daemon sends the captured event data to the SpyKer display tool on the development host. Allowing the Main buffer to wrap substantially reduces system overhead and intrusion in comparison to saving buffer to a file since the daemon does not need to empty the buffer while event tracing is occurring. To enable wrapping in the Main buffer, use GUI (refer to the *Luminosity User's Guide* for details) or the SpyKer `wrap` command if no GUI is used. If the Start buffer size is zero, it is automatically set to 40 KB.

NOTE: Whenever the Main buffer is configured to wrap, the Start buffer should be enabled to capture SpyKer trace header information.

The size of the Main buffer is configurable on a per trace basis.

NOTE: The Main buffer size must be at least 64 kilobytes. The GUI will not allow a trace if the Main buffer is not sufficiently large.

The default Main buffer size is 2048 KB.

Start Buffer

When enabled, SpyKer uses the Start buffer to save captured event trace data at the start of the trace, including SpyKer trace header information. If a start trigger is enabled, event trace captures do not start until the trigger event occurs. In any case, SpyKer records process creation, thread creation, and program load events into the Start buffer. It is also used for the trace start event, existing thread and process events, limited resources usage event, start/runtime VM schedules event and the new wrap mode event.

The Start buffer is not intended for saving other events that may just occur near the start of a trace.

Captured event trace data in the Start buffer cannot be overrun and is preserved regardless of whether data in the Main buffer is intentionally wrapped or accidentally overrun.

End Buffer

When enabled, SpyKer uses the End buffer to save captured event trace data after the Stop Event trigger occurs. SpyKer places captured event trace data in the End buffer until the buffer is full.

Enabling the End buffer lets SpyKer capture event trace data both before and after the Stop Event trigger. The Main buffer contains the captured event trace data up to and including the Stop Event trigger, and the End buffer contains captured event data afterward. The End buffer cannot be overrun.

Changing the Initial Buffer Sizes

Use the following instructions to create an info file for the SpyKer Driver:

1. Edit the `spykerinfo.c` file in the `$ENV_PREFIX/sys/drivers/spyker directory`:

   ```
   $ cd $ENV_PREFIX/sys/drivers/spyker
   $ vi spykerinfo.c
   ```

2. Modify the values in the `spyker_info` structure to reflect appropriate configuration values. For example, the `spyker_info` structure can be defined as follows:

   ```
   #include "spyker.h"
   struct spyker_info spyker_info0 = {
   0, /* start buffer size */
   1000000, /* main buffer size in bytes */
   0, /* end buffer size */
   0, /* nvRAM buffer location */
   };
   ```

3. Compile the SpyKer driver using the following command:

   ```
   $ make clean install
   ```

After this, existing KDIs should be also rebuilt.

APPENDIX A LynxOS-178 Networking Components

This Appendix describes the following networking protocols available with LynxOS-178:

- Transmission Communication Protocol/Internet Protocol suite - TCP/IP
- Network File System - NFS

Transmission Communication Protocol/Internet Protocol - TCP/IP

LynxOS-178 supports the TCP/IP protocol. TCP/IP, however, can be configured, installed, or removed at any time. In the Production Environment, networking is supported by the LCS network stack. In the Development Environment, one of two network stacks from Lynx Software Technologies can be chosen from: LCS and a BSD-derived stack.

Enabling/Disabling TCP/IP Support

On the LynxOS-178 Development Environment, depending on the BSP, TCP/IP support may be enabled or disabled by default.

To enable the BSD-derived TCP/IP support, make sure that the lines for TCP/IP in the `config.tbl` file read as follows:

```
I:hbtcpip.cfg
I:dtsec.cfg
```

To disable TCP/IP support, make sure that the lines for TCP/IP in the `config.tbl` file read as follows:

```
#I:hbtcpip.cfg
#I:dtsec.cfg
```

Common TCP/IP Utilities

The LynxOS-178 TCP/IP stacks feature standard TCP/IP data transfer services. LynxOS-178 provides a Berkeley Software Distribution (BSD) socket interface system and networking library functions needed to access the underlying TCP/IP suite network protocols (TCP, IP, and UDP).

Table A-1 below lists the LynxOS-178 TCP/IP components available in the Development Environment:

Table A-1: LynxOS-178 TCP/IP Components

Component	Definition
`ping`	Sends a packet of data to a system to determine if it is on the network. The ping command is used to test that TCP/IP is installed correctly.
`hosts`	The hosts database. This can be the `/etc/hosts` file, the Network Information Service (NIS) hosts map, the Internet domain name server, or any combination of these.
`rlogin`	Allows the user to log on to a remote host on the network. This requires that the user name be the same on both the local and remote machine. If the `/etc/hosts.equiv` file is set up, a password is not needed when performing a remote log on. rlogin requires the host computer to have a UNIX-compatible operating system.
`/etc/hosts`	The file that contains the list of hosts on the network.
`/etc/resolv.conf`	`resolv.conf` is used to determine a system's domain name, domain search paths, and IP addresses for name servers and routers.
`telnet`	A protocol that allows for remote access to a system. The system can run any operating system, UNIX-compatible or not.
`.rhosts`	A file that provides the remote authentication database for the `rlogin`, `rsh`, and `rcp` commands.
`rsh`	Allows users to connect and execute commands on a remote host. This command expects that the `/etc/hosts.equiv` and `.rhosts` files are configured properly. The `rsh` command works only when users are considered equivalent on the local and remote machines.

Table A-1: LynxOS-178 TCP/IP Components

rcp	A command that copies files and directories between different hosts. The remote copy command or rcp is a fast and efficient way to exchange data quickly between UNIX-compatible hosts. The /etc/hosts.equiv and .rhosts files must be correctly configured to use this command. Note that rcp does not copy symbolic links.
ftp	A file transfer program that uses the File Transfer Protocol. The ftp program transfers files to and from a remote network site. Passwords are required to access user directories.
ifconfig	Allows users to view and configure TCP/IP network interface parameters. The ifconfig command is not hardware dependent.
netstat	A command that lets users know the status of the network. It displays the contents of various network-related data structures.

For more information on these components, see the appropriate man pages.

This chapter uses examples of a prompt that displays the user name and hostname. The trailing $ indicates that the account is using the bourne shell. The following example shows the prompt for user jones on the system shark:

```
jones@shark$
```

The next sections provide an overview of some of the most common TCP/IP utilities. These sections introduce basics of common TCP/IP utilities, such as the ping command, remote computer access (telnet, rlogin), and file transfer between computers (ftp).

Testing TCP/IP (ping)

The simplest way to test the TCP/IP configuration of the system is to use the ping utility. Users can send a test message to any host on the network with the ping command. Users can ping either the IP address or hostname of the machine. This test verifies the correct operation of hardware and TCP/IP software connecting the hosts. The ping command continues to send packets to the addressed host once every second until the command is terminated with a **CTRL+C**.

Figure A-1 shows the ping command testing TCP/IP configuration by sending data packets to the IP address of a system:

```
$ ping 192.168.1.102
PING 192.168.1.102: 56 data bytes
64 bytes from 192.168.1.102 icmp_seq=0.time=10. ms
64 bytes from 192.168.1.102 icmp_seq=1.time=0. ms
64 bytes from 192.168.1.102 icmp_seq=2.time=0. ms
64 bytes from 192.168.1.102 icmp_seq=3.time=0. ms
^C
----192.168.1.102 PING Statistics----
4 packets transmitted, 4 packets received, 0% packet loss round-trip (ms) min/avg/max =
0/2/10
$
```

Figure A-1: Testing TCP/IP with ping

Figure A-2 shows the `ping` command testing TCP/IP configuration by sending data packets to the hostname of a system.

```
$ ping shark
PING shark (192.168.1.101): 56 data bytes
64 bytes from 192.168.1.101: icmp_seq=0.time=10. ms
64 bytes from 192.168.1.101: icmp_seq=1.time=0. ms
64 bytes from 192.168.1.101: icmp_seq=2.time=0. ms
64 bytes from 192.168.1.101: icmp_seq=3.time=0. ms
^C
----shark PING Statistics----
4 packets transmitted, 4 packets received, 0% packet loss round-trip (ms) min/avg/max =
0/2/10
$
```

Figure A-2: Using ping to test TCP/IP

Once the correct host setup is verified, every host on the network can use the `ping` command on other members of the network as shown in Figure A-3.

```
$ ping orca
PING orca (192.168.1.102): 56 data bytes
64 bytes from 192.168.1.102: icmp_seq=0.time=10. ms
64 bytes from 192.168.1.102: icmp_seq=1.time=0. ms
64 bytes from 192.168.1.102: icmp_seq=2.time=0. ms
64 bytes from 192.168.1.102: icmp_seq=3.time=0. ms
^C
----orca PING Statistics----
4 packets transmitted, 4 packets received, 0% packet loss round-trip (ms) min/avg/max =
0/2/10
$
```

Figure A-3: Pinging Other Hosts on a Network

Troubleshooting ping

Problems related to testing TCP/IP configurations with `ping` sometimes occur due to host lookup failures or problems in connectivity between the systems, not necessarily with the TCP/IP configuration on the local system.

The /etc/hosts File

Host lookup failures with `ping` can sometimes be attributed to malformed `/etc/hosts` or `/etc/resolv.conf` files. For example, if the host named `fish` in the `/etc/hosts` file is not defined, the ping command fails as shown in Figure A-4.

```
$ ping fish
ping: fish: Host name lookup failure
$
```

Figure A-4: Host Not Defined in /etc/hosts

Because the hostname `fish` is not defined on the local system, `ping` returns a hostname lookup failure. The `/etc/hosts` file provides a means of mapping IP addresses to hostnames. However, in larger networks, a Domain Name Service (DNS) server is typically used. The DNS server maintains a database of hostnames and IP addresses. If a user pings a system that is not defined in `/etc/hosts`, the system then sends a request to a DNS server (defined in `/etc/resolv.conf` – see "The /etc/resolv.conf file" on page 80).

However, a preferred method to test TCP/IP functionality is to first ping the IP address of a system, then attempt to ping the hostname of a system. If the IP address of a host is found, but the hostname lookup fails, TCP/IP functionality is correctly configured, but hostname resolution needs to be corrected. To resolve the hostname lookup failure, the correct IP address and hostname must be added to `/etc/hosts`.

Figure A-5 shows an example `/etc/hosts` configuration file:

```
#loopback address
127.0.0.1 localhost

#localhost address 192.168.1.103
stingray

#other host addresses 192.168.1.101
shark
192.168.1.102 orca
```

Figure A-5: /etc/hosts example file

In this example, the localhost entry is called a "loopback" address and is used to point to the local system. An entry also exists for the IP address and hostname of the local system. Any other IP address and hostname definitions point to other systems on the network.

The /etc/resolv.conf file

In addition to the `/etc/hosts` file, the `/etc/resolv.conf` file provides the domain name for the local system, domain search paths used when looking up hosts, and IP addresses for DNS servers.

Users can ping a host without providing a domain name by entering the following command:

```
# ping fish
```

If fish is not defined in `/etc/hosts`, the local system uses the search paths and DNS servers provided in `/etc/resolv.conf` to determine a fully-qualified domain name and IP address. The system uses the search entries in the `resolv.conf` file to determine a fully qualified domain name (for example, `fish.domain1.com`). For the system to find the IP address of a host, it must have access to one or more DNS servers. These DNS servers contain indexes of fully qualified domain names and valid IP addresses.

The structure of the `/etc/resolv.conf` file is as follows:

Figure A-6: /etc/resolv.conf example file

```
domain domain1.com

search domain1.com search domain2.com

nameserver 192.168.1.254
nameserver 192.168.1.253
```

In this example, the domain definition provides the domain of the local system. If the hostname is stingray, for example, the fully qualified hostname would be `stingray.domain1.com`.

The `search` definitions provide a means to resolve the fully-qualified domain name for hosts. For example, a system searching for the host fish first attempts to resolve the fully qualified domain name to the first search entry, or `fish.domain1.com` by sending a request to the local DNS server. If no entry exists in the DNS table for the system `fish.domain1.com`, the local system resolves the fully qualified domain name to the second search entry in resolv.conf, or fish.domain2.com. A second request is sent to the DNS server for the IP address of fish.domain2.com. This process continues until a valid host and domain is found or there are no more search paths. There is no limit to the number of search paths that can be used in /etc/resolv.conf.

nameserver definitions in /etc/resolv.conf are IP addresses pointing to local DNS servers. These DNS servers are used to translate the fully qualified domain name to a valid IP address.

ping Not Responding

If the ping command fails to respond with any output, terminate the program by pressing **CTRL+C**. Common reasons for ping failure include:

- The machine orca is not connected to the network.
- The machine orca is down or powered off.
- The /etc/hosts file on shark has the wrong Internet address for orca.
- The TCP/IP configuration for orca is broken.

Logging On to a Remote Computer (telnet, rlogin)

Users can log on to another host on the network using either one of the two utilities supplied with TCP/IP:

- telnet
- rlogin

telnet

The telnet utility allows users to log on any type of computer that supports TCP/IP. The computer can run any operating system, UNIX-compatible or not. This utility allows a LynxOS-178 user to access a system from anywhere on the network.

`telnet` is invoked with the hostname of a remote computer, as shown in Figure A-7.

```
jones@orca$ telnet shark
Trying...
Connected to shark. Escape character is
'^]'.

LynxOS-178 (shark) user name:

jones
password:

jones@shark$
```

Figure A-7: Using telnet to Remotely Log in

To access the system, users must supply a user name and password.

Terminate a `telnet` session by logging out of the system with the `exit` command, as shown in Figure A-8.

```
jones@shark$ exit
Connection closed by foreign host.

jones@orca$
```

Figure A-8: Terminating a telnet Session

rlogin

Users can use `rlogin` to remotely log on another computer similar to `telnet`. Unlike `telnet`, `rlogin` requires the host computer to have a UNIX-compatible operating system.

`rlogin` is invoked with the hostname of the remote computer, as shown in the figure below:

```
jones@orca$ rlogin shark Login shark
vt100 password:

jones@shark$
```

Figure A-9: Using rlogin

In the previous example, no user name is passed to `rlogin`. In this case, the user name on the local machine is used to log on the remote machine, for example, user `jones` on host `shark` logged on remote host `orca` as user `jones`.

If the user wants to log on to a system that does not have an identical user account, the login argument, `-l` followed by the desired user account must be added, as shown in Figure A-10.

```
jones@orca$ rlogin shark -l doc
Login shark vt100 password:

jones@shark$
```

Figure A-10: Remote Log in as Another User

Unlike `telnet`, the `rlogin` utility lets users take advantage of the information in `/etc/hosts.equiv` and `.rhosts` files. Users on machines that are set up to be considered local are not prompted for a password.

Executing Commands Remotely (rsh)

Another utility included with TCP/IP is `rsh`, the remote shell command. This utility allows users to perform the following tasks:

- Access remote hosts and redirect output to the local machine.
- Execute a command on that host.

Accessing Remote Hosts and Redirecting Output to the Local Machine

The `rsh` command only works when the user is considered equivalent on the local and remote machine.

The syntax for the `rsh` command is as follows:

```
rsh host [-l user_name] command
```

An optional user name can be given to execute the command as a specific user. This is useful if the current user account is not considered equivalent on the remote machine. `rsh` redirects standard input, standard output, and standard error from the remote machine to the local host.

The `who` utility displays users who are currently logged into a remote host, as shown in Figure A-11:

```
davis@shark$ rsh orca who
rootatc0Mon Dec 23 10:40:54
roottttyp0:0.0Mon Dec 23 10:41:19
roottttyp1:0.0Mon Dec 23 10:49:02
roottttyp2:0.0Mon Dec 23 11:05:45 davis@shark$
```

Figure A-11: Using who to Query Log ins on a Remote System

Executing a Command on a Remote Host

Also, commands can be invoked on a remote host as another user. In Figure A-12, user `jones` on host `orca` invokes the `whoami` command. This utility reports the current `login` account.

```
davis@shark$ rsh orca -l jones whoami
jones

davis@shark$
```

Figure A-12: Using rsh to Remotely Execute a Utility

Transferring Files Between Machines (ftp, rcp)

File transfer using TCP/IP is fast and efficient. Like the remote login utilities previously discussed, there are two ways to copy files between hosts:

- `ftp` for hosts of differing operating systems.
- `rcp` for UNIX compatible operating systems.

File Transfer Protocol - (ftp)

The File Transfer Protocol or `ftp` allows users numerous configuration options. In addition to the options provided in this document, review the `ftp man` page for setting advanced options.

Starting ftp

In its simplest invocation, `ftp` is called with the hostname of the remote machine. `ftp` prompts for a user login and password. A password must be provided to access user accounts.

In Figure A-13 user `davis` on host `orca` connects to host `shark` and logs on as user `jones`.

```
davis@shark$ ftp orca
Connected to orca.
220 orca FTP server (Version 4.162 Tue Nov 1 10:50:37 PST 1988) ready. Name (orca:davis):
jones
331 Password required for jones. Password:
230 User jones logged in. ftp>
```

Figure A-13: Connecting to a System with ftp

Retrieving Files from a Remote Host (get)

Once logged in, files can be retrieved from the remote host using the `get` command as shown in Figure A-14:

```
ftp> get .login
200 PORT command successful.
150 Opening ASCII mode data connection for .Login (209 bytes).
226 Transfer complete.
218 bytes received in 0.01 seconds (21.29KB/s) ftp>
```

Figure A-14: Downloading Files with ftp

Sending Files to a Remote Host (put)

Alternatively, files can be sent to the remote host using the `put` command as shown in Figure A-15.

```
ftp> put hosts.equiv
200 PORT command successful.
150 Opening ASCII mode data connection for hosts.equiv.
226 Transfer complete.
12 bytes send in 0.07 seconds (0.17KB/s) ftp>
```

Figure A-15: Uploading Files with ftp

NOTE: By default, the `ftp` transfer program operates in an ASCII text mode. Set the transfer mode to binary by entering binary at the `ftp` prompt.

Transferring Binary Files

To transfer binary files, the transfer mode must be changed by entering the binary command at the `ftp` prompt. For example:

```
ftp> binary
200 Type set to I.
ftp>
```

When transferring a binary file in ASCII mode instead of in binary mode, the file retains the same number of bytes, however, the byte order becomes corrupted. To preserve the integrity of a binary file, make sure that `ftp` is set to binary mode. A drawback of this transfer mode is that binary file transfers are more time-consuming than ASCII file transfers.

Listing ftp Commands

`ftp` commands are displayed by entering a question mark (`?`) at the `ftp` prompt. See the `ftp man` page for a complete list of commands.

Remote Copy — (rcp)

The remote copy command or `rcp` is a fast and efficient way to exchange data quickly between UNIX-compatible hosts. To be able to access files using `rcp`, users must have already set up the `/etc/hosts.equiv` and `.rhosts` files correctly. Files can be copied between hosts using syntax similar to the UNIX cp command.

The only difference is that the remote host's filename must be indicated properly. In the following example, file `/etc/hosts` is copied from host `orca` to host `shark`:

```
jones@shark$ rcp orca:/etc/hosts /tmp
```

Like the `cp` command, multiple files can be transferred to a directory:

```
jones@shark$ rcp /etc/passwd /etc/printcap orca:/tmp
```

Finally, the `rcp` command can be used to transfer files between two hosts that are different than the host currently logged into (assuming proper configuration). In the following example, the `/etc/passwd` file is copied from host `orca` to host `fish` from a user on host `shark`:

```
jones@shark$ rcp orca:/etc/passwd fish:/tmp/passwd
```

Driver Defaults

Table A-1 shows the default values within the driver information files. These files are located in `/sys/devices`.
To change the defaults, edit the file, compile it, and install TCP/IP support again. For more information, see “Enabling/Disabling TCP/IP Support” on page 75.

Table A-1: Default Values within Driver Information Files

Info File	Target	Defaults	Notes

`hbtcpip_info.c`	`all`	`Mbufs (multiple of 4): 512 clusters=(1/4 of mbufs) 128`
		`tcp_sendspace=16384`
		`tcp_receivespace=16384`
		`udp_sendspace=16384`
		`udp_receivespace=41984`
		`cluster_size=1024`
		`clshift_bits=10`
		`ipforwarding=1`
		`tcprexmtthresh=3`
		`tcp_mssdflt=512`
		`tcp_keepintvl=150`
		`tcp_do_rfc1323=0`
		`tcpip_max_prio=255`
			This file contains information about memory usage and other control args for the BSD-derived TCP/IP stack. Please see the hbtcpip0(4) man page for detailed information on configuration options.

Network File System—NFS

During initial installation of LynxOS-178, NFS client support is disabled by default. NFS, however, can also be configured, installed, or removed at any time after initial installation. Note that NFS requires TCP/IP to function. This section describes NFS basics, as well as advanced NFS configuration options.

Overview

NFS is a suite of user programs and kernel functionality that allow access to a remote host’s file systems as if they were local. All or part of a remote host’s file system is mounted into the local host’s file system, allowing transparent access to remote files by local users. Once mounted, any file on the remote file system is accessible. Such files can be operated on by most utilities, functioning no differently than a file located on the local disk.

When attempting to access a file in an NFS-mounted directory, the NFS client sends a request to the NFS server on the remote system. The NFS server accepts and manages these requests from the remote NFS client for access to the local disk. The server enforces permissions and performs the actual manipulations to the local disk.

Enabling/Disabling NFS Support

On the LynxOS-178 Development Environment, NFS support is disabled by default.

To enable NFS support, make sure that the lines for NFS in the config.tbl file read as follows:

```
#I:nullnfs.cfg
I:nfs.cfg
```

To disable NFS support, make sure that the lines for NFS in the `config.tbl` file read as follows:

```
I:nullnfs.cfg
#I:nfs.cfg
```

Tuning the NFS Client Kernel

The NFS client is tuned by changing the values of six kernel parameters. The structure `unfs_info`, found in `/sys/devices/nfsinfo.c` contains six tunable parameters:

- The maximum number of NFS file nodes that can be open at any time.

  The default is 64. The value of this parameter should be increased for heavy NFS traffic.

- The maximum number of NFS directories that can be mounted.

  The default is 8. If this value is increased, make sure that `NMOUNTS` in `/sys/bsp.<bsp_name>/uparam.h` is also increased to an equal or greater value.

- The maximum number of NFS client daemons that can be started at any time.

  The default is 32. In case of heavy NFS client traffic, multiple client daemons should be started. This can be done by duplicating the `/net/unfsio` line in `/net/rc.network`.

- The maximum number of NFS client requests that can be in the queue at any time.

The default is 32. The value of this parameter should be increased for heavy NFS traffic.

- The maximum number of bytes in an NFS read/write request.

  The default is 8192. This value should be reduced to 4096 or less to interface with systems that have slower (that is, 8-bit) Ethernet boards.

Edit the `/sys/devices/nfsinfo.c` file to change any of these parameters. Be sure to change only the values. After making the desired changes, the device library must be updated and the kernel rebuilt.

APPENDIX B LynxOS-178 Production and Development Mode APIs

The following table describes the LynxOS-178 API functions available from user space in both Production Mode and Development Mode. The APIs available to applications built in Development Mode are a superset of those available in Production Mode.

Column Descriptions

The “API name” column indexes the table’s items in alphanumeric order by name. Names are listed in case insensitive order for ease of searching through the table by the human reader.

The “Location” column describes whether the named item is in a library (giving the library’s name), a system call, or a macro. In cases where there are two ways to resolve the symbol, both are indicated in the column. System calls directly call the corresponding kernel functions using a hardware-specific mechanism. Library functions are functions implemented within the libraries.

The “DEV mode only?” column indicates if the API is only available in Development mode (Yes), or is also available in Production Mode (No). APIs marked “Yes” cannot be used in DO-178 certified applications.

The “Comment” column indicates general information, such as whether an API is obsolete, recommended alternatives to an API, header files containing a macro, whether an API is Lynx proprietary, and other useful information.

API name	Location	DEV mode only?	Comment
`_Exit`	`libc`	No	
`_exit`	`libc` / System Call	No	

API name	Location	DEV mode only?	Comment
_tolower	Macro	No	Obsolete. Defined in ctype.h.
_toupper	Macro	No	Obsolete. Defined in ctype.h.
a64l	libc	Yes	
abort	libc	No	
abs	libc	No	
accept	libc / System Call	No	
access	libc / System Call	No	
acos	libm	No	
acosh	libm	No	
adjtime	libc / System Call	No	
alarm	libc	No	
alphasort	libc	Yes	
asctime	libc	No	Obsolete.
asctime_r	libc	No	Obsolete. Use strtfime() instead.
asin	libm	No	
asinh	libm	No	
assert	Macro	No	Defined in assert.h.

API name	Location	DEV mode only?	Comment
atan	libm	No	
atan2	libm	No	
atanh	libm	No	
atexit	libc	No	
atof	libc	No	It is recommended that strtod() be used instead.
atoi	libc	No	It is recommended that strtol() be used instead.
atol	libc	No	It is recommended that strtol() be used instead.
atoll	libc	Yes	
basename	libc	Yes	
bdv_install	libc / System Call	No	Lynx proprietary.
bdv_uninstall	libc / System Call	Yes	Lynx proprietary.
bind	libc / System Call	No	
bsearch	libc	No	
cabs	libm	Yes	
calloc	libc	No	
cbrt	libm	Yes	
cdv_install	libc / System Call	No	Lynx proprietary.

API name	Location	DEV mode only?	Comment
cdv_uninstall	libc / System Call	Yes	Lynx proprietary.
ceil	libm	No	
cfgetispeed	libc	Yes	
cfgetospeed	libc	Yes	
cfsetispeed	libc	Yes	
cfsetospeed	libc	Yes	
chcdev	libc / System Call	Yes	Lynx proprietary.
chdir	libc / System Call	No	
chmod	libc / System Call	No	
chown	libc / System Call	No	
clearerr	libc	No	
clock	libc	No	
clock_getcpuclockid	libc / System Call	No	
clock_getres	libc / System Call	No	
clock_gettime	libc / System Call	No	
clock_nanosleep	libc / System Call	No	

API name	Location	DEV mode only?	Comment
`clock_settime`	`libc / System Call`	No	
`close`	`libc / System Call`	No	
`closedir`	`libc`	No	
`closelog`	`libc`	Yes	
`connect`	`libc / System Call`	No	
`copysign`	`libm`	Yes	
`cos`	`libm`	No	
`cosh`	`libm`	No	
`creat`	`libc`	No	
`create_pinit`	`libc / System Call`	No	Lynx proprietary, for use by cinit only.
`crypt`	`libcrypt`	Yes	
`ctermid`	`libc`	Yes	
`ctime`	`libc`	No	Obsolete.
`ctime_r`	`libc`	No	Obsolete.
`difftime`	`libc`	No	
`div`	`libc`	No	
`dr_install`	`libc / System Call`	No	Lynx proprietary.

API name	Location	DEV mode only?	Comment
dr_uninstall	libc / System Call	Yes	Lynx proprietary.
drand48	libc	Yes	
dup	libc / System Call	No	
dup2	libc / System Call	No	
encrypt	libcrypt	Yes	
endgrent	libc	Yes	
endhostent	libc	Yes	
endnetent	libc	Yes	
endprotoent	libc	Yes	
endpwent	libc	Yes	
endservent	libc	Yes	
environ	libc	No	Variable instantiated in libc.
erand48	libc	Yes	
erf	libm	Yes	
erfc	libm	Yes	
errno	Macro	No	Defined in errno.h.
execl	libc	No	

API name	Location	DEV mode only?	Comment
execle	libc	No	
execlp	libc	No	
execv	libc	No	
execve	libc / System Call	No	
execvp	libc	No	
exit	libc	No	
exp	libm	No	
expm1	libm	Yes	
fabs	libm	No	
fchdir	libc / System Call	No	
fchmod	libc / System Call	No	
fchown	libc / System Call	Yes	
fclose	libc	No	
fcntl	libc / System Call	No	
FD_CLR	Macro	No	Defined in sys/types.h.
FD_ISSET	Macro	No	Defined in sys/types.h.
FD_SET	Macro	No	Defined in sys/types.h.

API name	Location	DEV mode only?	Comment
FD_ZERO	Macro	No	Defined in sys/types.h.
fdatasync	libc / System Call	No	
fdopen	libc	No	
feof	Macro / libc	No	Macro defined in stdio.h.
ferror	Macro / libc	No	Macro defined in stdio.h.
fflush	libc	No	
ffs	libc	Yes	
fgetc	libc	No	
fgetpos	libc	No	
fgets	libc	No	
fileno	Macro / libc	No	Macro defined in stdio.h.
flock	libc / System Call	Yes	
flockfile	libc	No	
floor	libm	No	
fmod	libm	No	
fopen	libc	No	
fork	libc / System Call	No	

API name	Location	DEV mode only?	Comment
`fpathconf`	`libc`	Yes	
`fprintf`	`libc`	No	
`fputc`	`libc`	Yes	
`fputs`	`libc`	Yes	
`fread`	`libc`	No	
`free`	`libc`	No	
`freeaddrinfo`	`libc`	No	
`freopen`	`libc`	No	
`frexp`	`libc`	No	
`fscanf`	`libc`	Yes	
`fseek`	`libc`	No	
`fseeko`	`libc`	No	
`fsetpos`	`libc`	No	
`fstat`	`libc / System Call`	No	
`fstatfs`	`libc / System Call`	No	Lynx proprietary.
`fsync`	`libc / System Call`	No	
`ftell`	`libc`	No	

API name	Location	DEV mode only?	Comment
ftello	libc	No	
ftok	libc	Yes	
ftruncate	libc / System Call	No	
ftrylockfile	libc	No	
ftw	libc	Yes	Obsolete. Use ntfw() instead.
funlockfile	libc	No	
fwrite	libc	No	
getaddrinfo	libc	No	
getc	libc	No	
getchar	libc	Yes	
getcwd	libc	No	
getdents	libc / System Call	No	
getdtablesize	libc / System Call	No	
getegid	libc / System Call	No	
getenv	libc	No	
geteuid	libc / System Call	No	
getgid	libc / System Call	No	

API name	Location	DEV mode only?	Comment
getgrent	libc	Yes	
getgrgid	libc	Yes	
getgrgid_r	libc	Yes	
getgrnam	libc	Yes	
getgrnam_r	libc	Yes	
getgroups	libc / System Call	No	
gethostent	libc	Yes	
gethostname	libc / System Call	No	
getitimer	libc / System Call	No	Obsolete.
getlogin	libc	Yes	
getlogin_r	libc	Yes	
getnameinfo	libc	No	
getnetbyaddr	libc	Yes	
getnetbyname	libc	Yes	
getnetent	libc	Yes	
getopt	libc	Yes	
getpeername	libc / System Call	No	

API name	Location	DEV mode only?	Comment
getpgrp	libc / System Call	No	
getpid	libc / System Call	No	
getppid	libc / System Call	No	
getpriority	libc / System Call	No	
getprotobyname	libc	Yes	
getprotobynumber	libc	Yes	
getprotoent	libc	Yes	
getpwent	libc	Yes	
getpwnam	libc	Yes	
getpwnam_r	libc	Yes	
getpwuid	libc	Yes	
getpwuid_r	libc	Yes	
getrlimit	libc / System Call	No	
getrusage	libc / System Call	Yes	
gets	libc	Yes	Obsolete. Use fgets()/getline() instead.
getscheduler	libc / System Call	No	
getservbyname	libc	Yes	

API name	Location	DEV mode only?	Comment
getservbyport	libc	Yes	
getservent	libc	Yes	
getsockname	libc / System Call	No	
getsockopt	libc / System Call	No	
gettimeofday	libc / System Call	No	Obsolete.
getuid	libc / System Call	No	
gmtime	libc	No	
gmtime_r	libc	No	
hcreate	libc	Yes	
hdestroy	libc	Yes	
hsearch	libc	Yes	
htonl	libc	No	
htons	libc	No	
hypot	libm	Yes	
inet_addr	libc	Yes	
inet_ntoa	libc	Yes	
inet_ntop	libc	No	

API name	Location	DEV mode only?	Comment
`inet_pton`	`libc`	No	
`info`	`libc / System Call`	No	Lynx proprietary.
`Init_Global_Resources`	`libc / System Call`	No	Lynx proprietary, for use by cinit only.
`initstate`	`libc`	Yes	
`ioctl`	`libc / System Call`	No	
`isalnum`	`Macro / libc`	No	Macro defined in ctype.h.
`isalpha`	`Macro / libc`	No	Macro defined in ctype.h.
`isascii`	`libc`	No	Obsolete.
`isatty`	`libc`	No	
`iscntrl`	`Macro / libc`	No	Macro defined in ctype.h.
`isdigit`	`Macro / libc`	No	Macro defined in ctype.h.
`isgraph`	`Macro / libc`	No	Macro defined in ctype.h.
`isinf`	`Macro`	No	Defined in math.h.
`islower`	`Macro / libc`	No	Macro defined in ctype.h.
`isnan`	`Macro`	No	Defined in math.h.
`isprint`	`Macro / libc`	No	Macro defined in ctype.h.
`ispunct`	`Macro / libc`	No	Macro defined in ctype.h.

API name	Location	DEV mode only?	Comment
isspace	Macro / libc	No	Macro defined in ctype.h.
isupper	Macro / libc	No	Macro defined in ctype.h.
isxdigit	Macro / libc	No	Macro defined in ctype.h.
j0	libm	Yes	
j1	libm	Yes	
jn	libm	Yes	
jrand48	libc	Yes	
kill	libc / System Call	No	
killpg	libc / System Call	Yes	
l64a	libc	Yes	
labs	libc	No	
lcong48	libc	Yes	
ldexp	libc	No	
ldiv	libc	No	
lfind	libc	Yes	
lgamma	libm	Yes	
link	libc / System Call	No	

API name	Location	DEV mode only?	Comment
listen	libc / System Call	No	
localeconv	libc	Yes	
localtime	libc	No	
localtime_r	libc	No	
log	libm	No	
log10	libm	No	
log1p	libm	Yes	
log2	libm	Yes	
logb	libm	Yes	
longjmp	libc	No	
lrand48	libc	Yes	
lsearch	libc	Yes	
lseek	libc / System Call	No	
lseek64	libc / System Call	No	
lstat	libc / System Call	No	
malloc	libc	No	
mblen	libc	No	

API name	Location	DEV mode only?	Comment
mbstowcs	libc	No	
mbtowc	libc	No	
memccpy	libc	Yes	
memchr	libc	No	
memcmp	libc	No	
memcpy	libc	No	
memmove	libc	No	
memset	libc	No	
mkcontig	libc / System Call	Yes	Lynx proprietary.
mkdir	libc / System Call	No	
mkfifo	libc	No	
mknod	libc / System Call	No	
mktime	libc	No	
mmap	libc / System Call	No	
modf	libm	No	
mount	libc / System Call	No	Lynx proprietary.
mprotect	libc / System Call	Yes	

API name	Location	DEV mode only?	Comment
mq_close	libc	No	
mq_getattr	libc	No	
mq_notify	libc	No	
mq_open	libc	No	
mq_receive	libc	No	
mq_send	libc	No	
mq_setattr	libc	No	
mq_timedreceive	libc	No	
mq_timedsend	libc	No	
mq_unlink	libc	No	
mrand48	libc	Yes	
munmap	libc / System Call	No	
nanosleep	libc	No	
newconsole	libc / System Call	Yes	Lynx proprietary.
nextafter	libm	Yes	
nfsmount	libc / System Call	Yes	Lynx proprietary.
nrand48	libc	Yes	

API name	Location	DEV mode only?	Comment
ntohl	libc	No	
ntohs	libc	No	
open	libc / System Call	No	
opendir	libc	No	
optarg	libc	Yes	Variable instantiated in libc.
opterr	libc	Yes	Variable instantiated in libc.
optind	libc	Yes	Variable instantiated in libc.
pathconf	libc	Yes	
pause	libc	No	
pclose	libc	Yes	
perror	libc	No	
pipe	libc / System Call	No	
popen	libc	Yes	
posix_devctl	libc	No	
posix_spawn	libc	Yes	
posix_spawnattr_destroy	libc	Yes	
posix_spawnattr_getflags	libc	Yes	
posix_spawnattr_getpgroup	libc	Yes	

API name	Location	DEV mode only?	Comment
posix_spawnattr_getschedparam	libc	Yes	
posix_spawnattr_getschedpolicy	libc	Yes	
posix_spawnattr_getsigdefault	libc	Yes	
posix_spawnattr_getsigmask	libc	Yes	
posix_spawnattr_init	libc	Yes	
posix_spawnattr_setflags	libc	Yes	
posix_spawnattr_setpgroup	libc	Yes	
posix_spawnattr_setschedparam	libc	Yes	
posix_spawnattr_setschedpolicy	libc	Yes	
posix_spawnattr_setsigdefault	libc	Yes	
posix_spawnattr_setsigmask	libc	Yes	
posix_spawn_file_actions_addclose	libc	Yes	
posix_spawn_file_actions_adddup2	libc	Yes	
posix_spawn_file_actions_addopen	libc	Yes	
posix_spawn_file_actions_destroy	libc	Yes	
posix_spawn_file_actions_init	libc	Yes	
posix_spawnp	libc	Yes	
pow	libm	No	

API name	Location	DEV mode only?	Comment
printf	libc	No	
profil	libc / System Call	Yes	
pthread_atfork	libc	No	
pthread_attr_destroy	libc	No	
pthread_attr_getdetachstate	libc	No	
pthread_attr_getguardsize	libc	No	
pthread_attr_getinheritsched	libc	No	
pthread_attr_getschedparam	libc	No	
pthread_attr_getschedpolicy	libc	No	
pthread_attr_getscope	libc	No	
pthread_attr_getstack	libc	No	
pthread_attr_getstacksize	libc	No	
pthread_attr_init	libc	No	
pthread_attr_setdetachstate	libc	No	
pthread_attr_setguardsize	libc	No	
pthread_attr_setinheritsched	libc	No	
pthread_attr_setschedparam	libc	No	
pthread_attr_setschedpolicy	libc	No	

API name	Location	DEV mode only?	Comment
pthread_attr_setscope	libc	No	
pthread_attr_setstack	libc	No	
pthread_attr_setstacksize	libc	No	
pthread_barrier_destroy	libc	Yes	
pthread_barrier_init	libc	Yes	
pthread_barrier_wait	libc	Yes	
pthread_barrierattr_destroy	libc	Yes	
pthread_barrierattr_getpshared	libc	Yes	
pthread_barrierattr_init	libc	Yes	
pthread_barrierattr_setpshared	libc	Yes	
pthread_cancel	libc	No	
pthread_cleanup_pop	Macro	No	Macro defined in pthread.h.
pthread_cleanup_push	Macro	No	Macro defined in pthread.h.
pthread_cond_broadcast	Macro / libc	No	Macro defined in ipc_1c.h.
pthread_cond_destroy	Macro / libc	No	Macro defined in ipc_1c.h.
pthread_cond_init	Macro / libc	No	Macro defined in ipc_1c.h
pthread_cond_signal	Macro / libc	No	Macro defined in ipc_1c.h.
pthread_cond_timedwait	Macro / libc	No	Macro defined in ipc_1c.h.

API name	Location	DEV mode only?	Comment
pthread_cond_wait	Macro / libc	No	Macro defined in ipc_1c.h.
pthread_condattr_destroy	Macro / libc	No	Macro defined in ipc_1c.h.
pthread_condattr_getclock	libc	No	
pthread_condattr_getpshared	libc	Yes	
pthread_condattr_init	Macro / libc	No	Macro defined in ipc_1c.h.
pthread_condattr_setclock	libc	No	
pthread_condattr_setpshared	libc	Yes	
pthread_create	libc	No	
pthread_detach	libc	No	
pthread_equal	libc	No	
pthread_exit	libc	No	
pthread_getconcurrency	libc	No	Obsolete.
pthread_getcpuclockid	libc	No	
pthread_getschedparam	libc	No	
pthread_getspecific	libc	No	
pthread_join	libc	No	
pthread_key_create	libc	No	
pthread_key_delete	libc	No	

API name	Location	DEV mode only?	Comment
pthread_kill	libc	No	
pthread_mutex_destroy	Macro / libc	No	Macro defined in ipc_1c.h.
pthread_mutex_init	Macro / libc	No	Macro defined in ipc_1c.h.
pthread_mutex_lock	Macro / libc	No	Macro defined in ipc_1c.h.
pthread_mutex_timedlock	libc	No	
pthread_mutex_trylock	Macro / libc	No	Macro defined in ipc_1c.h.
pthread_mutex_unlock	Macro / libc	No	Macro defined in ipc_1c.h.
pthread_mutexattr_destroy	Macro / libc	No	Macro defined in ipc_1c.h.
pthread_mutexattr_getprioceilin g	Macro / libc	No	Macro defined in ipc_1c.h.
pthread_mutexattr_getprotocol	Macro / libc	No	Macro defined in ipc_1c.h.
pthread_mutexattr_getpshared	Macro / libc	Yes	Macro defined in ipc_1c.h.
pthread_mutexattr_init	Macro / libc	No	Macro defined in ipc_1c.h.
pthread_mutexattr_setprioceilin g	Macro / libc	No	Macro defined in ipc_1c.h.
pthread_mutexattr_setprotocol	Macro / libc	No	Macro defined in ipc_1c.h.
pthread_mutexattr_setpshared	Macro / libc	Yes	Macro defined in ipc_1c.h.
pthread_once	libc	No	
pthread_self	libc	No	
pthread_setcancelstate	libc	No	

API name	Location	DEV mode only?	Comment
pthread_setcanceltype	libc	No	
pthread_setconcurrency	libc	No	Obsolete.
pthread_setschedparam	libc	No	
pthread_setschedprio	libc	No	
pthread_setspecific	libc	No	
pthread_sigmask	libc	No	
pthread_testcancel	libc	Yes	
ptrace	libc / System Call	Yes	Lynx proprietary.
putc	libc	Yes	
putchar	libc	Yes	
putenv	libc	No	
puts	libc	Yes	
qsort	libc	No	
raise	libc	No	
rand	libc	No	
rand_r	libc	No	Obsolete.
random	libc	Yes	
read	libc / System Call	No	

API name	Location	DEV mode only?	Comment
readdir	libc	No	
readdir_r	libc	No	
readlink	libc / System Call	Yes	
readv	libc / System Call	Yes	
realloc	libc	No	
reboot	libc / System Call	No	Lynx proprietary.
recv	libc / System Call	No	
recvfrom	libc / System Call	No	
remainder	libm	Yes	
remove	libc	No	
rename	libc / System Call	No	
rewind	libc	Yes	It is recommended that fseek() be used instead.
rewinddir	libc	No	
rmdir	libc / System Call	No	
sbrk	libc / System Call	No	
scalbn	libm	Yes	

API name	Location	DEV mode only?	Comment
scandir	libc	Yes	
scanf	libc	Yes	
sched_get_priority_max	libc	No	
sched_get_priority_min	libc	No	
sched_getparam	libc	No	
sched_getscheduler	libc	No	
sched_rr_get_interval	libc	No	
sched_setparam	libc	No	
sched_setscheduler	libc	No	
sched_yield	libc	No	
seed48	libc	Yes	
seekdir	libc	Yes	
select	libc / System Call	No	
sem_close	libc	No	
sem_destroy	libc	No	
sem_getvalue	libc	No	
sem_init	libc	No	
sem_open	libc	No	

API name	Location	DEV mode only?	Comment
sem_post	libc	No	
sem_timedwait	libc	No	
sem_trywait	libc	No	
sem_unlink	libc	No	
sem_wait	libc	No	
send	libc / System Call	No	
sendto	libc / System Call	No	
setbuf	libc	No	It is recommended that setvbuf() be used instead.
setegid	libc	No	
seteuid	libc	No	
setgid	libc	No	
setgrent	libc	Yes	
setgroups	libc / System Call	No	
sethostname	libc / System Call	No	
setitimer	libc / System Call	No	Obsolete.
setjmp	libc	No	
setkey	libcrypt	Yes	

API name	Location	DEV mode only?	Comment
setlocale	libc	No	
setlogmask	libc	Yes	
setpgid	libc / System Call	No	
setpgrp	libc / System Call	Yes	Obsolete.
setpriority	libc / System Call	No	
setprotoent	libc	Yes	
setpwent	libc	Yes	
setresgid	libc / System Call	No	
setresuid	libc / System Call	No	
setrlimit	libc / System Call	No	
setscheduler	libc / System Call	No	Lynx proprietary.
setservent	libc	Yes	
setsid	libc / System Call	No	
setsockopt	libc / System Call	No	
setstate	libc	Yes	
settimeofday	libc / System Call	Yes	

API name	Location	DEV mode only?	Comment
setuid	libc	No	
setvbuf	libc	No	
shm_open	libc / System Call	No	
shm_unlink	libc / System Call	No	
shutdown	libc / System Call	No	
sigaction	libc / System Call	No	
sigaddset	libc	No	
sigblock	libc / System Call	No	
sigdelset	libc	No	
sigemptyset	libc	No	
sigfillset	libc	No	
sigismember	libc	No	
siglongjmp	libc	No	
signal	libc	No	
sigpause	libc / System Call	No	Obsolete.
sigpending	libc / System Call	No	
sigprocmask	libc / System	No	

API name	Location	DEV mode only?	Comment
	Call		
sigqueue	libc / System Call	No	
sigsetjmp	libc	No	
sigsetmask	libc / System Call	No	
sigsuspend	libc / System Call	No	
sigtimedwait	libc / System Call	No	
sigvec	libc / System Call	Yes	
sigwait	libc / System Call	No	
sigwaitinfo	libc	No	
sin	libm	No	
sinh	libm	No	
sleep	libc	No	
snprintf	libc	No	
socket	libc / System Call	No	
socketpair	libc / System Call	No	
sprintf	libc	No	

API name	Location	DEV mode only?	Comment
sqrt	libm	No	
srand	libc	No	
srand48	libc	Yes	
srandom	libc	Yes	
sscanf	libc	No	
st_build	libc / System Call	No	Lynx proprietary.
st_cancel	libc / System Call	No	Lynx proprietary.
st_detach	libc / System Call	No	Lynx proprietary.
st_join	libc / System Call	No	Lynx proprietary.
st_name	libc / System Call	No	Lynx proprietary.
st_resume	libc / System Call	No	Lynx proprietary.
stat	libc / System Call	No	
statfs	libc / System Call	No	Lynx proprietary.
stderr	Macro	No	Defined in stdio.h.
stdin	Macro	No	Defined in stdio.h.
stdout	Macro	No	Defined in stdio.h.

API name	Location	DEV mode only?	Comment
strcat	libc	No	
strchr	libc	No	
strcmp	libc	No	
strcoll	libc	No	
strcpy	libc	No	
strcspn	libc	No	
strdup	libc	Yes	
strerror	libc	No	
strerror_r	libc	No	
strftime	libc	No	
strlen	libc	No	
strncat	libc	No	
strncmp	libc	No	
strncpy	libc	No	
strpbrk	libc	No	
strrchr	libc	No	
strspn	libc	No	
strstr	libc	No	

API name	Location	DEV mode only?	Comment
strtod	libc	No	
strtok	libc	No	
strtok_r	libc	No	
strtol	libc	No	
strtoll	libc	Yes	
strtoul	libc	No	
strtoull	libc	Yes	
strxfrm	libc	No	
swab	libc	Yes	
symlink	libc / System Call	No	
sync	libc / System Call	No	
sysconf	libc	No	
sysctl	libc / System Call	Yes	
syslog	libc	Yes	
sysppc	libc / System Call	No	Lynx proprietary.
system	libc	Yes	
tan	libm	No	

API name	Location	DEV mode only?	Comment
tanh	libm	No	
tcdrain	libc	Yes	
tcflow	libc	Yes	
tcflush	libc	Yes	
tcgetattr	libc	Yes	
tcgetpgrp	libc	Yes	
tcsendbreak	libc	Yes	
tcsetattr	libc	Yes	
tcsetpgrp	libc	Yes	
tdelete	libc	Yes	
telldir	libc	Yes	
tempnam	libc	Yes	Obsolete. Use mkstemp() instead.
tfind	libc	Yes	
time	libc	No	
timer_create	libc	No	
timer_delete	libc / System Call	No	
timer_getoverrun	libc / System Call	No	
timer_gettime	libc / System	No	

API name	Location	DEV mode only?	Comment
	Call		
timer_settime	libc / System Call	No	
times	libc / System Call	No	
tmpfile	libc	Yes	
tmpnam	libc	Yes	Obsolete. Use mkstemp() instead.
tolower	libc	No	
toupper	libc	No	
tsearch	libc	Yes	
ttyname	libc	Yes	
ttyname_r	libc	Yes	
twalk	libc	Yes	
tzname	libc	No	Variable instantiated in libc.
tzset	libc	No	
ulimit	libc / System Call	Yes	Obsolete.
umask	libc / System Call	No	
umount	libc / System Call	No	Lynx proprietary.
uname	libc / System Call	No	

API name	Location	DEV mode only?	Comment
`ungetc`	`libc`	No	
`unlink`	`libc / System Call`	No	
`utime`	`libc`	Yes	Obsolete. Use getrlimit()/setrlimit() instead.
`utimes`	`libc / System Call`	Yes	
`va_arg`	`Macro`	No	Defined in stdarg.h.
`va_end`	`Macro`	No	Defined in stdarg.h.
`va_start`	`Macro`	No	Defined in stdarg.h.
`vfprintf`	`libc`	No	
`vprintf`	`libc`	No	
`vsnprintf`	`libc`	No	
`vsprintf`	`libc`	No	
`wait`	`libc / System Call`	No	
`wait3`	`libc / System Call`	Yes	
`waitpid`	`libc / System Call`	No	
`wcstombs`	`libc`	No	
`wctomb`	`libc`	No	
`write`	`libc / System Call`	No	

API name	Location	DEV mode only?	Comment
`writev`	`libc / System Call`	Yes	
`y0`	`libm`	Yes	
`y1`	`libm`	Yes	
`yield`	`libc / System Call`	No	
`yn`	`libm`	Yes	

APPENDIX C LynxOS-178 SKDB Commands

This Appendix describes common SKDB commands usage as well as lists all SKDB commands supported for LynxOS-178.

General notes

Parameter Validation

SKDB performs little validation for command arguments. Although SKDB catches most memory access faults resulting from SKDB commands, improper arguments may result in a system freeze.

Symbol Information

SKDB uses the kernel symbol table that is loaded at the startup time for symbol lookup. SKDB cannot do interactive symbolic debugging with a stripped kernel.

Address Expressions

SKDB accepts simple address expressions with symbolic notations for most commands that accept memory address parameters. The syntax is as follows:

- Number: Hexadecimal if starting with "`0x`"; octal if starting with "`0`"; or otherwise decimal.

- Symbol: The symbol absolute virtual address value (note the PowerPC requires a preceding dot for text symbols).
- Operator: + and **-** represent addition and subtraction respectively. Operations are performed left to right without precedence or associatively.

For example, if it is needed to set a breakpoint at the current PC address plus 20 bytes, the following command can be issued:

```
* b <pc_addr>+0x14
```

where *<pc_addr>* is the current PC address available in the kernel status display string.

Default Virtual Address Space

The LynxOS-178 memory model assigns a separate virtual address space to each process (kernel threads belong to process 0). Although all processes share the same kernel text, kernel data, and kernel heap in the kernel, each supervisor stack still belongs to its respective process virtual address space.

To access a memory location of a noncurrent process, use the T command to get the memory location PHYSBASE address.

The PHYSBASE address is the region of kernel address space where a mirror image of the system physical memory is mapped (aliased). Since any page that the kernel may access is found in this region and the page is visible to all processes at the same virtual address, SKDB uses PHYSBASE for quick memory reference in a noncurrent process virtual address space.

SKDB Commands

The following table lists all SKDB commands supported on LynxOS-178.

Table C-1. SKDB Commands

Command Format	Example	Description
Examine Memory		
<addr> [*<size>*]	* **0xdb100000**	Displays 4 or *<size>* bytes starting at memory location *<addr>*.
$*<symbol>* [*<size>*]	* **$currpid 10**	Displays 4 or *<size>* bytes at symbol *<symbol>*.
m {+|*<addr>*|*<symbol>* [*<size>*]}	* **m currpid 256**	Displays 64 or *<size>* bytes starting at memory location *<addr>* or symbol *<symbol>*.

Command Format	Example	Description
T *<vaddr>* [*<pid>*]	* **T 0xdb100000 9**	Translates virtual memory location *<vaddr>* to physical address using process ID *<pid>* or current process mapping.
+ [*<size>*]	* **+**	Displays the next 32 or *<size>* bytes of memory.
- [*<size>*]	* **- 10**	Displays the previous 32 or *<size>* bytes of memory.
Change Memory		
c {*<addr>*|*<symbol>*|%*<reg>*} *<data>*	* **c 0xdb100000 0x200**	Stores *<data>* as a word (4 bytes) at memory location *<addr>*, symbol *<symbol>*, or general register *<reg>*.
Find Symbol		
f <addr>	* **f 0xdb100000**	Displays symbol with closest offset to memory location *<addr>*.
&<symbol>	* &currpid	Displays the *<symbol>* address.
Display Data Structure		
s *<type>* *<arg>*	* **s proc 160**	Display data structure specified by *<type>* and *<arg>*.

Command Format	Example	Description
s st [<*tid*>|<*addr*>[1]]	*** s st 5**	Displays contents of st_entry structure for thread ID <*tid*>, memory location <*addr*>, or the current thread.
s proc [<*pid*>|<*addr*>[1]]	*** s proc**	Displays contents of pentry structure for process ID <*pid*>, memory location <*addr*>, or the current process.
s pss [<*pid*>|<*addr*>[1]]	*** s pss 0xdb100000**	Displays contents of pssentry structure for process ID <*pid*>, memory location <*addr*>, or the current process.
s inode{<*num*>|<*addr*>[1]}	*** s inode 45**	Displays contents of inode_entry structure for index <*num*> or memory location <*addr*>.
s block {<*num*>|<*addr*>[1]}	*** s block 0xdb100000**	Displays contents of buf_entry structure for index <*num*> or memory location <*addr*>.
s ihead {<*num*>|<*addr*>[1]}	*** s ihead 30**	Displays contents of ihead_entry structure for index <*num*> or memory location <*addr*>.
s file {<*num*>|<*addr*>[1]}	*** s file 0xdb100000**	Displays contents of file structure for index <*num*> or

Command Format	Example	Description
		memory location *<addr>*.
s fifo {*<num>*|*<addr>*[1]}	*** s fifo 49**	Displays contents of `fifo` structure for index *<num>* or memory location *<addr>*.
s fdentry *<addr>*[1]	*** s fdentry 0xdb100000**	Displays contents of `fdentry` structure for memory location *<addr>*.
s cdev {*<num>*|*<addr>*[1]}	*** s cdev 0xdb100000**	Displays contents of `cdevsw_entry` structure at index *<num>* or at memory location *<addr>*.
s bdev {*<num>*|*<addr>*[1]}	*** s bdev 0**	Displays contents of `bdevsw_entry` structure for index *<num>* or memory location *<addr>*.
s ectx {*<tid>*|*<addr>*[1]}	*** s ectx 6**	Displays contents of `econtext` structure for thread ID *<tid>* or memory location *<addr>*.
s fctx {*<tid>*|*<addr>*[1]}	*** s fctx 0xdb100000**	Displays contents of `fcontext` structure for thread ID *<tid>* or memory location *<addr>*.

Command Format	Example	Description
`s sem [[tid]|[st_t]|[pentry]|<sem_addr>]]`	`* s sem`	Shows current status of `csems` and PI mutexes of all threads, or optionally thread with ID *<tid>*, or thread specified by *<st_t>* address, or process specified by *<pentry>* address, or semaphore specified by *<sem_addr>* address.
`s +`	`* `**`s +`**	Displays contents of the next (in memory) data structure of the type being displayed.
`s -`	`* `**`s -`**	Displays contents of the previous (in memory) data structure of the type being displayed.
`s !!`	`* `**`s !!`**	Repeats contents (in memory) data structure of the type being displayed.
`s next`	`* `**`s next`**	Displays contents of the data structure pointed to by the `next` (or equivalent) field of the currently displayed data structure.

Command Format	Example	Description
s prev	* **s prev**	Displays contents of the data structure pointed to by the prev (or equivalent) field of the currently displayed data structure.
Stack Trace		
t [*<pid>*|-*<tid>*]	* **t -5**	Displays symbolic stack trace of process ID *<pid>*, thread ID *<tid>*, or the current thread.
v	* **v**	Toggles verbose mode for trace.
Display Registers, Processes, and Set Priority		
r[g] [*<pid>*|-*<tid>*]	* **r**	Displays CPU registers of process ID *<pid>* main thread, thread ID *<tid>*, or the current thread[2]. When rg (instead of r) is entered, displays general registers only.
p [vm*<id>*] [*<count>*]	* **p 20**	Displays process table (all or *<count>* items). For LynxOS-178 the list of processes can be filtered by VM ID. For this, the vm*<id>* argument can be used. *<id>* is a VM ID.
P *<prio>* *<tid>*	* **P 15 8**	Set kernel level priority of thread

Command Format	Example	Description
		ID *<tid>* to priority *<prio>*.
Breakpoints		
b	*** b**	Displays all breakpoints set.
b {*<addr>*|*<symbol>*}	*** b 0xdb100000**	Sets breakpoint at memory location *<addr>* or symbol <symbol>.
u [*<num>* [*<num>*...]]	*** u 5**	Unsets breakpoint *<num>* or multiple specified *<num>*s. If no *<num>* is specified, SKDB will ask whether all breakpoints should be unset and on confirmation unsets all breakpoints.
Watchpoints		
B[3]	*** B**	Shows all watchpoints set.
B *<num>* *<addr>* [*<mode>* *<size>*] [! *<ignore>*...][3]	*** B 1 currtptr w**	Sets watchpoint *<num>* at the memory location *<addr>* for *<mode>* (read, write, or read/write access) and byte size *<size>*. Optionally, ignores hits at *<ignore>* or multiply specified *<ignore>* addresses.

Command Format	Example	Description
`U` *<num>*[3]	`*` **`U 5`**	Unsets watchpoint *<num>*. If no *<num>* is specified, SKDB will ask whether all watchpoints should be unset and on confirmation unsets all watchpoints.
Single-Stepping		
`x`	`*` **`x`**	Single-steps current thread.
Disassembly		
`d [`*<addr>*`|`*<symbol>* `[`*<count>*`]]`	`*` **`d 20`**	Disassembles 10 or *<count>* instructions at the current PC, or at memory location *<addr>* or symbol *<symbol>*.
Miscellaneous		
`R`	`*` **`R`**	Restarts the operating system.
`z`	`*` **`z`**	Reset the default SKDB hot key.
`h`	`*` **`h`**	Displays a description of all available commands.
`?`	`*` **`?`**	Same as h.

[1]The address value must point to a valid table entry.

[2]Some architecture may not save all registers upon context switching.
[3] Not all target systems support the command.

APPENDIX D SpyKer Events and Commands

This Appendix describes SpyKer events and their payloads as well as the commands that can be specified in SpyKer command files.

SpyKer Events and Payloads

The following table lists all SpyKer events and their payloads supported on LynxOS-178.

Table D-1. SpyKer Events and Payloads

Event Number	Event Description	Payloads	
		Short	Long
0	`Context switch`	Superpid[1]	`struct te_cswitch_payload`
1	`System call`	System call #	N/A
2	`Interrupt`	Interrupt #	N/A
3	`Return from interrupt`	Interrupt #	N/A
4	`Processor exception`	Exception #	N/A
5	`Thread/Process stop`	N/A	N/A
6	`Program load`	PID of the process	Name of program loaded
7	`Thread/Process wait`	0	N/A
8	`Thread/Process wakeup`	Superpid of thread	N/A

Table D-1: SpyKer Events and Payloads (Continued)

Event Number	Event Description	Payloads	
		Short	Long
`9`	`Process exit`	N/A	N/A
`10`	`User thread exit`	N/A	N/A
`11`	`System thread exit`	Thread ID	N/A
`12`	`Return from system call`	System call #	`struct te_rsyscall_payload`
`13`	`Signal delivery (caught)`	Signal #	N/A
`14`	`Signal delivery (not caught)`	Signal #	N/A
`15`	`Memory allocation`	# of pages[2] requested	# of free pages (before allocation)
`16`	`Memory free`	# of pages being freed	# of free pages (before free)
`17`	`Kernel malloc`	# of bytes requested	Return value
`18`	`Kernel free`	# of bytes freed	Address of memory
`19`	`New system thread`	New thread ID	Thread name
`20`	`New user thread`	Superpid of new thread or - 1 for the first thread	`struct te_newut_payload`
`21`	`New process (fork)`	PID of new process	N/A
`22`	`Trace start`[3]	`getime()`	`struct te_start_payload`
`23`	`Existing process`[3]	PID or -1 as terminator	`struct te_exproc_payload`
`24`	`Existing thread`[3]	Thread ID or -1 as terminator	`struct te_exthrd_payload`
`25`	`Unknown event`	Unrecognized event type #	N/A
`26`	`Wrap mode event`[3]		`struct te_wrap_payload`

Table D-1: SpyKer Events and Payloads (Continued)

Event Number	Event Description	Payloads	
		Short	Long
27	Priority inheritance	Superpid	struct te_prinherit_payload
30	Thread rename	Thread ID	New thread name
31	Reserved 31	N/A	N/A
32	Reserved 32	N/A	N/A
33	Reserved 33	N/A	N/A
34	Reserved 34	N/A	N/A
35	Reserved 35	N/A	N/A
36	Reserved 36	N/A	N/A
37	Reserved 37	N/A	N/A
38	Reserved 37	N/A	N/A
39	Resource (de)allocation	VM/resource ID	Resource change
40	Resource usage/limits4	VM ID	struct te_178use_payload
41	Exceeding of a resource limit	VM/resource ID	N/A
42	Start/run-time VM schedules	Number of minor frames	struct te_178sched_payload
43	User 43	Arbitrary numeric value	Null-terminated string
44	User 44	Arbitrary numeric value	Null-terminated string
45	User 45	Arbitrary numeric value	Null-terminated string

1. The LynxOS-178 superpid is an encoding of the process ID and the thread ID.
2. Page size is normally 4 KB.
3. SpyKer private event.

SpyKer Commands

The following table lists all SpyKer commands that can be specified in SpyKer command files.

NOTE: All commands require the space before semicolon.

Table D-2. SpyKer Commands

Command	Description
`:strig` *<event>* `[`*<payload>*`] ;`	Sets the start trigger.
`:etrig` *<event>* `[`*<payload>*`] ;`	Sets the end trigger.
`:buffers [`*<start buffer size> <main buffer size>* *<end buffer size>*`] ;`	Sets buffer sizes in kilobytes. The sizes should be specified as decimal integers.

Command	Description
:filter <process_filter> <event_filter> <vm_filter> <cpu_filter>	Sets the event filter. By default all events are collected. Filters can be specified in arbitrary order. Only one filter of each type can be specified. The *<process_filter>* filter allows selecting processes to collect events for and can be specified by one or more options below. Only one option of each type can be specified. *uid [-] <uid>* - collect events only of those processes that have UID equal to *<uid>*.If "-" is specified, then include events only of those processes that have UID not equal to *<uid>*. *gid [-] <gid>* - collect events only of those processes that have GID equal to *<gid>*. If "-" is specified, then include events only of those processes that have GID not equal to *<gid>*. *pid [-] <pid>* - collect events only of the process that has PID equal to *<pid>*. If "-" is specified, then include events only of those processes that have PID not equal to *<pid>*. *pgrp [-] <pgrp>* - collect events only of those processes that have PGRP equal to *<pgrp>*.If "-" is specified, then include events only of those processes that have PGRP not equal to *<pgrp>*. The *<event_filter>* filter allows selecting events to collect and can be specified by one of the options below: events *<count> <32-bit mask>...<32-bit mask>* - collect only the events marked as 1 in the bit mask. Bit masks should be separated by spaces. *<count>* is a number of specified masks. events *[ids|no_ids] <count> <i1>* ... <in> - collect only events with specified numbers. If *no_ids* is specified, collect only those events that are not specified. Event numbers should be separated by spaces. *<count>* is a number of specified events. The *<vm_filter>* filter allows selecting VMs to collect events on and can be set by the following option: *no_vm <id1>...<idn>* - include only events that occur on VMs different from the specified ones. VM IDs should be separated by spaces. The *<cpu_filter>* filter allows selecting CPUs to collect events on and can be set by the following option: *no_cpu <i1>...<in>*- collect only events that occur on CPUs different from the specified ones. CPU numbers should be separated by spaces. NOTE: The following events are always captured regardless to the selected events, CPU, or VM filters: • Trace start • Existing process • Wrap mode • Resource usage/limits • Start/runtime schedules. Additionally, the process, VM and CPU filters are not applied to the following events: • New system thread • New user thread • New process (fork)• Program load • Context switch

<table>
<tr><th>Command</th><th>Description</th></tr>
<tr><td><code>:file [- | <filesize>] ;</code></td><td>:file <filesize> ; specifies that the trace should be collected in out.trc file where <filesize> is the maximal file size in kilobytes.

The trace collection will stop after the maximal file size is reached.

: file - ; specifies that the trace should be collected in memory and written in the out.trc file after the process completes.</td></tr>
<tr><td><code>:wrap <0 | 1> ;</code></td><td>Disables/enables wrap. The wrap is disabled if the trace is saved in the file.</td></tr>
<tr><td><code>:start</code></td><td>Starts tracing. Should be the last command in the command script.</td></tr>
</table>