RedHawk™ Linux®
Frequency-Based Scheduler User's Guide

0898005-350
January 2013

The information contained in this document is believed to be correct at the time of publication. It is subject to change without notice. Concurrent makes no warranties, expressed or implied, concerning the information contained in this document.

To report an error or comment on a specific portion of the manual, photocopy the page in question and mark the correction or comment on the copy. Mail the copy (and any additional comments) to Concurrent Computer Corporation, 2881 Gateway Drive, Pompano Beach, FL 33069. Mark the envelope **"Attention: Publications Department**." This publication may not be reproduced for any other reason in any form without written permission of the publisher.

Printed in U. S. A.

Revision History:	Level:	Effective With:
August 2002	000	RedHawk Linux 1.1 & 1.2
September 2003	110	RedHawk Linux 1.4
December 2003	200	RedHawk Linux 2.0
March 2004	210	RedHawk Linux 2.1
July 2004	220	RedHawk Linux 2.2
May 2005	230	RedHawk Linux 2.3
March 2006	240	RedHawk Linux 4.1
May 2007	250	RedHawk Linux 4.2
October 2007	260	RedHawk Linux 4.2
April 2008	300	RedHawk Linux 5.1
January 2009	320	RedHawk Linux 5.2.1
August 2009	330	RedHawk Linux 5.2.4
November 2009	340	RedHawk Linux 5.4
January 2013	350	RedHawk Linux 6.3

Preface

Scope of Manual

This manual describes the Frequency-Based Scheduler (FBS) and the Performance Monitor (PM) operating on the RedHawk Linux operating system. These utilities are used for scheduling processes at specified frequencies for the purpose of monitoring performance.

Instructions for using the associated real-time command processor (**rtcp**) and the C and FORTRAN library routines are included in this guide. Not included are instructions for using NightSim™, a graphical user interface for FBS and PM, which is documented in the *NightSim RT User's Guide.*

Structure of Manual

This manual consists of the following sections:

- Chapter 1, *Introduction*, provides an introduction to the Frequency-Based Scheduler and the Performance Monitor as well as software, configuration and access privilege requirements.
- Chapter 2, *Using the Frequency-Based Scheduler*, provides a description of the Frequency-Based Scheduler, its capabilities and user interface summary.
- Chapter 3, *Timing Sources*, explains how to use a real-time clock, an edge-triggered interrupt and a user-supplied device as the timing source for a frequency-based scheduler.
- Chapter 4, *Using the Performance Monitor*, provides a description of the Performance Monitor, its capabilities and user interface summary.
- Chapter 5, *Using rtcp*, explains the procedures for using the real-time command processor, **rtcp**, and provides reference information for each of its commands.
- Chapter 6, *The C Library Interface*, describes the C Library interface to the Frequency-Based Scheduler and the Performance Monitor and provides reference information for each of the routines.
- Chapter 7, *The FORTRAN Library Interface*, describes the FORTRAN Library interface to the Frequency-Based Scheduler and the Performance Monitor and provides reference information for each of the routines.
- Appendix A, *Example rtcp Script*, contains an example **rtcp** script.
- Appendix B, *rtcp Error Messages*, provides explanations of the errors that may be reported by **rtcp**.
- Appendix C, *Example C Interface*, contains an example program that shows how to use the C library interface to the Frequency-Based Scheduler and the Performance Monitor.

The *Glossary* contains definitions of technical terms that are important to understanding the concepts presented in this book.

The *Index* contains an alphabetical reference to key terms and concepts and numbers of pages where they occur in the text.

Syntax Notation

The following notation is used throughout this manual:

italic	Books, reference cards, and items that the user must specify appear in *italic* type. Special terms may also appear in *italic*.
`list bold`	User input appears in **`list bold`** type and must be entered exactly as shown. Names of directories, files, commands, options and man page references also appear in **`list bold`** type.
`list`	Operating system and program output such as prompts, messages and listings of files and programs appears in `list` type.
`[]`	Brackets enclose command options and arguments that are optional. You do not type the brackets if you choose to specify these options or arguments.
hypertext links	When viewing this document online, clicking on chapter, section, figure, table and page number references will display the corresponding text. Clicking on Internet URLs provided in blue type will launch your web browser and display the web site. Clicking on publication names and numbers in red type will display the corresponding manual PDF, if accessible.

Related Publications

The following table lists related Concurrent documentation. Click on the red entry to display the document PDF. These documents are also available by clicking on the "Documents" icon on the desktop and from Concurrent's web site at **www.ccur.com**.

Title	Pub No.
RedHawk Linux Release Notes Version x.x	0898003
RedHawk Linux User's Guide	0898004
Real-Time Clock and Interrupt Module (RCIM) User's Guide	0898007
NightSim RT User's Guide	0890480
NightView RT User's Guide	0890395

where *x.x* = release version

Contents

Chapter 6 The C Library Interface

Chapter 7 The FORTRAN Library Interface

Screens

Illustrations

Tables

1
Introduction

This chapter introduces you to the Frequency-Based Scheduler (FBS) and the Performance Monitor (PM). It also provides software and configuration requirements and discusses the privileges and capabilities needed to use these utilities.

Frequency-Based Scheduler Overview

The Frequency-Based Scheduler (FBS) is a task synchronization mechanism used to initiate processes at specified frequencies. Frequencies can be based on high-resolution clocks such as those provided by the Real-Time Clock and Interrupt Module (RCIM), an external interrupt source, or the completion of a cycle. The processes are then scheduled using a priority-based scheduler. FBS can control the periodic execution of multiple, coordinated processes utilizing major and minor cycles with overrun detection.

When used in conjunction with the Performance Monitor, FBS can be used to determine the best way of allocating processors to various tasks for a particular application.

You can easily configure FBS to meet the needs of specific applications. More than one frequency-based scheduler can be configured, each with different capabilities allowing a variety of scheduling techniques for different purposes. A full description of the Frequency-Based Scheduler is provided in Chapter 2.

Access to the major functions associated with frequency-based scheduling is achieved through the following interfaces:

- **`rtcp`**, the real-time command processor. **`rtcp`** is explained in Chapter 5.
- a set of library routines that can be called from application programs written in C. The library routines are explained in Chapter 6.
- a set of library routines that can be called from application programs written in FORTRAN. The library routines are explained in Chapter 7.
- NightSim, a Graphical User Interface (GUI) to FBS. NightSim is explained in the *NightSim RT User's Guide.*

Performance Monitor Overview

The Performance Monitor (PM) monitors use of the CPU by processes that are scheduled on the Frequency-Based Scheduler. Values obtained assist in determining how to redistribute processes among processors for improved load balancing and processing efficiency. A full description of the Performance Monitor and its capabilities is provided in Chapter 4.

Access to the major functions associated with the Performance Monitor is achieved through the following interfaces:

- **`rtcp`**, the real-time command processor. **`rtcp`** is explained in Chapter 5.
- a set of library routines that can be called from application programs written in C. The library routines are explained in Chapter 6.
- a set of library routines that can be called from application programs written in FORTRAN. The library routines are explained in Chapter 7.
- NightSim, a Graphical User Interface (GUI) to PM. NightSim is explained in the *NightSim RT User's Guide.*

Software Requirements

The Frequency-Based Scheduler (FBS) RPM distributed with the RedHawk Linux operating system provides kernel support for the Frequency-Based Scheduler, the Performance Monitor and **`rtcp(1)`**. This software must be installed in order to access FBS and PM. Refer to the appropriate version of the *RedHawk Linux Release Notes* for installation procedures.

If you want to use RCIM devices for timing FBS processes, the Real-Time Clock and Interrupt Module (RCIM) must be installed and configured in the system. Refer to the appropriate version of the *RedHawk Linux Release Notes* for installation procedures. Refer to the *Real-Time Clock and Interrupt Module (RCIM) User's Guide* for information about configuring the RCIM driver and using the clocks and interrupts.

Configuration

Configuration requirements for the Frequency-Based Scheduler and the Performance Monitor are listed below. For information on how to change kernel parameters, see the "Configuring and Building the Kernel" chapter in the *RedHawk Linux User's Guide.*

Frequency-Based Scheduler

The following system tunable parameters affect operation of FBS.

FBSCHED — This parrameter, accessible through the Frequency-Based Scheduler selection of the Kernel Configuration GUI, configures FBS in the system. By default, FBS is configured in each of the RedHawk Linux pre-built kernels. It can be configured as a module, if desired.

RCIM_IRQ_EXTENSIONS — This parameter, accessible through the Device Drivers selection of the Kernel Configuration GUI, provides hooks for special services in the RCIM driver designed for FBS use.

Performance Monitor

In addition to having FBS configured in the kernel, the Performance Monitor requires the following:

FBSCHED_PM — This tunable, accessible through the Frequency-Based Scheduler selection of the Kernel Configuration GUI, is used to configure the Performance Monitor in the system. It is configured by default in each of the RedHawk Linux pre-built kernels.

HRACCT — This parameter, accessible through the General Setup selection of the Kernel Configuration GUI, is required in order to use the Performance Monitor. It enables high resolution process accounting, which provides the means for measuring the execution time of each process for its timing values. This parameter is configured by default in the RedHawk Linux debug and trace kernels.

Privileges

RedHawk Linux supports a Pluggable Authentication Module (PAM) called **pam_capability(8)** that provides a role-based access control scheme. In this scheme, you set up a series of roles in the **capability.conf(5)** file. A role is defined as a set of valid Linux capabilities.

RedHawk systems are configured with the "fbscheduser" role. This role defines the set of capabilities needed to execute the libraries and kernel code that make up the FBS subsystem. The FBS user must be configured to use (at a minimum) the capabilities provided by the "fbscheduser" role by default. This is accomplished by adding a line in **/etc/security/capability.conf** for each FBS user with the "fbscheduser" role defined:

```
user     username       fbscheduser
```

where *username* is the name of the user.

Programs that are scheduled on a frequency-based scheduler might require other capabilities to be able to make the system calls that they utilize. The fbscheduser role may be used as a building block to create a more general purpose role. For example, the following two entries would grant all users in a group named "software" the capabilities to use the FBS subsystem (fbscheduser) as well as the capability to use **mlock(2)** and **mlockall(2)** (cap_ipc_lock) along with other capabilities in some user-defined role (user_defined).

```
role     realtimeuser   fbscheduser cap_ipc_lock user_defined
group    software       realtimeuser
```

During system initialization the RCIM device files, by default, are created writeable by all users (mode=0666). If the administrator wishes to restrict the permissions to only certain users, this must be done in such a way as to allow FBS users permission to write these files. One way to achieve this is to define a group that provides write access to the RCIM device files and assign FBS users to that group. An example of how to do this follows.

To add a group named "realtimer", do the following:

```
# groupadd -r realtimer
```

Next, give the realtimer group write access to the RCIM device files by editing **/etc/sysconfig/rcim** and setting the following:

```
RCIM_OWN=root
RCIM_GRP=realtimer
RCIM_PERM=0664
```

Finally, add FBS users to the group. The example here adds "joe" to the realtimer group:

```
# adduser -u 50 -g 100 -d /home/joe -G realtimer -s /bin/bash -c "Joe User" joe
```

2
Using the Frequency-Based Scheduler

This chapter contains a description of the Frequency-Based Scheduler and its capabilities. It explains how scheduler frequency is defined, how processes are scheduled and what overruns are and how they are managed. The user interfaces for accessing the scheduler are introduced.

What Is the Frequency-Based Scheduler?

The Frequency-Based Scheduler (FBS) is a task synchronization mechanism that enables you to run processes at specified frequencies, and optionally with a specified deadline. A number of frequency-based schedulers can be configured and used simultaneously. Scheduling frequencies are based on a user-defined number of minor cycles that compose a major frame. Each scheduler can use any of several available timing sources to define scheduling frequency or can schedule at the completion of a cycle. In addition, frame overruns can be monitored.

Specifically, FBS allows you to:

- define frequency in terms of the duration of a minor cycle and the number of minor cycles per major frame
- specify the scheduling parameters with which processes are scheduled
- control all scheduling features from one processor (that is, schedule and query a process on any processor)
- detect deadline violations for frequency-based scheduled processes having assigned deadline times
- detect frame overruns for all frequency-based scheduled processes
- obtain the status of a single frequency-based scheduled process, all frequency-based scheduled processes on a single processor, or all frequency-based scheduled processes on all processors
- remove one or all frequency-based scheduled processes from a scheduler
- reschedule a frequency-based scheduled process
- start, stop, and resume scheduling on a frequency-based scheduler
- connect a timing source to and disconnect it from a frequency-based scheduler
- control use of the real-time clock device as the timing source for a frequency-based scheduler
- configure up to 100 frequency-based schedulers system-wide in a single processor or multiprocessor environment
- use both frequency-based scheduling and static priority scheduling simultaneously

- set the soft overrun limit for a frequency-based scheduled process
- query the soft overrun limit and the total number of soft overruns incurred by a frequency-based scheduled process.

How Is Scheduler Frequency Defined?

Scheduler frequency is defined, in part, by defining the number of *minor cycles* that compose a *major frame* using either the **`fbsconfigure(3)`** library routine or **`rtcp(1)`**.

As the smallest unit of frequency maintained by a frequency-based scheduler, a minor cycle has a duration associated with it. The duration is the time that elapses between interrupts generated by the timing source attached to the scheduler. If, for example, the timing source is a real-time clock, the minor cycle duration is defined by specifying the number of clock counts per minor cycle and the number of microseconds per clock count.

A major frame is one pass through all of the minor cycles with which a frequency-based scheduler is configured. The duration of a major frame is determined by multiplying the duration of a minor cycle by the number of minor cycles per major frame.

If, for example, you configure a scheduler with 100 minor cycles per major frame and you use as the timing source a real-time clock with a clock count of 10,000 and a clock count duration of one microsecond, each minor cycle has a duration of 10,000 microseconds, or 0.01 second, and each frame a duration of one second.

End-of-cycle scheduling is triggered when the last process scheduled during the current minor cycle of the current major frame completes its processing.

How Are Processes Scheduled?

A process is scheduled to run at a certain frequency by specifying the first minor cycle in which the process is to be awakened in each major frame (called the *starting base cycle*) and the frequency with which it is to be awakened (called the *period*).

If, for example, you schedule Process-1 with a starting base cycle of zero and a period of two, the process will be awakened once every two minor cycles, starting with the first minor cycle in the frame.

If you schedule Process-2 with a starting base cycle of one and a period of four, that process will be awakened once every four minor cycles, starting with the second minor cycle in the frame.

If you then schedule Process-3 with a starting base cycle of two and a period of two, that process will be awakened once every two minor cycles, starting with the third minor cycle in the frame.

On a scheduler configured with 100 minor cycles per major frame, these processes are awakened as illustrated in Table 2-1.

Table 2-1. Process Scheduling Example 1

Minor Cycle	Processes Awakened
0	Process-1
1	Process-2
2	Process-1, Process-3
3	
4	Process-1, Process-3
5	Process-2
.	
.	
.	
97	Process-2
98	Process-1, Process-3
99	

The maximum frequency that a process can be scheduled is once per minor cycle (a period of one); the minimum frequency is once per major frame (in the case of the example, a period of 100).

A process runs until it calls an FBS library routine that causes it to sleep. FBS wakes those sleeping processes that are scheduled to be awakened in the current minor cycle of the current major frame and repeats the process for each minor cycle in the current frame. It continues to repeat the entire process on every major frame until the scheduler is disabled.

To extend the previous example, we can use the three processes from the previous example:

- Process-1 - starting base cycle=0, period=2
- Process-2 - starting base=1, period=4,
- Process-3 - starting base=2, period=2

If we also configure a scheduler with 100 minor cycles per major frame, a minor cycle duration of 10,000 microseconds (0.01 second), and a major frame duration of one second, the three processes are scheduled as illustrated in Table 2-2.

Table 2-2. Process Scheduling Example 2

Major Frame	Time (sec.)	Minor Cycle	Processes Awakened
0	0	0	Process-1
	0.01	1	Process-2
	0.02	2	Process-1, Process-3
	...		
	0.97	97	Process-2
	0.98	98	Process-1, Process-3
	0.99	99	
1	1.00	0	Process-1
	1.01	1	Process-2
	1.02	2	Process-1, Process-3
	...		
	1.97	97	Process-2
	1.98	98	Process-1, Process-3
	1.99	99	
	...		
n	*n*.00	0	Process-1
	n.01	1	Process-2
	n.02	2	Process-1, Process-3
	...		
	n.97	97	Process-2
	n.98	98	Process-1, Process-3
	n.99	99	

As illustrated in Table 2-2, when the current major frame is zero and the current minor cycle is zero, the scheduler wakes Process-1. After 0.01 second, it wakes Process-2; after 0.02 second, it wakes Process-1 and Process-3; and so on.

At one second, when the current major frame becomes one, the current minor cycle becomes zero again, and the scheduler wakes Process-1. After 0.01 second, it wakes Process-2; after 0.02 second, it wakes Process-1 and Process-3; and so on. The scheduler continues repeating this process for as long as it is enabled.

Tolerating Frame Overruns

A *frame overrun* occurs when a scheduled process does not finish its processing before it is scheduled to run again.

Frame overruns are classified in two categories:

hard overruns	catastrophic failures of the scheduled process
soft overruns	catastrophic failures only if the process reaches its limit on the number of soft overruns tolerated. Each scheduled process has a soft overrun limit, defaulting to 0. Letting a process survive a reasonable number of soft overruns makes the system more flexible and efficient. Some soft overruns result from random, unpredictable, or external events unlikely to recur. Other soft overruns result from only minor frame overruns. Soft overruns give the scheduled process a chance to recover from a frame overrun and return to synchronization.

Note that there are no overruns associated with end-of-cycle scheduling. By definition, the cycle ends when all processes scheduled are blocked in the **`fbswait(3)`** call.

RedHawk Linux counts both soft and hard overruns for each scheduled process, but only hard overruns for each scheduler. Other processes can get these counts by querying the scheduled process or scheduler.

When scheduling a process, it is possible to specify that the scheduler is to be stopped by the kernel when that process running under it causes a hard overrun. Failing to specify that the scheduler should stop when a process causes a hard overrun allows the scheduler to continue to run, regardless of the number of hard overruns.

When scheduling a process, it is also possible to specify a consecutive soft overrun limit count that the process will tolerate and have processed as soft overruns by the kernel. Note that the default consecutive soft overrun limit is zero. With the limit set to zero, *all* overruns incurred by the process are treated as hard overruns (see below).

When a scheduled process overruns a frame and is not configured to be blocked in **`fbswait(3)`** when a frame interrupt occurs, the kernel decides whether to treat this overrun as a soft or hard overrun using the following steps:

- The consecutive soft overrun counter for the process is incremented.
- If the count does not reach or exceed the soft overrun limit for the process, the overrun is treated as a soft overrun. The process does not block the next time it calls **`fbswait`**; instead it returns immediately from the **`fbswait`** call with a status value of 2.
- Once the soft overrun limit reaches or exceeds the per-process limit, the overrun is treated as a hard overrun. The process blocks the next time it calls **`fbswait`**, and when the next normally scheduled frequency-based scheduler wakeup for that process occurs, the process returns out of the **`fbswait`** call returning a status value of 0. Note that a status of 2 is *not* returned in the hard overrun case.

Detecting Deadline Violations

A *deadline violation* occurs when a scheduled process exceeds an applied deadline time before it calls the FBS library routine that causes it to sleep. When a deadline violation is detected, the scheduler will be halted if the applied deadline specifies that the FBS should halt on deadline violation.

A deadline may be optionally applied to each process scheduled on the FBS. In addition, each deadline may be configured to halt the scheduler upon detection, or leave the scheduler running. Deadlines are detected when the scheduled process calls an FBS library routine that causes it to sleep, at the end of its processing. For earlier detection of deadline violations, another FBS library routine provides an explicit test for deadline violation.

A deadline may be configured to halt the FBS upon detection of a deadline violation. This may be used to debug an unstable application, or to avoid catastrophic failure when interfacing with external hardware, for example.

FBS deadlines may be removed from a process using a *deadline kind* of DEADLINE_CLEAR, or may be set using a deadline kind of DEADLINE_WALL_TIME. A CLOCK_MONOTONIC timing source is used to measure execution time when a DEADLINE_WALL_TIME deadline applies to the process. At this time, no other clock types are supported.

Each deadline time must be measured from an appropriate *deadline origin*. FBS supports two different deadline origins, depending on your timing needs:

- A *cycle-relative deadline* is used when a task needs to be complete by a fixed time within a minor cycle of a major frame. If, for example, you configure a scheduler with 100 minor cycles per major frame and each minor cycle has a duration of 10 milliseconds, and your process is scheduled to run starting in cycle 2 with a period of 10, then it will run once every 100 milliseconds, beginning with cycle 2. If the data it produces must be ready to be written to a hardware device every 100 milliseconds beginning with cycle 8, then the process must complete within 60 milliseconds. By setting a 60 millisecond cycle-relative deadline, your program could detect and potentially stop the FBS if the process ever takes longer than 60 milliseconds to complete its work.
- A *task-relative deadline* is used when a task needs to complete its work within a fixed duration of time once it begins execution. For example, if Process-1 and Process-2 are scheduled in the same cycle, with Process-1 at a higher real-time priority, then Process-1 will be scheduled to run prior to Process-2, and Process-2 will run as soon as Process-1 sleeps. If you wish to detect an occasionally long execution time for Process-2 that you know should never take more than 12 milliseconds to complete, you can use a 12 millisecond task-relative deadline to halt the FBS if Process-2 ever takes too long.

In addition, the NightSim GUI supports another form of deadline, measured in cycles, instead of units of time. A *cycle-count deadline* may be specified when a task needs to complete before the start of a following cycle, or before a proportional point within the cycle (for example, 75%) which will vary with the frequency of the real-time clock. See the *NightSim RT User's Guide* for more information.

NOTE

Scheduling with deadlines is supported in RedHawk version 5.2.1 and later.

User Interface

FBS is accessed using one of the following:

- the real-time command processor–**`rtcp`**. This program acts as a command interpreter for FBS, allowing you to perform key FBS operations by entering commands from the keyboard or invoking a script. **`rtcp`** is explained in Chapter 5, including an illustration showing the sequence in which you might invoke the commands.
- a set of library routines. This interface enables you to perform the entire range of functions associated with the scheduler from application programs written in C or FORTRAN. The C library routines are explained in Chapter 6; the FORTRAN library routines are explained in Chapter 7. Included are illustrations showing the sequence in which you might call the routines.
- NightSim. This is a Graphical User Interface (GUI) to the entire range of FBS functions. NightSim is explained in the *NightSim RT User's Guide.*

Debugging Frequency-Based Scheduler Processes

Debugging processes that have been scheduled on a frequency-based scheduler can be done using NightView a general-purpose, source-level debugger.

To be able to debug a C executable program, you must compile the source program specifying the **`-g`** option.

The NightView commands you can use to debug frequency-based scheduled processes are:

`attach` attach to a running process. This command allows you to debug a process that is already running

`detach` detach from an attached process. This command allows you to release an attached process from the control of the debugger.

To use NightView to debug a frequency-based scheduled process, you must supply the process ID (PID). You can easily obtain the PID for a frequency-based scheduled process by using the **`ps(1)`** command. You can obtain the PID for a selected process name by using the C library routine **`nametopid(3)`**. Use of each of these routines is explained in the corresponding man pages.

For NightView to attach to a running process, the debugger's effective user and group ID must match the effective user and group ID of the process controlled by the debugger.

For additional information on the procedures for using the **`attach`** and **`detach`** commands, see the *NightView RT User's Guide*.

3
Timing Sources

This chapter contains the procedures for utilizing various timing sources for frequency-based scheduling.

Overview

The Real-Time Clock and Interrupt Module (RCIM) provides real-time clocks and edge-triggered interrupts suitable as a timing source for a frequency-based scheduler. Each RCIM provides up to eight real-time clock timers (RTCs) and up to twelve edge-triggered interrupts (ETIs). These can be used locally or distributed across systems connected in an RCIM chain. A user-defined device can also be used as a timing source.

Before running the scheduler, the timing source is selected by "attaching" it to the scheduler. If a real-time clock is used, you can define the duration of a minor cycle by setting the count and the resolution values.

The sections that follow provide information about the available timing sources.

Using a Real-Time Clock

This section provides information for using an RCIM real-time clock as the timing source for a frequency-based scheduler. Discussions include:

- an overview of the real-time clock device
- a description of the user-interface to the device
- general procedures for using the device

Understanding the Real-Time Clock Device

The real-time clock (RTC) device is designed for a variety of timing and frequency control functions. It provides a range of clock count values and a set of resolutions that, taken together, produce many different timing intervals––a feature that makes it particularly appropriate for frequency-based scheduling.

The Real-Time Clock and Interrupt Module (RCIM) provides up to eight real-time clocks (0-7) on each system. When multiple systems are connected using an RCIM chain, all RTCs can be designated to be distributed; that is, interrupts are sent to all connected systems. A distributed RTC may be located on any system within the RCIM chain.

Each RTC is referenced through its own character special device file:

/dev/rcim/rtc*N*

For more information, refer to the **rcim(4)** man page or the *Real-Time Clock and Interrupt (RCIM) User's Guide*.

Understanding the User Interface

When using a real-time clock as a timing source for a frequency-based scheduler, any of the interfaces to FBS can be used to attach the clock to or detach the clock from the scheduler, or to set, start and stop the clock. The actions to be performed and the **rtcp** commands and C and FORTRAN library routines that are used are given in the section "General Procedures for Using a Real-Time Clock" below. Using NightSim for selecting and operating a real-time clock for frequency-based scheduling is described in the *NightSim RT User's Guide*.

It is recommended that you use **rtcp** or the routines contained in the C library as your interface to the real-time clock. The **rtcp** command is the easiest to use because it isolates you from most of the initialization tasks.

Real-time clocks can be controlled directly by using the system calls **open(2)**, **close(2)**, and **ioctl(2)**. Note that this device does not support **read(2)** and **write(2)** system calls. See the **rcim_rtc(4)** man page for details.

General Procedures for Using a Real-Time Clock

Whether you elect to use **rtcp** or the routines contained in the libraries as your interface to the real-time clock, the general procedures for using a real-time clock for frequency-based scheduling are the same:

STEP 1: Attach a real-time clock to a frequency-based scheduler.

STEP 2: Establish the duration of a minor cycle by specifying the clock count value and the resolution per clock count.

STEP 3: Start the real-time clock counting.

STEP 4: Start the simulation.

STEP 5: Stop the simulation.

STEP 6: Stop the real-time clock counting.

STEP 7: Detach the real-time clock.

The **rtcp** commands and C and FORTRAN routines that correspond to each step are presented in Table 3-1.

Table 3-1. rtcp Commands and Routines for using RTCs

Step	`rtcp` Command	C Routine	FORTRAN Routine
1	ats	fbsattach	fbsattach
2	stc	fbssetrtc	fbssetrtc
3	rc	fbsrunrtc	fbsrunrtc
4	start	fbsintrpt	fbsintrpt
5	stop	fbsintrpt	fbsintrpt
6	sc	fbsrunrtc	fbsrunrtc
7	dts	fbsdetach	fbsdetach

Refer to Chapter 5 for descriptions of the **`rtcp`** commands, Chapter 6 for explanations of the routines included in the C library and Chapter 7 for the FORTRAN routines.

Using an Edge-Triggered Interrupt

This section provides information for using an edge-triggered interrupt as the timing source for a frequency-based scheduler. Discussions include:

- an overview of the edge-triggered interrupt device
- a description of the user-interface to the device

Understanding the Edge-Triggered Interrupt

The edge-triggered interrupt (ETI) device is a software interface to the external interrupt lines provided by the RCIM. External interrupt lines provide a means for the computer system to detect an external interrupt coming into the system from any user device that generates a signal pulse. This signal pulse can be used as the timing source for a frequency-based scheduler.

There are up to twelve edge-triggered interrupts (0-11) per RCIM. When multiple systems are connected via an RCIM chain, all ETIs may be designated to be distributed; that is, interrupts are sent to all connected systems. A distributed ETI may be located on any system within the RCIM chain.

Each ETI is referenced through its own character special device file:

`/dev/rcim/eti`*N*

For more information, refer to the **`rcim(4)`** man page or the *Real-Time Clock and Interrupt (RCIM) User's Guide*.

Understanding the User Interface

When using an edge-triggered interrupt as a timing source for a frequency-based scheduler, any of the interfaces to FBS can be used to attach the interrupt to or detach it from the scheduler. Ensure that it is already generating interrupts when you start the simulation.

The **rtcp** commands and C and FORTRAN library routines that are used to attach and detach the interrupt from the scheduler are given in Table 3-2.

Table 3-2. rtcp Commands and Routines for using ETIs

rtcp Command	C Routine	FORTRAN Routine
ats	fbsattach	fbsattach
dts	fbsdetach	fbsdetach

Refer to Chapter 5 for descriptions of the **rtcp** commands, Chapter 6 for explanations of the routines included in the C library and Chapter 7 for the FORTRAN routines. Using NightSim for selecting and operating a real-time clock for frequency-based scheduling is described in the *NightSim RT User's Guide*.

Edge-triggered interrupts can be controlled directly by using the system calls **open(2)**, **close(2)**, and **ioctl(2)**. Note that this device does not support the **read(2)**, **write(2)** and **mmap(2)** system calls. See the **rcim_eti(4)** man page for details.

Using a User-Supplied Timing Device

You may wish to use your own device as the timing source for a frequency-based scheduler. To use your own device, you must ensure the following:

- Your device driver supports the IOCTLVECNUM **ioctl** call
- Your device driver supports the IOCTLKEEPALIVE ioctl call
- Your device generates a series of interrupts

FBS makes the IOCTLVECNUM **ioctl** call to provide the timing source driver with an IRQ that is associated with the FBS interrupt handler routine. The timing source driver must invoke this interrupt handler by calling invoke_irq() from its interrupt handler.

The **ioctl** call requires the following specifications:

```
#include <linux/ioctl/vecnum.h>

ioctl (fd, IOCTLVECNUM, arg);
```

fd the file descriptor for the device

IOCTLVECNUM the command to pass the IRQ to the timing source driver

arg — an IRQ whose interrupt handler(s) is to be invoked in addition to the interrupt handler naturally attached to this device. A value of zero (0) disables a previous association.

To prevent the associated device from shutting down when the final close to the device is made, the following is also provided:

```
#include <linux/ioctl/keepalive.h>

ioctl (fd, IOCTLKEEPALIVE, arg);
```

arg — when nonzero, enables the keepalive state enabling the device to continue running, if possible, after final close of the device. When zero (0), disables the keepalive state.

At the bottom of the interrupt handler, the following is done to invoke the handler(s) for the given IRQ:

```
#include <linux/interrupt.h>

if (irq)
    invoke_irq (irq, (struct pt_regs *)0);
```

Examples of Linux drivers implementing this interface can be found in the following RedHawk kernel source directory:

*<kernel-sourcedir>***/drivers/char/rcim**

4
Using the Performance Monitor

This chapter contains a description of the Performance Monitor and its capabilities. It explains the values that are monitored and how to enable monitoring of system idle time and unscheduled processes. Suggestions for optimizing the performance of a simulation are given and user interface methods are introduced.

What Is the Performance Monitor?

The Performance Monitor (PM) allows you to monitor use of a CPU by processes that are scheduled using a frequency-based scheduler.

The Performance Monitor provides the ability to:

- obtain PM values by process or processor
- control all performance monitoring features from one processor (that is, enable performance monitoring for any processor)
- start and stop performance monitoring by process or processor
- clear PM values by process or processor
- set the timing mode to include or exclude interrupt times

Note that high-resolution process accounting must be configured in the system in order to use the Performance Monitor. See the "Configuration" section in Chapter 1 for more information.

What Values Are Monitored?

The performance monitor keeps track of the time that a process spends running from the time that it is awakened by a frequency-based scheduler until it calls **`fbswait(3)`**. Time is measured in microseconds.

One instance of a process being awakened by a scheduler is referred to as an *iteration* or a *cycle*. PM values for frequency-based scheduled processes are reported both in terms of cycles or iterations, and in terms of *major frames*. They reflect what has happened since the last time PM values were cleared and performance monitoring was enabled.

When the PM timing mode is set to include interrupt time, the user and system times of a process total the elapsed time that accrues when the process is running; time spent servicing interrupts is added to the system time of the process. When the timing mode is set to exclude interrupt time, a process' user and system times total the time that accrues when the process is the currently running process, excluding time spent servicing interrupts. Whether the timing mode is set to include or exclude interrupt time, context switch time is always included in the new process' system time.

Whether performance monitoring is enabled for a single frequency-based scheduled process or for all frequency-based scheduled processes on a processor, the following types of values are maintained for each process:

Total iterations, cycles	The number of times the process has been awakened by the scheduler
Last time	The amount of time the process has spent running from the last time it has been awakened by the scheduler until it has called **`fbswait(3)`**
Total time	The total amount of time the process has spent running in all cycles
Minimum cycle time	The least amount of time the process has spent running in a cycle
Minimum cycle cycle	The number of the minor cycle in which the minimum cycle time has occurred
Minimum cycle frame	The number of the major frame in which the minimum cycle time has occurred
Maximum cycle time	The greatest amount of time the process has spent running in a cycle
Maximum cycle cycle	The number of the minor cycle in which the maximum cycle time has occurred
Maximum cycle frame	The number of the major frame in which the maximum cycle time has occurred
Minimum frame time	The least amount of time the process has spent running during a major frame
Minimum frame frame	The number of the major frame in which the minimum frame time has occurred
Maximum frame time	The greatest amount of time the process has spent running during a major frame
Maximum frame frame	The number of the major frame in which the maximum frame time has occurred
Number of overruns	The number of times the process has caused a frame overrun

Monitoring Idle Time

The Performance Monitor can monitor a processor's *idle time*. Idle time refers to the time the CPU is not busy.

By monitoring a processor's idle time, you can determine the amount of CPU time available for allocation to additional processes.

You monitor a processor's idle time by adding the process **/idle** to a frequency-based scheduler and scheduling it on the desired processor. You can monitor idle time for a number of different processors by adding **/idle** to a selected frequency-based scheduler more than once and scheduling it on a different processor each time. You can also add **/idle** to more than one frequency-based scheduler. It is important to note, however, that you can schedule **/idle** on a particular processor only once.

To add **/idle** to a frequency-based scheduler, do one of the following:

- execute the **rtcp** command **sp**
- make a call to **sched_pgmadd(3)** from a C program
- make a call to **schedpgmadd(3f)** from a FORTRAN program

A description of the **sp** command is provided on page 5-32. A description of the **sched_pgmadd(3)** routine can be found on page 6-51 and on the man page. A description of the **schedpgmadd(3f)** routine can be found on page 7-55 and on the man page.

When you add **/idle** to a frequency-based scheduler, the only parameter you must specify is the CPU. The default scheduling priority for **/idle** is zero. The starting base cycle is zero, and the period is one. The **/idle** process is scheduled every minor cycle, starting with the first minor cycle in each major frame.

NOTE

When using the **sched_pgmadd(3)** subroutine to add **/idle** to a scheduler, only one bit can be set in the bit mask that specifies the processor. To add **/idle** to a scheduler and schedule it on more than one processor, you must call the subroutine repeatedly, specifying a different processor on each call.

After **/idle** is scheduled, a unique frequency-based scheduler identifier is returned. Use this number to identify **/idle** when performing tasks related to FBS or PM.

You obtain scheduling information for **/idle** in the same way you obtain it for other frequency-based scheduled processes, by doing one of the following:

- execute the **rtcp** command **vp**
- make a call to **sched_fbsqry(3)** or **sched_pgmqry(3)** from a C program
- make a call to **schedfbsqry(3f)** or **schedpgmqry(3f)** from a FORTRAN program

A description of the **vp** command is provided on page 5-35. A description of the **sched_fbsqry** routine is provided on page 6-36, **sched_pgmqry** is on page 6-61, **schedfbsqry(3f)** is on page 7-52, **schedpgmqry(3f)** is on page 7-59 and descriptions are on the respective man pages.

If you enable performance monitoring for the processor(s) on which **/idle** is scheduled or for the process itself, you can obtain all PM values.

Monitoring Unscheduled Processes

The Performance Monitor provides the additional capability of monitoring the performance of *unscheduled processes*. Unscheduled processes are those that are not awakened by the scheduler and do not call **fbswait**.

To be able to obtain PM values for such processes, add them to a frequency-based scheduler and specify a starting base cycle of zero and a period of zero. Other scheduling parameters you must specify include the process scheduling priority and the CPU on which it is to be scheduled. Optionally specify an octal value to be passed to a process that is scheduled on a frequency-based scheduler. The halt on overrun flag does not apply to an unscheduled process.

You add unscheduled processes to a frequency-based scheduler by doing one of the following:

- executing the **rtcp** command **sp**
- making a call to **sched_pgmadd(3)** from a C program
- making a call to **schedpgmadd(3f)** from a FORTRAN program

An explanation of the **sp** command is provided on page 5-32. A description of the **sched_pgmadd** routine is located on page 6-51, **schedpgmadd** is on page 7-55 and descriptions are on the respective man pages.

After a process is scheduled, the unique frequency-based scheduler identifier is returned. You can subsequently use this number to identify the process when you are performing tasks related to FBS or PM.

You obtain scheduling information for unscheduled processes in the same way you obtain it for other frequency-based scheduled processes by doing one of the following:

- executing the **rtcp** command **vp**
- making a call to **sched_pgmqry(3)** from a C program
- making a call to **schedpgmqry(3f)** from a FORTRAN program

An explanation of the **vp** command is provided on page 5-35. A description of the **sched_pgmqry** routine can be found on page 6-61, **schedpgmqry** is on page 7-59 and descriptions are on the respective man pages.

PM values maintained for unscheduled processes include:

- last time
- total time
- minimum frame time and the number of the frame in which it occurred
- maximum frame time and the number of the frame in which it occurred

You obtain these values by enabling performance monitoring for the processor(s) on which the processes have been scheduled or on the individual processes.

User Interface

PM is accessed using one of the following:

- the real-time command processor–**`rtcp`**. This program acts as a command interpreter for PM, allowing you to perform key performance monitoring operations by entering commands from the keyboard or invoking a script. **`rtcp`** is explained in Chapter 5, including an illustration showing the sequence in which you might invoke the commands.
- a set of library routines. This interface enables you to perform the entire range of functions associated with PM from application programs written in C and FORTRAN. The library routines are explained in Chapter 6 and Chapter 7, respectively, including an illustration showing the sequence in which you might call the routines.
- NightSim. This is a Graphical User Interface (GUI) to the entire range of PM functions. NightSim is explained in the *NightSim RT User's Guide.*

Optimizing the Performance of a Simulation

One of the benefits of using multiprocessor systems for real-time processing is that you can optimize the performance of a simulation by distributing processes among several processors.

Using FBS to schedule the programs that make up a simulation allows you to use the Performance Monitor to determine the extent to which the processes are using a CPU and to find out whether or not they are running at the specified frequency.

A program is scheduled on a processor when it is added to a frequency-based scheduler. The processor on which it is scheduled is determined by the CPU bias (affinity) specified when it is added to the scheduler. After programs have been scheduled, enable performance monitoring on one or more processes or processors and run the simulation.

By examining the PM values that are maintained for each frequency-based scheduled process, you can determine:

- the processors to which the processes have been assigned
- the amount of time the processes have spent running
- the processes that have not run at their assigned frequency

If you find that a process is not running at its assigned frequency, examine the frequency, the amount of CPU time being used by the other processes, and the CPU biases for all processes. Deadlines may also be used to pinpoint instances when a process runs too long.

Note that if the CPU bias of a process identifies more than one processor, you cannot determine how much time the process has spent on a particular CPU specified in the bit mask because of dynamic load balancing. To avoid dynamic load balancing, specify only one processor in the bit mask. By using performance monitoring, you can then tell how much time a process has spent on its assigned CPU. You can redistribute processes as necessary.

You can also enable performance monitoring for a processor's idle time. The procedures for doing so are explained in the section "Monitoring Idle Time" in this chapter. By examining the amount of idle time on each processor, you will be able to identify the processors that have the lightest load and calculate the additional amount of CPU time that can be used for scheduling real-time processes.

You can determine the processor assignments that are optimal for your simulation by analyzing the PM values for frequency-based scheduled processes and for idle time on selected processors. Redistribute your frequency-based scheduled processes as necessary by changing their CPU biases.

It is important to note that in order to do so, you must first remove the process from the scheduler on which it has been scheduled and then add it to a scheduler. For an overview of the Frequency-Based Scheduler and the interfaces that accommodate its use, refer to Chapter 2.

The performance of a task can also be influenced by the way in which the system's CPUs are configured. See the "Real-Time Performance" chapter of the *RedHawk Linux User's Guide* for more about CPU shielding and hyper-threading.

5
Using rtcp

This chapter describes the real-time command processor, **rtcp**. It describes the execution modes, the sequence of commands to be used for frequency-based scheduling and performance monitoring, how to get help, and a complete description of each command.

What Is the Real-Time Command Processor?

The real-time command processor (**rtcp**) is a program that acts as a command interpreter for the Frequency-Based Scheduler and the Performance Monitor. **rtcp** allows you to perform key operations by entering commands from the keyboard or invoking a script from the shell command line. It reads the commands and interprets them as requests to execute the related services.

There are two modes of execution: direct and interactive. These modes are described in detail on page 5-2.

rtcp also has a help facility that makes it possible for you to obtain online information about commands and arguments. Procedures for using the help facility are explained on page 5-5.

rtcp and the Frequency-Based Scheduler

rtcp commands associated with FBS enable you to perform such key operations as:

- configuring a scheduler
- scheduling programs
- saving a scheduler configuration
- setting up a timing source
- running a simulation
- querying status

See Table 5-1 for a summary of the **rtcp** commands associated with the Frequency-Based Scheduler. An overview of the Frequency-Based Scheduler is provided in Chapter 2. It is recommended that you read this chapter and the chapter on Performance Monitor before using **rtcp**.

rtcp and the Performance Monitor

rtcp commands associated with the Performance Monitor enable you to perform such key operations as:

- clearing performance monitor values
- starting and stopping performance monitoring
- setting the timing mode
- querying values

See Table 5-1 for a summary of the **rtcp** commands associated with the Performance Monitor. An overview of the Performance Monitor is provided in Chapter 4. It is recommended that you read this chapter and the chapter on the Frequency-Based Scheduler before using **rtcp**.

Execution Modes

The real-time command processor provides two modes for executing commands:

direct mode	enables you to invoke real-time command processor commands from the shell command line
interactive mode	enables you to invoke the real-time command processor, itself, from the shell command line and then to enter the desired commands from within the command processor

Using Direct Mode

Use direct mode in one of the following ways. Each method is described below.

- Invoke the real-time command processor with a command name and its arguments at the system command prompt.
- Invoke the real-time command processor at the system command prompt, redirecting the standard input to come from a file instead of the terminal keyboard.
- Invoke a script at the system command prompt.

Invoking rtcp with a Command Name and Arguments

Use the following format when invoking **rtcp** with a command and arguments:

```
$ rtcp command [–option [argument] ] [–option [argument] ]  . . .
```

Note that you are allowed to enter only one command on the command line at a time. If you need more than one line to enter a command and its arguments, enter a backslash (\) at the end of the line to continue on the next line.

Invoking rtcp with Commands Redirected from a Script File

This method requires that you create a file that contains the **rtcp** commands you wish to execute, and the file is used as standard input to **rtcp**. Create the file using a text editor of your choice or by executing the **svs** (Save Scheduler Configuration) command. The **svs** command is explained on page 5-17.

If you use a text editor to create the file, enter each command on a separate line. You may use either of the following formats:

```
rtcp command [–option [argument] ] [–option [argument] ]  . . .
```

or

```
command [–option [argument] ] [–option [argument] ]  . . .
```

If you need more than one line to enter a command and its arguments, enter a backslash (\) at the end of the line to continue on the next line.

After you have created the file, invoke the real-time command processor, and redirect the standard input to come from the file:

```
$ rtcp < rtcp_input_file
```

Invoking a Script that Calls rtcp

This method requires that you create an executable file that contains the **rtcp** commands you wish to execute. The first line in the file can contain a command or the following text:

```
#!program_name
```

where *program_name* specifies the name of the file that contains the shell to be invoked; for example **/bin/bash**.

Each command must be entered on a separate line according to the following format:

```
rtcp command [–option [argument] ] [–option [argument] ]  . . .
```

If you need more than one line to enter a command and its arguments, enter a backslash (\) at the end of the line to continue on the next line.

After you have created the file, make it an executable file by using the **chmod(1)** command, then invoke it from the command line as you would any other command:

```
$ chmod 755 rtcp_script
$ rtcp_script
```

For information on use of the **chmod(1)** command, see the corresponding man page. See Appendix A for an example of a real-time command processor script.

Using Interactive Mode

Use the following procedure to interactively invoke the real-time command processor:

1. Specify the **rtcp** command at the system command prompt:

   ```
   $ rtcp
   ```

   The real-time command processor prompt is then displayed:

   ```
   rtcp>
   ```

2. At the prompt, type real-time processor commands by using the following format:

   rtcp>*command* [–*option* [*argument*]] [–*option* [*argument*]] ...

   If you need more than one line to enter a command and its arguments, enter a backslash (\) to cause the line to be continued.

   In most instances, if the command is successfully executed, a message is displayed and, where applicable, configuration data, scheduling information, or performance monitor values are displayed.

   Messages and data associated with the commands are included in the reference information that begins on page 5-7. If an error occurs, a message indicating the nature of the error is displayed. Error messages are listed and described in Appendix B.

3. To exit the command processor and return to the shell, type the following:

   rtcp>**ex**

   The system command prompt is again displayed.

Getting Help

Two methods of accessing information about **rtcp** online are available:

- **rtcp(1)** man page
- **rtcp** help facility

Access the **rtcp** help facility using the **he** command. The information provided includes:

- a list and brief description of all **rtcp** processor commands
- a description and format of a particular command
- a list and description of all command arguments

To display a list of all commands, enter the **he** command as follows:

```
he
```

Commands are displayed as illustrated in Screen 5-1.

Screen 5-1. rtcp Command Display

```
            rtcp commands

ats  - attach timing source to FBS            chs - modify FBS access permissions
cs   - configure FBS                          dts- detach timing source from FBS
rms  - remove FBS                             svs- save FBS configuration to a file
vc   - view current frame/cycle count         vs - view FBS configuration
ls   - list FBSs

rc   - run real-time clock                    sc - stop real-time clock
stc  - set real-time clock values             gtc- get real-time clock values

start  -  start FBS                           resume  -  resume FBS
stop   -  stop FBS

rmp  - remove a process on a FBS              rsp- reschedule a process on a FBS
sp   - schedule a process on a FBS            vp - view scheduled processes on FBS

cpm  - clear performance monitor tables       pm - start/stop performance monitor
vcm  - view/modify PM timing mode             vpm- view performance

he   - help                                   ex - exit rtcp
```

To display a description of a particular command, enter **he** with the name of the command as an argument. For example, to display a description of the **ats** command, specify:

```
he ats
```

A description of the **ats** command and the format for entering the command are displayed as follows:

```
Attach timing source to a FBS

rtcp ats -s scheduler -d device | -e
```

To display a list of command arguments, enter **he** with **option** as the argument:

```
he option
```

The first screen of **rtcp** arguments is displayed as illustrated in Screen 5-2.

Screen 5-2. First Screen of rtcp Arguments

```
    rtcp parameters

-a                  remove program from FBS and terminate
-b {F|R|O}          scheduling policy
-c cpu_bias         cpu bias (* = all CPUs) (default = current CPU)
-d name             devicename or filename
-e                  EOC flag
-f frequency        number of minor cycles to next wakeup (default = 1)
-h {halt|nohalt}    halt FBS on deadline violation (default = nohalt)
-i fpid             process fpid number (default = -1)
-m start_cycle      1st minor cycle to wakeup (default = 0)
-n proc_name        process name
-o {halt|nohalt}    halt FBS on overrun flag (default = nohalt)
-p priority         process priority
-r {cycle|task}     deadline origin flag (default = cycle)
-s scheduler        FBS scheduler key
-t {in|ex}          include or exclude interrupt time in pm monitor
-v parameter        process initiation parameter
-x {av|mi|ma|al}    performance monitor display option (default = average)

Enter 'he op2' for more parameters
```

To display the second screen of **rtcp** arguments, enter **he** with **op2** as the argument:

```
he op2
```

The second screen of **rtcp** arguments is displayed as illustrated in Screen 5-3.

Screen 5-3. Second Screen of rtcp Arguments

```
        rtcp parameters

-C cycles/frame       number of minor cycles per major frame
-D duration           clock tick duration (default = 10us)
-G gid                effective group ID for FBS (default = current user)
-I permissions        permissions for FBS in octal (default = 0600)
-L soft_limit         soft overrun limit (default = 0)
-M progs/cycle        maximum number of processes per minor cycle
-N progs/fbs          maximum number of processes per FBS
-O clock_ticks        number of clock ticks per minor cycle
-P {ON|OFF}           enable/disable performance monitor (default = OFF)
-R {-1 | 0 | 1}       reset process flag (default = 0)
-S delay              time to delay, in seconds
-T deadline           deadline microseconds (default = Clear)
-U uid                effective user ID for FBS (default = current user)
```

rtcp Commands

rtcp Command Summary

rtcp commands are listed in Table 5-1. Complete information about each command as well as a command sequence flowchart are provided below.

Table 5-1. rtcp Commands

Command	Page	Description
Frequency-Based Scheduler:		
ats	5-11	Attach timing source to a frequency-based scheduler
chs	5-12	Change permissions for a frequency-based scheduler
cs	5-13	Configure a frequency-based scheduler
dts	5-15	Detach timing source from a frequency-based scheduler
gtc	5-23	Get real-time clock values
ls	5-21	List all frequency-based schedulers configured on the system
rc	5-21	Start real-time clock

(contined on next page)

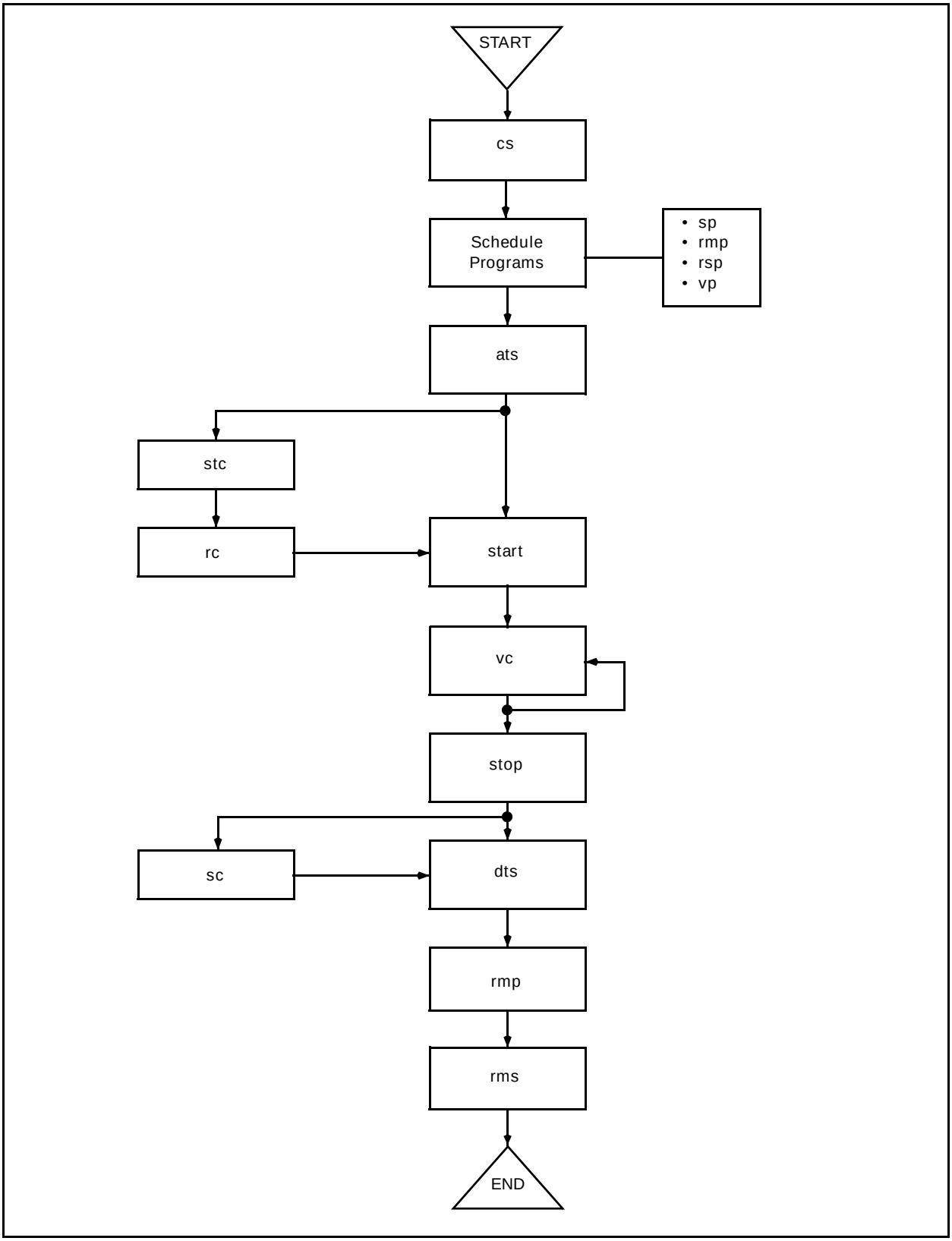
Table 5-1. rtcp Commands (Continued)

Command	Page	Description
Frequency-Based Scheduler (Continued):		
resume	5-25	Resume scheduling on a frequency-based scheduler
rmp	5-26	Remove a process from a frequency-based scheduler
rms	5-16	Remove a frequency-based scheduler
rsp	5-28	Reschedule a process
sc	5-22	Stop real-time clock
sp	5-32	Schedule a process on a frequency-based scheduler
start	5-24	Start scheduling on a frequency-based scheduler
stc	5-22	Set real-time clock values
stop	5-25	Stop scheduling on a frequency-based scheduler
svs	5-17	Save scheduler configuration
vc	5-18	View minor cycle/major frame count
vp	5-35	View processes on a frequency-based scheduler
vs	5-19	View scheduler configuration
Performance Monitor:		
cpm	5-38	Clear performance monitor values
pm	5-40	Start/stop performance monitoring
vcm	5-42	View or modify performance monitor timing mode
vpm	5-43	View performance monitor values
rtcp:		
ex	5-47	Exit real-time command processor
he	5-47	Display help information

Command Sequence

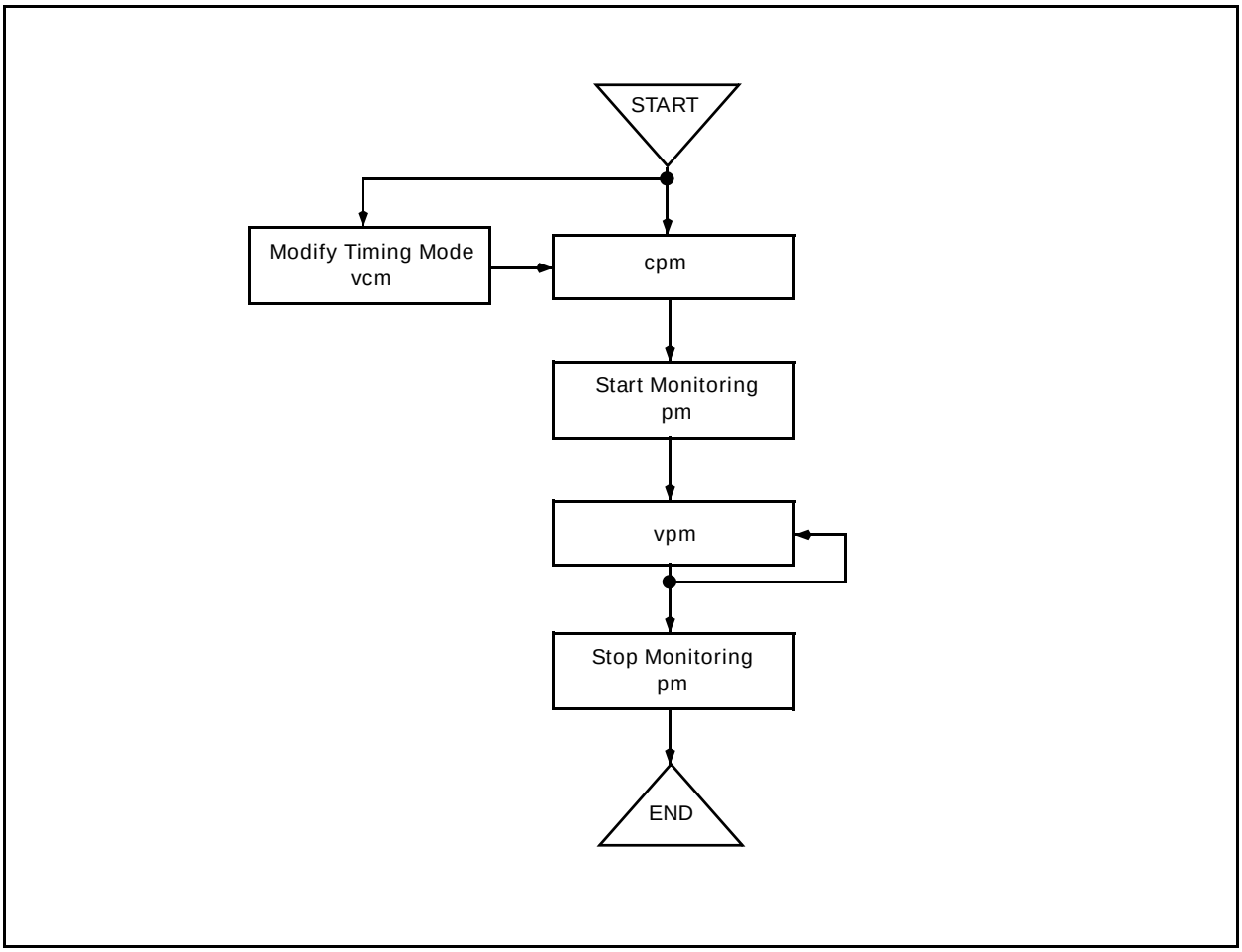
The order in which you might execute the **`rtcp`** commands associated with the Frequency-Based Scheduler is illustrated in Figure 5-1.

Figure 5-1. rtcp Command Sequence: Frequency-Based Scheduler



The order in which you might execute the commands associated with the Performance Monitor is illustrated in Figure 5-2.

Figure 5-2. rtcp Command Sequence: Performance Monitor



Using rtcp Commands

This section provides reference information for all of the commands supported by the real-time command processor. Commands are presented in the order in which they are described in the help facility. For each command, the following information is provided:

- a description of the command
- the format for entering the command
- detailed descriptions of each argument
- an example of the output from the command

ats – Attach Timing Source to a Frequency-Based Scheduler

The **ats** command attaches a timing source to a frequency-based scheduler or specifies end-of-cycle scheduling. In the latter case, scheduling is triggered when the last process scheduled during the current minor cycle completes its processing.

Synopsis

ats -s *scheduler* {**-d** *device* | **-e**}

Arguments

-s *scheduler*	the numeric key associated with the frequency-based scheduler for which the timing source is to be attached or end-of-cycle scheduling specified. The scheduler must previously have been configured. It can be any positive integer value.
-d *device*	the path name of the real-time clock or edge-triggered interrupt being used as the timing source for the specified scheduler. Refer to Chapter 3 for detailed information on the form associated with each type of device.
-e	specifies end-of-cycle scheduling. In this case, execution of the processes in the next minor cycle occurs when the last process scheduled to run in the current minor cycle finishes its processing for the cycle.

Display

The following message is displayed when the specified timing source is successfully attached to the scheduler or if end-of-cycle scheduling is successfully enabled:

```
Scheduler attached
```

chs – Change Permissions for a Frequency-Based Scheduler

The **chs** command changes the permissions assigned for a frequency-based scheduler. In order to do this, you must have an effective user ID equal to that of the creator/owner of the frequency-based scheduler.

Synopsis

chs -s *scheduler* **[-I** *permissions***] [-G** *gid***] [-U** *uid***]**

Arguments

-s *scheduler* — the numeric key associated with the frequency-based scheduler for which the permissions are to be changed. The scheduler must previously have been configured. See the section "cs – Configure a Frequency-Based Scheduler" on page 5-13 for information on configuring a frequency-based scheduler.

-I *permissions* — the permissions required for operations related to the specified scheduler.

The *permissions* argument specifies three octal digits—the first indicates permissions granted to the owner, the second those granted to the group, and the third those granted to other users.

The octal method for changing permissions associated with a scheduler is the same as that used for specifying *mode* with the **chmod** command (see the **chmod(1)** man page). The default, **600,** grants read and alter (write) permission to the owner only.

-G *gid* — the effective group ID of the selected frequency-based scheduler. The default effective group is the current user.

-U *uid* — the effective user ID of the selected frequency-based scheduler. The default is the current user.

Display

The following message is displayed when the permissions assigned to the scheduler are successfully changed:

```
Scheduler permissions changed
```

cs – Configure a Frequency-Based Scheduler

The **cs** command creates a frequency-based scheduler.

To create a scheduler, you must specify a key, which is a user-selected numeric identifier with which the scheduler will be associated. You must also define:

- the number of minor cycles that compose a major frame on the scheduler
- the maximum number of tasks that can be scheduled during a minor cycle
- the maximum number of tasks scheduled on a scheduler at one time

A frequency-based scheduler uses two types of permission that control users' ability to perform scheduler operations: read and alter (write). Read permission is required to perform query operations. Alter permission is required to:

- schedule, remove, and reschedule programs
- attach a timing source to and detach it from a scheduler
- start, stop, and resume scheduling

Permissions are assigned when you create the scheduler. They are specified in the same way in which permissions associated with files are assigned. See the **chmod(1)** man page for assistance in specifying permissions.

When you execute the **cs** command, a unique, positive frequency-based scheduler identifier and corresponding data structure are created for the specified key if *both* of the following conditions are met:

- the key is not already associated to a frequency-based scheduler identifier
- the number of frequency-based schedulers already configured is less than the maximum number of schedulers allowed on your system

The newly created frequency-based scheduler identifier is displayed on the terminal.

When you specify a key that is already associated with a frequency-based scheduler, the corresponding frequency-based scheduler identifier is displayed on your terminal screen if *all* of the following conditions are met:

- the number of minor cycles specified by the **-C** *cycles/frame* argument matches the number of minor cycles associated with the existing scheduler
- the maximum specified by the **-M** *progs/cycle* argument is less than or equal to the maximum number of processes per minor cycle associated with the existing scheduler
- the maximum specified by the **-N** *progs/fbs* argument is less than or equal to the maximum number of processes allowed on the existing scheduler

If these conditions are not met, an error message is displayed.

Synopsis

cs -s *scheduler* **-C** *cycles/frame* **-M** *progs/cycle* **-N** *progs/fbs* **[-I** *permissions***]** \
[-R *reset***]**

Arguments

-s *scheduler* — a key for the frequency-based scheduler that you wish to create. The key is a user-selected numeric identifier with which the scheduler will be associated. The value of *scheduler* can be any positive integer value.

-C *cycles/frame* — the number of minor cycles that compose a frame on the specified frequency-based scheduler.

-M *progs/cycle* — the maximum number of programs that can be scheduled to execute during one minor cycle.

-N *progs/fbs* — the maximum number of programs that can be scheduled on the specified scheduler at one time. This value must be less than or equal to the product obtained by multiplying the values specified for the *cycles/frame* and the *progs/cycle* arguments.

-I *permissions* — permissions required for operations related to the specified scheduler.

The *permissions* argument specifies three octal digits—the first indicates permissions granted to the owner, the second those granted to the group, and the third those granted to other users.

The octal method for changing permissions associated with a scheduler is the same as that used for specifying *mode* with the **chmod** command (see the **chmod(1)** man page). The default, **600,** grants read and alter (write) permission to the owner only.

-R *reset* — the manner in which processes currently scheduled on the specified scheduler are to be handled.

The value of the *reset* argument can be 1, -1, or 0.

Specifying 0 (the default) allows these processes to remain on the scheduler. Specifying 1 removes these processes from the scheduler but allows them to continue executing. Specifying -1 removes these processes from the scheduler and terminates them.

Display

The following message is displayed when the scheduler is successfully configured:

```
Scheduler 10 has FBS ID of 3
```

`Scheduler`

Indicates the user-specified key for the selected frequency-based scheduler. It is important to note that this value is required by most of the real-time command processor commands.

`FBS ID`

Indicates the unique positive integer value representing the identifier for the selected frequency-based scheduler.

dts – Detach Timing Source from a Frequency-Based Scheduler

The **`dts`** command detaches the timing source from a frequency-based scheduler or disables end-of-cycle scheduling. If the timing source is a real-time clock, it is recommended that you stop the clock prior to invoking this routine by making a call to **`sc`** (page 5-22).

Synopsis

`dts –s` *scheduler*

Arguments

`–s` *scheduler* — the numeric key associated with the frequency-based scheduler for which the timing source is to be detached or end-of-cycle scheduling disabled. The scheduler must previously have been configured. See the section "cs – Configure a Frequency-Based Scheduler" on page 5-13 for information on configuring a frequency-based scheduler.

Display

The following message is displayed when the timing source is successfully detached from the scheduler or end-of-cycle scheduling is successfully disabled:

```
Scheduler detached
```

rms – Remove a Frequency-Based Scheduler

The **rms** command removes a frequency-based scheduler. Prior to executing this command, you must ensure that the timing source for the scheduler has been detached or that end-of-cycle scheduling has been disabled (see "dts – Detach Timing Source from a Frequency-Based Scheduler" on page 5-15 for information on use of the **dts** command).

Note that to remove a scheduler, the calling process must have an effective user ID equal to that of the owner/creator of the frequency-based scheduler.

Synopsis

rms –s *scheduler* [**–a**]

Arguments

–s *scheduler*	the numeric key associated with the frequency-based scheduler that you wish to remove. The scheduler must previously have been configured. See the section "cs – Configure a Frequency-Based Scheduler" on page 5-13 for information on configuring a frequency-based scheduler.
–a	all processes currently scheduled on the specified scheduler are to be removed from the scheduler and terminated. If this option is not specified, all processes currently scheduled on the specified scheduler are removed but continue executing.

Display

The command displays the following message when the specified scheduler is successfully removed:

```
Scheduler removed
```

svs – Save Scheduler Configuration

The **svs** command stores configuration and scheduling data for a selected frequency-based scheduler in a file.

When you execute this task, the **rtcp** commands entered from the command line are saved to the specified output file. The file can then be used as an **rtcp** input file. The commands saved by **svs** are those used to configure a scheduler (**cs**), schedule programs on it (**sp**), attach a timing source to it (**ats**), and set the real-time clock (**stc**).

Synopsis

svs –s *scheduler* **–d** *output_file_name*

Arguments

–s *scheduler*	the numeric key associated with the frequency-based scheduler for which you wish to store configuration and scheduling data. The scheduler must previously have been configured. See the section "cs – Configure a Frequency-Based Scheduler" on page 5-13 for information on configuring a frequency-based scheduler.
–d *output_file_name*	a standard path name identifying the file in which you wish configuration data to be stored. It can be a full or relative path name of up to 1024 characters.

Display

No message is displayed if the scheduler configuration is successfully saved to a file.

vc – View Minor Cycle/Major Frame Count

The **vc** command displays the current minor cycle and major frame count values for a frequency-based scheduler. These values help determine the progress of a simulation.

Synopsis

vc -s *scheduler*

Arguments

-s *scheduler*	the numeric key associated with the frequency-based scheduler for which you wish to view the current cycle and frame counts. The scheduler must previously have been configured. See the section "cs – Configure a Frequency-Based Scheduler" on page 5-13 for information on configuring a frequency-based scheduler.

Display

When successfully executed, the command displays the following types of information:

```
Major frame = 65 Minor cycle = 25
```

`Major frame`

Indicates the number of the current major frame for the simulation running on the selected scheduler.

`Minor cycle`

Indicates the number of the current minor cycle for the simulation running on the selected scheduler.

vs – View Scheduler Configuration

The **vs** command allows you to view information related to a frequency-based scheduler. Viewable information includes:

- the key and identifier associated with the scheduler
- the number of minor cycles per major frame, the maximum number of programs per minor cycle, and the maximum number of programs per scheduler
- the user and group IDs of the owner and creator of the scheduler
- the permissions assigned for the scheduler
- the total number of overruns for all processes on the scheduler
- an indication of whether the scheduler is in the run or the stop state
- the CPUs that are active in the system and the CPUs for which performance monitoring has been enabled
- the path name of the device that has been attached to the scheduler

Synopsis

vs –s *scheduler*

Arguments

–s *scheduler*	the numeric key associated with the frequency-based scheduler for which you wish to view current information. The scheduler must previously have been configured. See the section "cs – Configure a Frequency-Based Scheduler" on page 5-13 for information on configuring a frequency-based scheduler.

Display

When successfully executed, the command displays configuration and status information:

```
Scheduler 417 has FBS ID of 32768:    Cycles per frame = 101
Max programs per cycle = 10:    Max programs per FBS = 110
owner uid   = 9999:    owner gid   = 101
creator uid = 9999:    creator gid = 101:
total overruns = 0:    access mode = 600:    flags word = 1
active CPU mask = '----xxxx':    active PM CPU mask = '----x--x'
interrupt device name = /dev/rcim/rtc2
FBS is currently running
```

`Scheduler`

Indicates the user-specified key for the selected frequency-based scheduler.

`FBS ID`

Indicates the unique, positive integer value representing the identifier for the selected frequency-based scheduler.

`Cycles per frame`

Indicates the number of minor cycles that compose a major frame on the selected scheduler.

`Max programs per cycle`

Indicates the maximum number of programs that can be scheduled in a minor cycle on the selected scheduler.

`Max programs per FBS`

Indicates the maximum number of programs that can be scheduled on the selected scheduler at one time.

`owner uid`

Indicates the user ID of the scheduler's owner.

`owner gid`

Indicates the group ID of the scheduler's owner.

`creator uid`

Indicates the user ID of the scheduler's creator.

`creator gid`

Indicates the group ID of the scheduler's creator.

`total overruns`

Indicates the total number of overruns for all processes on the selected scheduler.

`access mode`

Indicates an octal value representing the permissions assigned to the selected scheduler.

`flags word`

Indicates if a timing source has been attached to the selected scheduler or if end-of-cycle scheduling has been enabled. If true, this field displays 1; otherwise, it displays 0.

`active CPU mask`

Contains a mask of the CPUs that are active in the system. The rightmost position corresponds to the first logical CPU. The letter **x** signifies that a CPU is active; the dash (–) signifies that it is not.

`active PM CPU mask`

Contains a mask of the CPUs for which performance monitoring has been enabled. The rightmost position corresponds to the first logical CPU. The letter **x** signifies that performance monitoring has been enabled on a CPU; the dash (–) signifies that it has not.

`interrupt device name`

If a timing source has been attached to the selected scheduler, this field contains the full path name of the device. If end-of-cycle scheduling has been enabled, this field contains the following: `EOC triggering`.

`FBS is`

Indicates the state of the selected scheduler.

ls – Display All Schedulers on the System

The **`ls`** command displays a single line of information for each frequency-based scheduler configured on the system.

Synopsis

`ls`

Display

This command displays the following information for each configured scheduler:

```
ID   SCHEDULER   PERMISSIONS OWNER_UID   OWNER_GID   STATE       DEVICE
---  ----------  ----------- ---------   ----------  -------     ------------
0    37          --rw-rw-r-- fbsusr1     5309        running     /dev/rcim/rtc0
1    38          --rw-rw-r-- fbsusr2     5309        running     /dev/rcim/rtc1
2    39          --rw-rw-r-- fbsusr1     5309        stopped     /dev/rcim/rtc2
3    40          --rw-rw-r-- fbsusr3     5309        running     /dev/rcim/rtc3
```

rc – Start Real-Time Clock

The **`rc`** command starts the real-time clock that has been specified as the timing source for a selected frequency-based scheduler (see "ats – Attach Timing Source to a Frequency-Based Scheduler" on page 5-11 for an explanation of the **`ats`** command). Note that you must first have set the count and resolution values for the real-time clock by executing the **`stc`** command (page 5-22).

Synopsis

`rc -s` *scheduler*

Arguments

`-s` *scheduler* — the numeric key associated with the frequency-based scheduler for which you wish to start the attached real-time clock. The scheduler must previously have been configured. See the section "cs – Configure a Frequency-Based Scheduler" on page 5-13 for information on configuring a frequency-based scheduler.

Display

The following message is displayed when the real-time clock is successfully started:

```
Clock started
```

sc – Stop Real-Time Clock

The **sc** command stops the real-time clock that has been specified as the timing source for a selected frequency-based scheduler.

Synopsis

sc –s *scheduler*

Arguments

–s *scheduler* — the numeric key associated with the frequency-based scheduler for which you wish to stop the attached real-time clock. The scheduler must previously have been configured. See the section "cs – Configure a Frequency-Based Scheduler" on page 5-13 for information on configuring a frequency-based scheduler.

Display

The following message is displayed when the real-time clock is successfully stopped:

```
Clock stopped
```

stc – Set Real-Time Clock

The **stc** command establishes the duration of a minor cycle by setting the count and duration values for a real-time clock that has been specified as the timing source for a selected frequency-based scheduler.

Synopsis

stc –s *scheduler* [**–D** *clock_duration*] **–O** *clock_count* [**–W** *tick_count*]

Arguments

–s *scheduler* — the numeric key associated with the frequency-based scheduler to which the real-time clock has been attached. The scheduler must previously have been configured. See the section "cs – Configure a Frequency-Based Scheduler" on page 5-13 for information on configuring a frequency-based scheduler.

–D *clock_duration* — the duration in microseconds of one clock count. The value of *clock_duration* must be one of the following: 1, 10, 100, 1000, 10000. The default value is 10.

–O *clock_count* — the number of clock counts per minor cycle. The value of *clock_count* can range from 2 to 65535.

–W *tick_count* — For devices that have watchdog timer support, the *tick_count* is passed into the driver as a count of sequential interrupts that are allowed to be missed, before the watchdog timer takes over driving the FBS. For the driver to support this feature, it must accept the WATCHDOGSET(_IOW('w',14, int)) and WATCHDOGGET(_IOR('w',13, int)) **ioctl** commands.

NOTE: Although a *clock_count* of 1 cannot be used, a timing interval equal to a *clock_duration* value can be set by using the next lower *clock_duration* value

and a *clock_count* of 10; e.g., -D 1000 -O 10 results in a timing interval of 10,000 microseconds.

Display

The following message is displayed when the real-time clock is successfully set:

```
Clock set
```

gtc – Display Real-Time Clock Settings

The **gtc** command displays the current count and duration values for a real-time clock that has been specified as the timing source for a selected frequency-based scheduler. The clock must previously have been set using the **stc** command (page 5-22).

Synopsis

gtc –s *scheduler*

Arguments

–s *scheduler* — the numeric key associated with the frequency-based scheduler to which the real-time clock has been attached. The scheduler must previously have been configured. See the section "cs – Configure a Frequency-Based Scheduler" on page 5-13 for information on configuring a frequency-based scheduler.

Display

When the real-time clock is currently set, this command displays the following information:

```
Clock count = 50: duration = 1000
```

start – Start Scheduling on a Frequency-Based Scheduler

The **start** command starts scheduling processes on a frequency-based scheduler. When you execute this command, the minor cycle, major frame, and overrun count values are set to zero.

Prior to executing this command, you must have executed the **ats** command to attach a timing source to the scheduler or to specify end-of-cycle scheduling (see "ats – Attach Timing Source to a Frequency-Based Scheduler" on page 5-11 for an explanation of the **ats** command).

If you have specified a real-time clock as the timing source for the scheduler, scheduling will not start until you have set the clock using the **stc** commands (page 5-22), and started the clock using the **rc** command (page 5-21).

If you have specified an edge-triggered interrupt as the timing source, it must already be generating interrupts in order for scheduling to start.

Synopsis

start -s *scheduler*

Arguments

-s *scheduler*	the numeric key associated with the frequency-based scheduler on which you wish to start scheduling. The scheduler must previously have been configured. See the section "cs – Configure a Frequency-Based Scheduler" on page 5-13 for information on configuring a frequency-based scheduler.

Display

The following message is displayed when scheduling on the specified scheduler is successfully started:

```
FBS started
```

resume – Resume Scheduling on a Frequency-Based Scheduler

The **resume** command resumes scheduling processes on a selected frequency-based scheduler with the major frame, minor cycle, and overrun count values the same as they were when you stopped the scheduler.

Synopsis

resume -s *scheduler*

Arguments

-s *scheduler*	the numeric key associated with the frequency-based scheduler for which you wish to resume scheduling. The scheduler must previously have been configured. See the section "cs – Configure a Frequency-Based Scheduler" on page 5-13 for information on configuring a frequency-based scheduler.

Display

The following message is displayed when scheduling on the specified scheduler is successfully resumed:

```
FBS resumed
```

stop – Stop Scheduling on a Frequency-Based Scheduler

The **stop** command stops scheduling processes on a selected frequency-based scheduler.

Synopsis

stop -s *scheduler*

Arguments

-s *scheduler*	the numeric key associated with the frequency-based scheduler for which you wish to stop scheduling. The scheduler must previously have been configured. See the section "cs – Configure a Frequency-Based Scheduler" on page 5-13 for information on configuring a frequency-based scheduler.

Display

The following message is displayed when scheduling on the specified scheduler is successfully stopped:

```
FBS stopped
```

rmp – Remove a Process from a Frequency-Based Scheduler

The **rmp** command removes a process from a frequency-based scheduler. Identify the process that you wish to remove by specifying one of the following:

- the name of the process and the CPU on which it is scheduled
- the process' frequency-based scheduler process identifier
- the name of the process, the CPU on which it is scheduled, and its frequency-based scheduler process identifier

NOTE

The only method that can be used to identify a process that has been scheduled multiple times on the same CPU is to specify its frequency-based scheduler identifier.

Synopsis

rmp –s *scheduler* {**–i** *fpid* | –**n** *proc_name* [**–c** *cpu_bias*]} [**–a**]

Arguments

–s *scheduler* — the numeric key associated with the frequency-based scheduler on which the process is scheduled. The scheduler must previously have been configured. See the section "cs – Configure a Frequency-Based Scheduler" on page 5-13 for information on configuring a frequency-based scheduler.

–i *fpid* — the frequency-based scheduler process identifier for the process to be removed. This value is displayed when you successfully schedule a program by executing the **sp** command (page 5-32). If you have not identified the process by name, you must specify this argument.

The default value for *fpid* is -1. If you accept the default value, you must identify the process by name and CPU.

–n *proc_name* — a standard path name identifying the process to be removed from the specified scheduler. This can be a full or relative path name of up to 1024 characters.

If you do not specify this argument, you must provide the frequency-based scheduler process identifier by specifying the **–i** *fpid* argument.

–c *cpu_bias* — the processor(s) to be used with the value of the **–n** *proc_name* argument to identify the process to be removed from the specified scheduler.

The value of *cpu_bias* may be a single CPU ID or a list of CPU IDs. CPU IDs range from zero to one less than the number of CPUs, where the number 0 represents the first logical CPU, 1 represents the second, and so on.

A list of CPU IDs may specify a sequence or a range of numbers––for example, **`-c 1,3–5,7`**. Note that you must use commas to separate items in the list. Specify the entire range of CPU IDs by entering an asterisk (*****).

If you do not specify the **`-c`** *cpu_bias* argument, the default processor is the CPU on which **`rtcp`** is currently executing.

`-a` the process is to be removed from the specified scheduler and terminated. If this option is not specified, the process is removed from the scheduler but allowed to continue executing.

Display

If the specified process is successfully removed from the scheduler, the following message is displayed:

```
Process removed
```

rsp – Reschedule a Process

The **rsp** command changes the scheduling parameters for a process that has been scheduled on a frequency-based scheduler. You may wish, for example, to change the process' scheduling policy or priority. You may also wish to change the frequency with which the process is scheduled to run. You cannot, however, change the CPU on which it has been scheduled.

The effective user ID of the calling process must match the effective user ID of the target process (the process for which the scheduling policy and priority are being set) for the following conditions:

- if you execute this command and you wish to change a process' scheduling policy to the first-in-first-out (FIFO) or the round-robin policy
- if you wish to change the priority of a process scheduled under the FIFO or the round-robin policy
- if you wish to raise the priority of a process scheduled under the time-sharing policy above a per-process limit

You can reschedule a process without first having executed the **rmp** command (page 5-26) to remove it from the scheduler or the **stop** command (page 5-25) to stop scheduling.

Use one of the following methods to identify the process that you wish to reschedule:

- specify the name of the process and the CPU on which it is scheduled
- specify the process' frequency-based scheduler process identifier
- specify the name of the process, the CPU on which it is scheduled, and its frequency-based scheduler process identifier.

NOTE

The only method that can be used to identify a process that has been scheduled multiple times on the same CPU is to specify its frequency-based scheduler identifier.

Synopsis

rsp –s *scheduler* {**-i** *fpid* | –**n** *proc_name* [**-c** *cpu_bias*]} [**-f** *frequency*] \
[**-m** *start_cycle*] [**-b** *policy*] [**-p** *priority*] [**-o** *halt_flag*] [**-L** *soft_limit*] [**-T** *deadline*] [**-h** *halt_flag*] [**-r** *origin*]

Arguments

-s *scheduler* — the numeric key associated with the frequency-based scheduler on which the process is scheduled. The scheduler must previously have been configured. See the section "cs – Configure a Frequency-Based Scheduler" on page 5-13 for information on configuring a frequency-based scheduler.

-i *fpid* — the frequency-based scheduler process identifier for the process to be rescheduled. This value is displayed when you execute the **sp** command (page 5-32). If you have not identified the process

by name, you must specify this argument. The default value for *fpid* is -1. If you use the default value, you must identify the process by name and CPU.

-n *proc_name* — a standard path name identifying the process to be rescheduled. This can be a full or relative path name of up to 1024 characters. If you do not specify this argument, you must provide the frequency-based scheduler process identifier by specifying the **-i** *fpid* argument.

-c *cpu_bias* — the processor(s) to be used with the value of the **-n** *proc_name* argument to identify the process to be rescheduled.

The value of *cpu_bias* may be a single CPU ID or a list of CPU IDs. CPU IDs range from 0 to one less than the number of CPUs, where the number 0 represents the first logical CPU, 1 represents the second, and so on.

A list of CPU IDs may specify a sequence or a range of numbers––for example, **–c 1,3–5,7**. Note that you must use commas to separate items in the list. Specify the entire range of CPU IDs by entering an asterisk (*****).

If you do not specify this argument, the default processor is the CPU on which the real-time command processor is currently executing.

-f *frequency* — the frequency with which the specified process is to be awakened in each major frame.

A frequency of 1 indicates that the specified process is to be awakened every minor cycle; a frequency of 2 indicates that it is to be awakened once every two minor cycles and so on.

Specify the number of minor cycles representing the frequency with which you wish the process to be awakened. The value of *frequency* can range from 1 to the number of minor cycles that compose a frame on the scheduler. The default is 1.

The total number of minor cycles per frame is defined by executing the **cs** command (page 5-13).

-m *start_cycle* — the first minor cycle in which the specified process is to be awakened in each frame. The value of *start_cycle* can range from 0 to the total number of minor cycles per frame minus 1. The default is 0. The total number of minor cycles per frame is defined by executing the **cs** command (see page 5-13).

-b *policy* — the POSIX scheduling policy for the specified program: **F** to select the first-in-first-out (FIFO) scheduling policy, **R** to select the round-robin scheduling policy or **O** to select the time-sharing scheduling policy.

If you do not specify the **-b** *policy* argument, the default policy is the time-sharing scheduling policy.

Note: It is recommended that you specify this argument.

-p *priority* — the scheduling priority for the specified process. The default value is 0.

The range of priority values that you can enter is governed by the value of the policy argument. Determine the allowable range of priorities associated with each policy (**F, R,** or **O**) by invoking the **run(1)** command from the shell and not specifying any options or arguments. Higher numerical values correspond to *more* favorable scheduling priorities.

-o *halt_flag* — indicates whether or not the scheduler should be stopped in the event that the specified process causes a frame overrun. The value of *halt_flag* must be either **halt** or **nohalt**. The default is **nohalt**.

-L *soft_limit* — the soft overrun limit for the process. The default is 0. If you reschedule a process that already has a non-zero soft overrun limit set and you do not specify a soft overrun limit, the process' soft overrun limit is set to 0.

-T *deadline* — Specifies the maximum time the task is expected to execute before returning to **fbswait(3)**. Can be **Clear**, which removes any deadline constraint, or a positive number denoting a deadline time in microseconds. The default value is **Clear**.

-h *halt_flag* — Enables you to indicate if the scheduler should be stopped in the event that a deadline violation is detected for the specified process. Can be set to **halt** or **nohalt**. The default is **nohalt**.

-r *origin* — Indicates the starting point from which to measure time when testing for deadline violations. Can be set to **cycle** to measure time from the beginning of the cycle in which the task is scheduled, or **task** to measure time starting when the task exits **fbswait(3)** and begins executing. The default is **cycle**.

Display

When the specified process is successfully rescheduled, the following is displayed:

```
CPU    FPID  Prio.   Freq.    Start   Halt  SoftLimit  Ddln(us)/Origin    HaltOnDL    Process Name
 0        1   20/F     2          0    T        2           250/C             T       /home/jojo/wc
 0        0   53/F     2          0    T        0            None             -       /home/jojo/wc
```

CPU

The identifier for the CPU on which the specified process is scheduled.

FPID

The unique frequency-based scheduler procesLs identifier for the specified process. This identifier is displayed by the real-time command processor

when you schedule a program on the scheduler using the **sp** command (page 5-32).

`Prio.`

The scheduling priority of the specified process.

`Freq.`

The frequency with which the specified process is scheduled to be awakened in each major frame.

`Start`

The first minor cycle in which the specified process is scheduled to be awakened in each major frame.

`Halt`

Indicates whether or not the “halt on overrun” flag has been set for the specified process; Y (yes) or N (no).

`SoftLimit`

The soft overrun limit of the process.

`Ddln(us)/Origin`

The deadline time in microseconds, displayed with a slash and the origin code. The origin is `C` for DL_CYCLE_RELATIVE and `T` for DL_TASK_RELATIVE.

`HaltOnDl`

The value of the “Halt on deadline violation” flag: `T`, `F`, or if no deadline is assigned, “-” .

`Process Name`

The full path name of the process that has been scheduled on the selected frequency-based scheduler.

sp – Schedule a Process on a Frequency-Based Scheduler

The **sp** command creates a process and schedules it on a frequency-based scheduler.

The effective user ID of the calling process must match the effective user ID of the target process (the process for which the scheduling policy and priority are being set) for the following conditions:

- if you execute this command and you wish to change a process' scheduling policy to the first-in-first-out (FIFO) or the round-robin policy
- if you wish to change the priority of a process scheduled under the FIFO or the round-robin policy
- if you wish to raise the priority of a process scheduled under the time-sharing policy above a per-process limit.

If you wish to modify the CPU bias of the process when you invoke this command, the real or effective user ID of the calling process must match the real or saved user ID of the process for which the CPU assignment is being changed.

When you execute this command, the real-time command processor returns a unique frequency-based scheduler process identifier. You can subsequently use this identifier to specify the process when you are executing other commands.

Synopsis

sp –s *scheduler* **–n** *proc_name* [**–c** *cpu_bias*] [**–f** *frequency*] [**–m** *start_cycle*] [**–b** *policy*] \
[**–p** *priority*] [**–v** *parameter*] [**–o** *halt_flag*] [**–L** *soft_limit*] [**-T** *deadline*] [**-h** *halt_flag*] [**-r** *origin*] [**--** *arg1* [*arg2...*]]

Arguments

–s *scheduler* — the numeric key associated with the frequency-based scheduler on which the process is to be scheduled. The scheduler must previously have been configured. See the section "cs – Configure a Frequency-Based Scheduler" on page 5-13 for information on configuring a frequency-based scheduler.

–n *proc_name* — a standard path name identifying the program that you wish to schedule. This can be a full or relative path name of up to 1024 characters.

–c *cpu_bias* — the CPU bias for the specified program. The CPU bias determines the processor or processors on which the program can be scheduled.

The value of *cpu_bias* may be a single CPU ID or a list of CPU IDs. CPU IDs range from 0 to one less than the number of CPUs, where the number 0 represents the first logical CPU, 1 represents the second, and so on.

A list of CPU IDs may specify a sequence or a range of numbers––for example, **–c 1,3–5,7**. Note that you must use commas to separate items in the list. Specify the entire range of CPU IDs by entering an asterisk (*).

If you do not specify this argument, the default processor is the CPU on which the real-time command processor is currently executing.

-f *frequency* — the frequency with which the specified process is to be awakened in each major frame.

A frequency of 1 indicates that the specified process is to be awakened every minor cycle; a frequency of 2 indicates that it is to be awakened once every two minor cycles, and so on.

Specify the number of minor cycles representing the frequency with which you wish the process to be awakened. The value of *frequency* can range from 1 to the number of minor cycles that compose a frame on the scheduler. The default value is 1. The total number of minor cycles per frame is defined by executing the **cs** command (see page 5-13).

-m *start_cycle* — the first minor cycle in which the specified process is scheduled to be awakened in each frame. The value of *start_cycle* can range from 0 to the total number of minor cycles per frame minus one. The default value is 0. The total number of minor cycles per frame is defined by executing the **cs** command (see page 5-13).

-b *policy* — the POSIX scheduling policy for the specified process: **F** to select the first-in-first-out (FIFO) scheduling policy, **R** to select the round-robin scheduling policy or **O** to select the time-sharing scheduling policy.

If you do not specify the **-b** *policy* argument, the default policy is the time-sharing scheduling policy.

Note: It is recommended that you specify this argument.

-p *priority* — the scheduling priority for the specified process. The default value is 0.

The range of priority values is governed by the value of the policy argument. Determine the allowable range of priorities associated with each policy (**F, R,** or **O**) by invoking the **run(1)** command from the shell and not specifying any options or arguments.

Higher numerical values correspond to more favorable scheduling priorities.

-v *parameter* — an integer value to be passed to a process that is scheduled on a frequency-based scheduler.

The value of *parameter* must be a 32-bit decimal number.

-o *halt_flag* — indicates whether or not the scheduler should be stopped in the event that the specified program causes a frame overrun.

The value of *halt_flag* must be either **halt** or **nohalt**. The default is **nohalt**.

-L *soft_limit* — the soft overrun limit for the process. The default is 0.

-T *deadline*	Specifies the maximum time the task is expected to execute before returning to **fbswait(3)**. Can be **Clear**, which removes any deadline constraint, or a positive number denoting a deadline time in microseconds. The default value is **Clear**.
-h *halt_flag*	Enables you to indicate if the scheduler should be stopped in the event that a deadline violation is detected for the specified process. Can be set to **halt** or **nohalt**. The default is **nohalt**.
-r *origin*	Indicates the starting point from which to measure time when testing for deadline violations. Can be set to **cycle** to measure time from the beginning of the cycle in which the task is scheduled, or **task** to measure time starting when the task exits **fbswait(3)** and begins executing. The default is **cycle**.
-- *arg1* [*arg2...*]	arguments that are passed to the scheduled process' command line, in *argv[]*

Display

The following information is displayed when the specified process is successfully scheduled on the frequency-based scheduler:

```
fpid 199 assigned to process task02
```

`fpid`

The unique frequency-based scheduler process identifier assigned to the specified process.

`process`

The full path name of the process that has been scheduled on the selected frequency-based scheduler.

vp – View Processes on a Frequency-Based Scheduler

The **vp** command displays information about a particular frequency-based scheduled process or all frequency-based scheduled processes on one or more processors on a selected frequency-based scheduler.

You can display information about all frequency-based scheduled processes on a specified processor or all processors by specifying the scheduler and the CPU(s) on which the processes are scheduled.

If you wish to display information for a particular frequency-based scheduled process, you can identify the process by specifying:

- the name of the process and the CPU on which it is scheduled
- the process' frequency-based scheduler process identifier
- the name of the process, the CPU on which it is scheduled, and its frequency-based scheduler process identifier

NOTE

The only method that can be used to identify a process that has been scheduled multiple times on the same CPU is to specify its frequency-based scheduler identifier.

Information displayed for each process includes:

- the CPU on which the process is executing
- the frequency-based scheduler process identifier
- the process' scheduling priority
- the frequency (the number of minor cycles indicating the frequency with which the process is awakened in each major frame)
- the starting base cycle (the first minor cycle in which the process is scheduled to be awakened in each major frame)
- the value of the "halt on overrun" flag
- the soft overrun limit of the process
- the set deadline time
- the deadline origin indicator
- the value of the "halt on deadline violation" flag
- the path name of the process.

Synopsis

vp –s *scheduler* [**–n** *proc_name*] [**–i** *fpid*] [**–c** *cpu_bias*]

Arguments

–s *scheduler* — the numeric key associated with the frequency-based scheduler for which scheduling information is to be displayed. The scheduler must previously have been configured. See the "cs –

Configure a Frequency-Based Scheduler" section on page 5-13 for information on configuring a frequency-based scheduler.

-n *proc_name* — a standard path name identifying a particular process for which scheduling information is to be displayed. This can be a full or relative path name of up to 1024 characters.

-i *fpid* — the frequency-based scheduler process identifier for a particular process for which scheduling information is to be displayed. This value is displayed when you successfully schedule a program by executing the **sp** command (page 5-32).

The default value for *fpid* is -1. If you accept the default value, you must identify the process by name and CPU.

-c *cpu_bias* — the processor(s) for which scheduling information is to be displayed.

The value of *cpu_bias* may be a single CPU ID or a list of CPU IDs. CPU IDs range from 0 to one less than the number of CPUs, where the number 0 represents the first logical CPU, 1 represents the second, and so on.

A list of CPU IDs can specify a sequence or a range of numbers—for example, **-c 1,3–5,7**. Note that you must use commas to separate items in the list. Specify the entire range of CPU IDs by entering an asterisk (*).

The default processor is the CPU on which the real-time command processor is currently executing.

Display

The following information is displayed when the command is successfully executed:

```
CPU    FPID  Prio.   Freq.    Start   Halt   SoftLimit  Ddln(us)/Origin    HaltOnDL    Process Name
 0        1   20/F     2          0     T        2           250/C             T         /home/jojo/wc
 0        0   53/F     2          0     T        0            None             -         /home/jojo/wc
```

CPU

The identifiers for the CPUs on which the respective processes are scheduled.

FPID

The frequency-based scheduler process identifiers for the respective processes. Identifiers are displayed by the real-time command processor when you schedule a program on the scheduler using the **sp** command (page 5-32).

`Prio.`

The scheduling priorities of the respective processes.

`Freq.`

The frequency with which the respective processes are scheduled to be awakened in each major frame.

`Start`

The first minor cycle in which the respective processes are scheduled to be awakened in each major frame.

`Halt`

Indicates whether or not the “halt on overrun” flag has been set for the respective processes; Y (yes) or N (no).

`SoftLimit`

The soft overrun limit of the process.

`Ddln(us)/Origin`

The deadline time in microseconds, displayed with a slash and the origin code. The origin is `C` for `DL_CYCLE_RELATIVE` and `T` for `DL_TASK_RELATIVE`.

`HaltOnDl`

The value of the “Halt on deadline violation” flag: `T`, `F`, or if no deadline is assigned, “-” .

`Process Name`

The full path names of the processes that have been scheduled on the selected frequency-based scheduler.

cpm – Clear Performance Monitor Values

The **cpm** command clears performance monitor values for a particular process or all processes on one or more processors for a selected frequency-based scheduler.

You can clear values for all frequency-based scheduled processes on a specified processor or all processors by specifying the scheduler and the CPU(s) on which the processes are scheduled.

If you wish to clear performance monitor values for a particular frequency-based scheduled process, you can identify the process by specifying:

- the name of the process and the CPU on which it is scheduled
- the process' frequency-based scheduler process identifier
- the name of the process, the CPU on which it is scheduled, and its frequency-based scheduler process identifier

Synopsis

cpm –s *scheduler* [**–i** *fpid*] [–**n** *proc_name*] [**–c** *cpu_bias*]

Arguments

–s *scheduler* — the numeric key associated with the frequency-based scheduler on which the process or processes are scheduled. The scheduler must previously have been configured. See the section "cs – Configure a Frequency-Based Scheduler" section on page 5-13 for information on configuring a frequency-based scheduler.

–i *fpid* — the unique frequency-based scheduler process identifier for a particular process for which values are to be cleared. This value is displayed when you execute the **sp** command (page 5-32).

The default value for *fpid* is -1. If you accept the default value, you must identify the process by name and CPU.

–n *proc_name* — a standard path name identifying a particular process for which values are to be cleared. This can be a full or relative path name of up to 1024 characters.

–c *cpu_bias* — the processor(s) for which performance monitor values are to be cleared.

The value of *cpu_bias* may be a single CPU ID or a list of CPU IDs. CPU IDs range from 0 to one less than the number of CPUs, where the number 0 represents the first logical CPU, 1 represents the second, and so on.

A list of CPU IDs may specify a sequence or a range of numbers—for example, **–c 1,3–5,7**. Note that you must use commas to separate items in the list. Specify the entire range of CPU IDs by entering an asterisk (*).

The default processor is the CPU on which the real-time command processor is currently executing.

Display

The following message is displayed when the performance monitor values are successfully cleared:

```
Performance monitor values cleared
```

NOTE

This command clears the soft overrun count for all processes specified by the user.

pm – Start/Stop Performance Monitoring

The **pm** command starts or stops performance monitoring for a particular process or all processes on one or more processors for a selected frequency-based scheduler.

You can start or stop performance monitoring for all frequency-based scheduled processes on a specified processor or all processors by specifying the scheduler and the CPU(s) on which the processes are scheduled.

If you wish to start or stop performance monitoring for a particular frequency-based scheduled process, you can identify the process by specifying:

- the name of the process and the CPU on which it is scheduled
- the process' frequency-based scheduler process identifier
- the name of the process, the CPU on which it is scheduled, and its frequency-based scheduler process identifier

Synopsis

pm –s *scheduler* [**–i** *fpid*] [–**n** *proc_name*] [**–c** *cpu_bias*] [**–P** *pm_flag*]

Arguments

–s *scheduler* — the numeric key associated with the frequency-based scheduler on which the process or processes are scheduled. The scheduler must previously have been configured. See the section "cs – Configure a Frequency-Based Scheduler" on page 5-13 for information on configuring a frequency-based scheduler.

–i *fpid* — the unique frequency-based scheduler process identifier for a particular process for which performance monitoring is to be started or stopped. This value is displayed when you execute the **sp** command (page 5-32).

The default value for *fpid* is -1. If you accept the default value, you must identify the process by name and CPU.

–n *proc_name* — a standard path name identifying a particular process for which performance monitoring is to be started or stopped. This can be a full or relative path name of up to 1024 characters.

–c *cpu_bias* — the processor(s) for which performance monitoring is to be started or stopped.

The value of *cpu_bias* may be a single CPU ID or a list of CPU IDs. CPU IDs range from 0 to one less than the number of CPUs, where the number 0 represents the first logical CPU, 1 represents the second, and so on.

A list of CPU IDs may specify a sequence or a range of numbers—for example, **–c 1,3–5,7**. Note that you must use commas to separate items in the list. Specify the entire range of CPU IDs by entering an asterisk (*).

The default processor is the CPU on which the real-time command processor is currently executing.

`-P` *pm_flag* indicates whether performance monitoring is to be started or stopped. The value of *pm_flag* must be either **ON** or **OFF**. The default is **OFF**.

Display

The following message is displayed when performance monitoring is successfully started:

```
Performance monitoring enabled
```

The following message is displayed when performance monitoring is successfully stopped:

```
Performance monitoring disabled
```

vcm – View/Modify Performance Monitor Timing Mode

The **vcm** command views or modifies the performance monitor timing mode. The timing mode can be set to include or exclude time spent servicing interrupts from the performance monitor timing values.

When interrupt time is included, a process' user and system times will total the elapsed time which accrues when the process is the currently running process, including all time spent servicing interrupts. The time spent servicing interrupts is added to the process' system time.

When excluding interrupt time, a process' user and system times will total the time which accrues when the process is the currently running process, excluding all time spent servicing interrupts.

CAUTION

The timing mode is the mode in which the High-Resolution Process Timing Facility operates system-wide. It affects all processes running on all CPUs.

Synopsis

vcm [**-t** *pm_tmode*]

Arguments

-t *pm_tmode*	specifies whether interrupt time is to be included in or excluded from performance monitor timing values. The value of *pm_tmode* must be either **in** (included) or **ex** (excluded).

Display

One of the following messages is displayed when you successfully execute the **vcm** command to view the performance monitor timing mode:

```
PM timing mode includes interrupt times.
```

or

```
PM timing mode excludes interrupt times.
```

vpm – View Performance Monitor Values

The **vpm** command displays performance monitor values for one process or all processes on one or more processors for a selected frequency-based scheduler.

Display values for all frequency-based scheduled processes on a processor or all processors by specifying the scheduler and the CPU(s) on which the processes are scheduled.

If you wish to display performance monitor values for a particular frequency-based scheduled process, you can identify the process by specifying:

- the name of the process and the CPU on which it is scheduled
- the process' frequency-based scheduler process identifier
- the name of the process, the CPU on which it is scheduled, and its frequency-based scheduler process identifier

Synopsis

vpm –s *scheduler* [**–i** *fpid*] [–**n** *proc_name*] [**–c** *cpu_bias*] [**–x** *pm_output*]

Arguments

–s *scheduler* — the numeric key associated with the frequency-based scheduler on which the process or processes are scheduled. The scheduler must previously have been configured. See the section "cs – Configure a Frequency-Based Scheduler" on page 5-13 for information on configuring a frequency-based scheduler.

–i *fpid* — the unique frequency-based scheduler process identifier for a particular process for which performance monitor values are to be displayed. This value is displayed when you execute the **sp** command (see page 5-32).

The default value for *fpid* is -1. If you accept the default value, you must identify the process by name and CPU.

–n *proc_name* — a standard path name identifying a particular process for which performance monitor values are to be displayed. This can be a full or relative path name of up to 1024 characters.

–c *cpu_bias* — the processor(s) for which performance monitor values are to be displayed.

The value of *cpu_bias* may be a single CPU ID or a list of CPU IDs. CPU IDs range from 0 to one less than the number of CPUs, where the number 0 represents the first logical CPU, 1 represents the second, and so on.

A list of CPU IDs may specify a sequence or a range of numbers—for example, **–c 1,3–5,7**. Note that you must use commas to separate items in the list. Specify the entire range of CPU IDs by entering an asterisk (*).

The default processor is the CPU on which the real-time command processor is currently executing.

-x *pm_output* indicates the type of values to display. Four types of values can be specified: average, minimum, maximum, or all.

Specify **av** (average) to display: the number of iterations, or cycles; the last time; the total time; the average time; and the number of overruns.

Specify **mi** (minimum) to display: the number of iterations; the minimum cycle time and the number of the minor cycle and the major frame in which it has occurred; and the minimum frame time and the number of the major frame in which it has occurred.

Specify **ma** (maximum) to display: the number of iterations; the maximum cycle time and the number of the minor cycle and major frame in which it has occurred; and the maximum frame time and the number of the major frame in which it has occurred.

Specify **al** (all) to display average, minimum, and maximum values.

Times are reported in microseconds. The default value is **av** (average).

Display

The following information is displayed when you select **average** values and the command is successfully executed:

```
                                                                  OverRuns
fpid     Iterations    TimeLast(us)     TotalTime(us)     Average(us)     Hard        Soft
199         480            1023            499200            1040           0           0
198         120            2013            240960            2008           1           2
197         240            1521            378960            1579           0           3
```

`fpid`

The unique frequency-based scheduler process identifiers for the processes. This identifier is displayed by the real-time command processor when you schedule a program on the scheduler using the **sp** command (see page 5-32).

`Iterations`

The number of times that the processes have been awakened by the frequency-based scheduler since the last time that performance monitor values have been cleared and performance monitoring has been enabled.

`TimeLast(us)`

The amount of time in microseconds that the processes have spent running from the last time that they were awakened by the frequency-based scheduler until they called **`fbswait`**.

`TotalTime(us)`

The cumulative times in microseconds that the processes have spent running in all cycles, or iterations.

`Average(us)`

The average amount of time in microseconds that the respective processes have spent running in all cycles, or iterations. These values are obtained by dividing the values reported in the Total Time column by the values reported in the Iterations column.

`Overruns`

`Hard` The number of times that the respective processes have caused a catastrophic frame overrun.

`Soft` The number of times that the respective processes have caused a non-catastrophic frame overrun.

The following information is displayed when you select **`minimum`** values and the command is successfully executed:

		Minimum Cycle		Minimum Frame	
fpid	Iterations	time(us)	Frame/Cycle	time(us)	Frame
199	30	1002	6/7	8013	17
198	30	1943	2/1	3995	22
197	30	1312	1/2	5314	11

`fpid`

The unique frequency-based scheduler process identifiers for the respective processes. This identifier is displayed by the real-time command processor when you schedule a program on the scheduler using the **`sp`** command (see page 5-32).

`Iterations`

The number of times that the respective processes have been awakened by the frequency-based scheduler since the last time that performance monitor values have been cleared and performance monitoring has been enabled.

`Minimum Cycle time(us)`

The least amount of time in microseconds that the processes spent running in a cycle, or iteration.

`Minimum Cycle Frame/Cycle`

The number of the major frame and the minor cycle in which the minimum cycle time has occurred.

`Minimum Frame time(us)`

The least amount of time in microseconds that the processes spent running during a major frame.

`Minimum Frame Frame`

The number of the major frame in which the minimum frame time has occurred.

The following information is displayed when you select **`maximum`** values and the command is successfully executed:

```
                              Maximum Cycle                    Maximum Frame
fpid      Iterations      time(us)       Frame/Cycle       time(us)          Frame

199           30            1303             2/4             9502             12
198           30            2201             7/5             4391             17
197           30            1917             9/6             7431             24
```

`fpid`

The unique frequency-based scheduler process identifiers for the respective processes. This identifier is displayed by the real-time command processor when you schedule a program on the scheduler using the **sp** command (see page 5-32).

`Iterations`

The number of times that the respective processes have been awakened by the frequency-based scheduler since the last time that performance monitor values were cleared and performance monitoring was enabled.

`Maximum Cycle time(us)`

The greatest amount of time in microseconds that the processes spent running in a cycle, or iteration.

`Maximum Cycle`
`Frame/Cycle`

The number of the major frame and the minor cycle in which the maximum cycle time has occurred.

`Maximum Frame`
`time(us)`

The greatest amount of time in microseconds that the processes have spent running during a major frame.

`Maximum Frame`
`Frame`

The number of the major frame in which the maximum frame time has occurred.

ex – Exit Real-Time Command Processor

The **`ex`** command is used only when you are using the real-time command processor in interactive mode. It exits the command processor and returns to the shell.

Synopsis

`ex`

The system command prompt is displayed when the command is successfully executed.

he – Display Help Information

The **`he`** command displays the following types of help information for the real-time command processor:

- a list and brief descriptions of all commands
- a description and format of a particular command
- a list and description of all command arguments

Synopsis

`he` [*command* | **`option`** | **`op2`**]

Arguments

command	the command for which you wish to obtain information
`option`	the first screen of command arguments is to be displayed
`op2`	the second screen of command arguments is to be displayed

Display

If you specify the **he** command without an argument, a list of all commands with a brief description is displayed. See Screen 5-1 on page 5-5.

If you specify the **he** command with the *option* argument, the first screen of **rtcp** command arguments is displayed. See Screen 5-2 on page 5-6.

If you specify the **he** command with the *op2* argument, the second screen of **rtcp** command arguments is displayed. See Screen 5-3 on page 5-7.

If you specify the **he** command with the *command* argument, a description of the specified command and the format for entering the command is displayed. For example, **he chs** displays the following:

```
Change FBS permissions

rtcp chs -s scheduler -I permissions [-G gid] [-U uid]
```

6
The C Library Interface

This chapter describes the C library interface to the Frequency-Based Scheduler and the Performance Monitor. Library information, call sequences for using the scheduler and the Performance Monitor, and details for each of the library routines are provided.

Overview

Access to the functions associated with the Frequency-Based Scheduler and the Performance Monitor is provided through libraries of routines that can be called from application programs written in C.

The following information is provided in this chapter for each routine:

- a description of the routine
- the synopsis of the routine
- detailed descriptions of each parameter
- the return value

Information about the individual routines, including an illustration of the sequence in which you would call the routines during frequency-based scheduling, begins on page 6-4. The same information is provided for performance monitoring beginning on page 6-69.

An example program that illustrates use of the C library interface to the Frequency-Based Scheduler and the Performance Monitor is provided in Appendix C.

Compiling and Linking Programs

When statically linking a C program, the following libraries are required:

```
/usr/lib/libccur_fbsched.a
/usr/lib/libccur_rt.a
```

When dynamically linking a C program, the following libraries are used:

```
/usr/lib/libccur_fbsched.so
/usr/lib/libccur_rt.so
```

To compile and link a C program, the command line instruction is as follows:

gcc [*options...*] *source_file*`.c -lccur_fbsched -lccur_rt`

For additional information on compiling and linking procedures, refer to the **gcc(1)** and **ld(1)** man pages.

The Big-SMP FBS Interface

Historically, the FBS API has used 32 bit integers as CPU masks. This kept the FBS from being usable on big-SMP platforms; that is, on platforms which have one or more CPU IDs with a value >= 32.

Therefore, a new FBS API has been designed. To aid in the transformation of applications from the old to the new API, the new FBS API has been made as identical as possible to the old. Please note that this new API does not replace the old; as long as the maximum CPU ID on a platform is <32, both the old and the new FBS API can be used interchangeably.

The general transformation rules used to create the new FBS API are:

1. Everywhere in the old API that an integer was used as a CPU mask, a pointer to a CPU set, as defined in **cpuset(3)**, is used instead.
2. For every old FBS API function that directly or indirectly uses an integer bitmask, the new big-SMP version of the function has been created with the same name as the old, with a '_big' suffixed. For example, **fbsinfo_big(3)** is the big-SMP variant of **fbsinfo(3)**.
3. For every old FBS API data structure that has an integer CPU mask field, the new big-SMP variant has the same name as the old with a '_big' inserted somewhere in its name. For example, the big-SMP variant of 'struct pgm2_ds' is 'struct pgm2_big_ds'.
4. For each new FBS API data structure, each cpuset field has the same name as the old integer CPU mask field, with the characters 'cpu' replaced by 'cpuset' or 'cpuset_'. For example, the field 'pm_cpuactive' in 'struct fbsinfo_ds' becomes the field 'pm_cpuset_active' in 'struct fbsinfo_big_ds'.
5. No change is made to those FBS API data structures or functions which do not directly or indirectly reference an integer CPU mask. For example, 'struct pgmqry_ds' has no CPU masks in it, so it is used by both **pmqrypgm(3)** and by its big-SMP variant, **pgmqrypgm_big(3)**.

All in all, there are 21 new functions and 9 new data structures in the FBS API.

To make the conversion of applications a bit easier, the following cpuset extensions are allowed:

1. For those places in the FBS API where a cpuset is an output-only parameter, if the cpuset pointer is NULL then no cpuset will be returned. This is useful for those cases where it is known that the application isn't using the returned cpuset (which is likely to be most of the time).
2. For those places in the FBS API where a cpuset is an input-only parameter, the pointer to the cpuset can be one of several special values in lieu of being a pointer to a real cpuset. These special pointers are:

   FBS_CPUSET_ALLCPUS The full set.

`FBS_CPUSET_CURCPU`	A `cpuset` with a single CPU ID, that of the invoking CPU.
`FBS_CPUSET_ONECPU(`*`cpu`*`)`	A `cpuset` with a single CPU ID, that of the given CPU ID *`cpu`*.

3. If the `cpuset` pointer points to a real `cpuset`, which happens to be the empty set, then that empty set is treated the same as `FBS_CPUSET_CURCPU`. This mirrors what happens in the old, small-SMP interface when a CPU mask is set to the (otherwise illegal) value of 0.

4. If a `cpuset` argument is an input-output parameter, its pointer can still be set to one of the above special values, but in that case the output `cpuset` will not be returned.

Frequency-Based Scheduler Routines

Frequency-Based Scheduler routines provide access to the key features of the scheduler. They enable you to perform such basic operations as:

- configuring a scheduler
- scheduling programs on a scheduler
- setting up and connecting a timing source to a scheduler
- starting, stopping and resuming scheduling on a scheduler
- getting information about scheduled processes
- rescheduling and removing scheduled processes
- disconnecting a timing source
- removing a scheduler

Routine Summary

Frequency-Based Scheduler routines are summarized in Table 6-1. Complete information about each routine is provided under the section "Using Frequency-Based Scheduler Routines."

Table 6-1. Frequency-Based Scheduler C Library Routines

Routine	Page	Description
fbsaccess	6-7	Change permissions for a frequency-based scheduler
fbsattach	6-8	Attach timing source to a frequency-based scheduler
fbsavail	6-9	Query if FBS is configured in the system
fbsconfigure	6-10	Configure a frequency-based scheduler
fbscycle	6-13	Return minor cycle/major frame count
fbsdetach	6-14	Detach timing source from a frequency-based scheduler
fbsdir	6-15	Return list of scheduler keys
fbsgetpid	6-16	Return process ID of scheduled process
fbsgetrtc	6-17	Get real-time clock values
fbsid	6-18	Return the frequency-based scheduler identifier for a key
fbsinfo fbsinfo_big	6-19	Return information for a frequency-based scheduler
fbsintrpt	6-22	Start/stop/resume scheduling
fbsremove	6-23	Remove a frequency-based scheduler

(continued on next page)

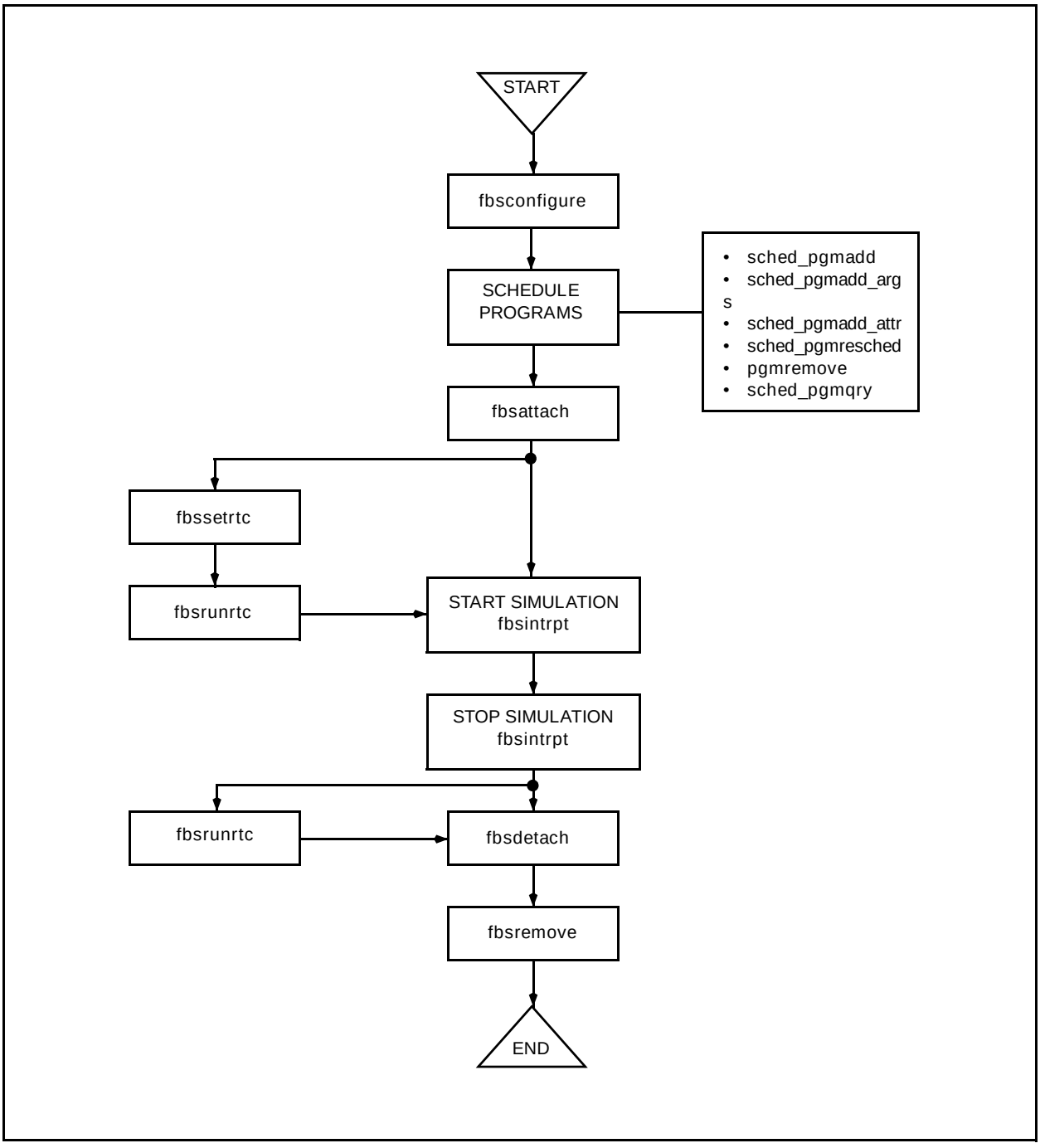
Table 6-1. Frequency-Based Scheduler C Library Routines (Continued)

Routine	Page	Description
fbsresume	6-24	Resume scheduling on a frequency-based scheduler
fbsrunrtc	6-25	Start/stop real-time clock
fbsschedself	6-26	Add a calling process to a frequency-based scheduler
fbssetrtc	6-28	Set real-time clock
fbstrig	6-29	Make a sleeping frequency-based scheduled process runnable
fbswait	6-30	Wait on a frequency-based scheduler
namepid namepid_big	6-31	Return the process ID for a specified process name
nametopid nametopid_big	6-31	Return the process ID for a specified process name on a specified scheduler
pgmremove pgmremove_big	6-33	Remove a process from a frequency-based scheduler
pgmtrigger	6-35	Trigger a process on a frequency-based scheduler
sched_fbsqry sched_fbsqry_big	6-36	Query processes on a frequency-based scheduler
sched_pgm_deadline_query sched_pgm_deadline_query_big	6-39	Query the assigned deadline time
sched_pgm_deadline_test sched_pgm_deadline_test_big	6-42	Check a process for a deadline violation
sched_pgm_set_deadline sched_pgm_set_deadline_big	6-45	Set or clear the process deadline time
sched_pgm_set_soft_overrun_limit sched_pgm_set_soft_overrun_limit_big	6-48	Set soft overrun limit
sched_pgmadd sched_pgmadd_big	6-51	Schedule a process on a frequency-based scheduler
sched_pgmadd_args sched_pgmadd_args_big	6-54	Schedule a process on a frequency-based scheduler with arguments
sched_pgmadd_attr sched_pgmadd_attr_big	6-58	Schedule a process on a frequency-based scheduler with arguments and attributes
sched_pgmqry sched_pgmqry_big	6-61	Query a process
sched_pgmresched, sched_pgmresched_big	6-65	Reschedule a process

C Library Call Sequence

The approximate order in which you might call the routines from an application program is illustrated in Figure 6-1.

Figure 6-1. C Library Call Sequence: Frequency-Based Scheduler



Using Frequency-Based Scheduler Routines

In the sections that follow, the Frequency-Based Scheduler routines contained in the **libccur_fbsched** library are presented in alphabetical order.

fbsaccess – Change Permissions for a Frequency-Based Scheduler

This routine changes the permissions assigned for a selected frequency-based scheduler. It is important to note that the permissions can be changed only by a process that has an effective user ID that is equal to that of the owner/creator of the frequency-based scheduler.

Synopsis

```
#include <fbsched.h>

int fbsaccess(fbs_id, uid, gid, permissions)
int fbs_id, uid, gid, permissions;
```

Parameters

fbs_id — a unique positive integer value representing the identifier for a frequency-based scheduler. This value can be obtained using **fbsid(3)** (page 6-18) or **fbsconfigure(3)** (page 6-10).

Enter MY_FBS to reference the frequency-based scheduler on which the calling process is scheduled without knowing the identifier.

uid — an integer value representing the effective user ID of the specified frequency-based scheduler.

gid — an integer value representing the effective group ID of the specified frequency-based scheduler.

permissions — a bit pattern used to set the permissions associated with the specified frequency-based scheduler. Bit patterns and corresponding permissions are presented in Table 6-2.

Table 6-2. Frequency-Based Scheduler Permissions

Bit Pattern	Permissions
400	Read by user
200	Alter by user
060	Read, alter by group
006	Read, alter by others

Return Value

A return value of 0 indicates that the call has been successful. A return value of -1 indicates that an error has occurred; `errno` is set to indicate the error. Refer to the **fbsaccess(3)** man page for a listing of the types of errors that may occur.

fbsattach – Attach Timing Source to a Frequency-Based Scheduler

This routine attaches a timing source to a frequency-based scheduler or specifies end-of-cycle scheduling. The timing source can be a real-time clock or an edge-triggered interrupt device.

Synopsis

```
#include <fbsched.h>

int  fbsattach(fbs_id, devname)
int  fbs_id;
char *devname;
```

Parameters

fbs_id — a unique positive integer value representing the identifier for a frequency-based scheduler. This value can be obtained using **fbsid(3)** (page 6-18) or **fbsconfigure(3)** (page 6-10).

Enter MY_FBS to reference the frequency-based scheduler on which the calling process is scheduled without knowing the identifier.

devname — a null string or the path name of the device that is to be used as the timing source for the specified scheduler.

If *devname* contains a null string, end-of-cycle scheduling is specified; that is, execution of the processes in the next minor cycle will occur when the last process scheduled to execute in the current minor cycle finishes its execution for that cycle.

If *devname* contains a path name, it may refer to a real-time clock or an edge-triggered interrupt. See Chapter 3 for information about timing source device files.

Return Value

A return value of 0 indicates that the call has been successful. A return value of -1 indicates that an error has occurred; `errno` is set to indicate the error. Refer to the **fbsattach(3)** man page for a listing of the types of errors that may occur.

fbsavail – Query if the Frequency-Based Scheduler is Configured

This routine queries the currently running kernel to determine if the Frequency-Based Scheduler is configured.

Synopsis

```
#include <fbsched.h>

int fbsavail()
```

Return Value

A return value of 1 indicates that FBS is configured. A return value of 0 indicates that FBS is not configured.

fbsconfigure – Configure a Frequency-Based Scheduler

This routine configures a frequency-based scheduler or obtains configuration details for a scheduler that has already been configured.

Indicate the function to be performed using the value assigned to the *cycles* field of the `fbsconfig_ds` structure as follows:

- If the value specified in *cycles* is NOT equal to 0, and if the number of existing frequency-based schedulers is less than the system imposed limit (default = 10) a frequency-based scheduler is created and an identifier is returned in the *fbs_id* field.

  The value assigned to the *reset* field of the `fbsconfig_ds` structure can be used to further qualify the action to be taken when *cycles* is not equal to 0 and the scheduler already exists.

- If the value of *cycles* is equal to 0, the current configuration for the frequency-based scheduler is returned in the `fbsconfig_ds` structure.

  The user-supplied *key* field can be used to identify a frequency-based scheduler.

Synopsis

```
#include <fbsched.h>

int fbsconfigure(fbs_buf)
struct fbsconfig_ds {
    int key;
    int cycles;
    int progs;
    int max;
    int reset;
    int configflg;
    int fbs_id;
} *fbs_buf;
```

Parameters

To create a frequency-based scheduler, you must specify the following parameters as described.

fbs_buf An `fbsconfig_ds` structure that contains the information with which you wish to configure a frequency-based scheduler. The type of information specified in each component is presented below.

key an integer value that may be used later to identify the frequency-based scheduler to be created. Zero is not a valid key value but can be assigned to indicate that a key is not to be used. This is an optional identifier since the *fbs_id* value returned here can always be used to reference a frequency-based scheduler.

cycles an integer value indicating the number of minor cycles that compose a frame on the specified scheduler.

progs an integer value indicating the maximum number of programs that can be scheduled to execute during one minor cycle.

max an integer value indicating the maximum number of programs that can be scheduled on the specified scheduler at one time. This value must be less than or equal to the product that is obtained by multiplying the values specified for the *cycles* and *progs* parameters.

reset an integer value indicating whether or not processes currently scheduled on the specified scheduler are to be killed before the scheduler is reconfigured. Acceptable values and corresponding results are as follows:

<0 Kill and remove all processes currently scheduled on the specified scheduler
0 Ignore all processes currently scheduled on the specified scheduler
>0 Remove all processes currently scheduled on the specified scheduler

configflg an integer value indicating the permissions to be assigned to the specified scheduler.

fbs_id a unique positive integer value that is returned by **`fbsconfigure`** and represents the identifier for the specified frequency-based scheduler. It is important to note that this identifier is required by most of the library routines for the Frequency-Based Scheduler and the Performance Monitor.

To obtain information for an existing frequency-based scheduler, you must specify the following parameters as described.

fbs_buf An `fbsconfig_ds` structure to which information is returned for an existing frequency-based scheduler. The types of information specified or returned in each component is presented below.

key the user-supplied value specified when the scheduler was created. This value can be used to identify the frequency-based scheduler for which configuration information is to be returned.

cycles the integer value 0 indicating that current configuration information for the specified scheduler is to be returned. An integer value indicating the number of minor cycles that compose a frame on the specified scheduler is returned to this component.

progs the maximum number of programs that can be scheduled to run during one minor cycle on the specified scheduler.

max the maximum number of programs that can be scheduled on the specified scheduler at one time.

configflg	the permissions assigned to the specified scheduler.
fbs_id	a unique positive integer value representing the identifier for the specified frequency-based scheduler.

Return Value

A return value of 0 indicates that the call has been successful. A return value of -1 indicates that an error has occurred; `errno` is set to indicate the error. Refer to the **`fbsconfigure(3)`** man page for a listing of the types of errors that may occur.

fbscycle – Return Minor Cycle/Major Frame Count

This routine obtains the current minor cycle and major frame count values for a frequency-based scheduler. These values enable you to determine the progress of a simulation.

Synopsis

```
#include <fbsched.h>

int fbscycle(fbs_id, cycle_buf)
int fbs_id;
struct fbscycle_ds {
    int ccycle;
    int cframe;
} *cycle_buf;
```

Parameters

fbs_id — a unique positive integer value representing the identifier for a frequency-based scheduler. This value can be obtained using **fbsid(3)** (page 6-18) or **fbsconfigure(3)** (page 6-10).

Enter MY_FBS to reference the frequency-based scheduler on which the calling process is scheduled without knowing the identifier.

cycle_buf — refers to an `fbscycle_ds` structure to which **fbscycle** will return integer values indicating the current minor cycle and major frame for the specified scheduler. The *ccycle* component will contain the number of the cycle. The *cframe* component will contain the number of the frame.

Return Value

A return value of `0` indicates that the call has been successful. A return value of -1 indicates that an error has occurred; `errno` is set to indicate the error. Refer to the **fbscycle(3)** man page for a listing of the types of errors that may occur.

fbsdetach – Detach Timing Source from a Frequency-Based Scheduler

This routine detaches the currently attached timing source from a frequency-based scheduler or disables end-of-cycle scheduling. If the timing source is a real-time clock, it is recommended that you stop the clock prior to invoking this routine. You can do so by making a call to **fbsrunrtc** (see page 6-25 for an explanation of this routine).

Synopsis

```
#include <fbsched.h>

int fbsdetach(fbs_id)
int fbs_id;
```

Parameters

fbs_id — a unique positive integer value representing the identifier for a frequency-based scheduler. This value can be obtained using **fbsid(3)** (page 6-18) or **fbsconfigure(3)** (page 6-10).

Enter MY_FBS to reference the frequency-based scheduler on which the calling process is scheduled without knowing the identifier.

Return Value

A return value of 0 indicates that the call has been successful. A return value of -1 indicates that an error has occurred; `errno` is set to indicate the error. Refer to the **fbsdetach(3)** man page for a listing of the types of errors that may occur.

fbsdir – Return a List of Scheduler Keys

This routine returns the list of keys associated with the frequency-based schedulers currently configured in the system.

Synopsis

```
#include <fbsched.h>

fbsdir(key_t *keylist, size_t *keycnt)
```

Parameters

keylist — a pointer to a memory reservation to which the keys of frequency-based schedulers configured on the system is returned. The area must large enough to store the number of `key_t` type elements specified by *keycnt*.

keycnt — the number of frequency-based scheduler keys configured on the system. If this number is less than the number of schedulers configured on the system, an error is returned.

Return Value

A return value of 0 indicates that the call has been successful. A return value of -1 indicates that an error has occurred; `errno` is set to indicate the error. Refer to the **`fbsdir(3)`** man page for a listing of the types of errors that may occur.

fbsgetpid – Return Process ID for a Scheduled Process

This routine returns the process ID (pid) of a process currently scheduled on a frequency-based scheduler.

Synopsis

```
#include <fbsched.h>

int fbsgetpid(fbs_id, *name_ptr, fpid)
int  fbs_id, fpid;
char *name_ptr;
```

Parameters

fbs_id — a unique positive integer value representing the identifier for a frequency-based scheduler. This value can be obtained using **fbsid(3)** (page 6-18) or **fbsconfigure(3)** (page 6-10).

Enter MY_FBS to reference the frequency-based scheduler on which the calling process is scheduled without knowing the identifier.

name_ptr — a pointer to the path name that identifies the process. This argument is currently ignored and should be NULL. Use **namepid(3)** or **nametopid(3)** to obtain a process ID by specifying the process' name.

fpid — an integer value providing the unique frequency-based scheduler process identifier for the process. This value is obtained when you make a call to **sched_pgmadd** (see page 6-51 for an explanation of this routine).

Return Value

Upon successful completion, **fbsgetpid** returns the process ID. A return value of -1 indicates that an error has occurred; errno is set to indicate the error. Refer to the **fbsgetpid(3)** man page for a listing of the types of errors that may occur.

fbsgetrtc – Obtain Current Values for a Real-Time Clock

This routine obtains the current count and resolution values for the real-time clock attached to a specified frequency-based scheduler.

Synopsis

```
#include <fbsched.h>

int fbsgetrtc(fbs_id, count, resolution)
int fbs_id,*count, *resolution;
```

Parameters

fbs_id — a unique positive integer value representing the identifier for a frequency-based scheduler. This value can be obtained using **fbsid(3)** (page 6-18) or **fbsconfigure(3)** (page 6-10).

Enter MY_FBS to reference the frequency-based scheduler on which the calling process is scheduled without knowing the identifier.

count — the current number of clock counts per minor cycle. This value can range from 1 to 65535.

resolution — an integer value indicating the current duration in microseconds of one clock count. This value is one of the following: 1, 10, 100, 1000, or 10000.

Return Value

A return value of 0 indicates that the call has been successful. A return value of -1 indicates that an error has occurred; `errno` is set to indicate the error. Refer to the **fbsgetrtc(3)** man page for a listing of the types of errors that may occur.

fbsid – Return the Frequency-Based Scheduler Identifier for a Key

This routine obtains the frequency-based scheduler identifier associated with a particular user-specified key. The key must match the key that was specified when the scheduler was created via a call to **fbsconfigure(3)**.

Synopsis

```
#include <fbsched.h>

int fbsid(fbs_key)
int fbs_key;
```

Parameters

fbs_key — an integer value identifying the frequency-based scheduler. This value is the same value specified for *key* when the scheduler was created using **fbsconfigure** (see page 6-10 for an explanation of this routine).

Return Value

Upon successful completion, **fbsid** returns an integer value representing the unique frequency-based scheduler identifier associated with the key. A return value of -1 indicates that an error has occurred; `errno` is set to indicate the error. Refer to the **fbsid(3)** man page for a listing of the types of errors that may occur.

fbsinfo, fbsinfo_big – Return Information for a Frequency-Based Scheduler

These routines obtain information related to a selected frequency-based scheduler that cannot be obtained by invoking other routines (for example, **sched_fbsqry**, **sched_pgmqry**).

Information includes:

- user and group IDs of the owner and the creator of the scheduler
- permissions assigned for the scheduler
- key associated with the scheduler's identifier
- total number of overruns for all processes on the scheduler
- CPUs that are active in the system
- CPUs on which performance monitoring has been enabled
- FBS enabled flag
- path name of the device that has been attached to the scheduler

These two routines are identical except for their treatment of CPU masks. For details, see "The Big-SMP FBS Interface" on page 6-2.

Synopsis

```
#include <fbsched.h>

int fbsinfo(fbs_id, info_buf, devname)
int fbs_id;
struct fbsinfo_ds {
    int uid;
    int gid;
    int cuid;
    int cgid;
    int mode
    int key;
    int flags;
    int devid;
    int overruns;
    int cpuactive;
    int pm_cpuactive;
    int enabled;
    int filler[29]
} *info_buf;
char *devname;

int fbsinfo_big(fbs_id, info_buf, devname)
int fbs_id;
struct fbsinfo_big_ds {
    int uid;
    int gid;
    int cuid;
    int cgid;
    int mode
    int key;
    int flags;
```

```
    int devid;
    int overruns;
    int enabled;
    int ifill[9];
    cpuset_t *cpuset_active;
    cpuset_t *pm_cpuset_active;
    int vfill[9]
} *info_buf;
char *devname;
```

Parameters

fbs_id a unique positive integer value representing the identifier for a frequency-based scheduler. This value can be obtained using **fbsid(3)** (page 6-18) or **fbsconfigure(3)** (page 6-10).

Enter MY_FBS to reference the frequency-based scheduler on which the calling process is scheduled without knowing the identifier.

info_buf An fbsinfo_ds or fbsinfo_big_ds structure to which **fbsinfo** or **fbsinfo_big**, respectively, returns information about the specified scheduler. The information returned in each component of the structure is presented below.

uid	owner's user ID
gid	owner's group ID
cuid	creator's user ID
cgid	creator's group ID
mode	access modes
key	key
flags	flags word
devid	reserved for future use
overruns	total number of hard overruns for all processes on the scheduler
cpuactive	mask of CPUs active in the system
pm_cpuactive	mask of CPUs on which performance monitoring has been enabled
cpuset_active	pointer to a mask of CPUs, in **cpuset(3)** format, in the system. If this is the NULL pointer, then no cpuset information is returned.
pm_cpuset_active	pointer to a mask of CPUs, in **cpuset(3)** format, on which performance monitoring has been enabled. If this is the NULL pointer, then no cpuset information is returned.
enabled	FBS enabled flag

devname — a variable to which **fbsinfo** and **fbsinfo_big** returns the path name of the device being used as the timing source for the specified frequency-based scheduler. If end-of-cycle scheduling is specified, *devname* contains a null string.

Return Value

A return value of 0 indicates that the call has been successful. A return value of -1 indicates that an error has occurred; `errno` is set to indicate the error. Refer to the **fbsinfo(3)** and **fbsinfo_big(3)** man pages for a listing of the types of errors that may occur.

fbsintrpt – Start/Stop/Resume Scheduling

This routine starts, stops or resumes scheduling on a frequency-based scheduler. If you invoke this routine to start scheduling, the minor cycle, major frame, and overrun count values are reset. If you invoke it to resume scheduling, these values are not reset.

Prior to invoking **fbsintrpt**, you must have invoked **fbsattach** to specify end-of-cycle scheduling or attach a timing source to the frequency-based scheduler on which you are starting scheduling (see page 6-8 for an explanation of **fbsattach**).

If you have specified a real-time clock as the timing source, scheduling will not begin until you have set and started the clock (see pages 6-28 and 6-25 for explanations of **fbssetrtc** and **fbsrunrtc**, respectively).

Synopsis

```
#include <fbsched.h>

int fbsintrpt(fbs_id, intrflag)
int fbs_id, intrflag;
```

Parameters

fbs_id — a unique positive integer value representing the identifier for a frequency-based scheduler. This value can be obtained using **fbsid(3)** (page 6-18) or **fbsconfigure(3)** (page 6-10).

Enter MY_FBS to reference the frequency-based scheduler on which the calling process is scheduled without knowing the identifier.

intrflag — an integer value indicating whether scheduling of processes on the specified scheduler is to be started, stopped or resumed. Acceptable values and corresponding results are:

<0	Start scheduling (waking up) processes with the initial frame, cycle, and overrun count values set to zero
0	Stop scheduling processes and save the count values for the current frame and cycle
>0	Resume scheduling processes with the frame, cycle, and overrun count values set to the values that were saved when the scheduler was last stopped

Return Value

A return value of 0 indicates that the call has been successful. A return value of -1 indicates that an error has occurred; errno is set to indicate the error. Refer to the **fbsintrpt(3)** man page for a listing of the types of errors that may occur.

fbsremove – Remove a Frequency-Based Scheduler

This routine removes a frequency-based scheduler and frees the data structure associated with it. Prior to invoking **fbsremove**, you must ensure that the timing source is detached from the scheduler (see page 6-14 for information on the use of **fbsdetach**).

It is important to note that **fbsremove** removes all processes scheduled on the specified scheduler. It is recommended, however, that you remove all scheduled processes prior to invoking **fbsremove**. You can do so by making a call to **pgmremove** (see page 6-33 for information on the use of this routine).

Note that to remove a frequency-based scheduler, the calling process must have an effective user ID that is equal to that of the owner/creator of the scheduler.

Synopsis

```
#include <fbsched.h>

int fbsremove(fbs_id, ab)
int fbs_id, ab;
```

Parameters

fbs_id — a unique positive integer value representing the identifier for a frequency-based scheduler. This value can be obtained using **fbsid(3)** (page 6-18) or **fbsconfigure(3)** (page 6-10).

Enter MY_FBS to reference the frequency-based scheduler on which the calling process is scheduled without knowing the identifier.

ab — an integer value indicating the manner in which processes scheduled on the scheduler are to be handled. Acceptable values and corresponding results are:

<0	Kill and remove all processes currently scheduled on the specified scheduler
≥0	Remove all processes currently scheduled on the specified scheduler

Return Value

A return value of 0 indicates that the call has been successful. A return value of -1 indicates that an error has occurred; `errno` is set to indicate the error. Refer to the **fbsremove(3)** man page for a listing of the types of errors that may occur.

fbsresume – Resume Scheduling on a Frequency-Based Scheduler

The **fbsresume** library routine resumes scheduling of processes on a frequency-based scheduler at the specified minor cycle, major frame, and overrun count.

Note that to resume scheduling of processes on a frequency-based scheduler, the calling process must have alter permission for the scheduler.

If you wish to resume scheduling of processes on a frequency-based scheduler without altering the scheduler's current frame, cycle, and overrun values, it is recommended that you use the **fbsintrpt(3)** routine (see page 6-22 for an explanation of this routine).

CAUTION

The **fbsresume** routine clears Performance Monitor values for all processes scheduled on the specified scheduler. Changing the frame and cycle count for the scheduler causes the values that are being maintained by the Performance Monitor to be inaccurate.

Synopsis

```
#include <fbsched.h>

int fbsresume(fbs_id, frame, cycle, overruns)
int fbs_id, frame, cycle, overruns;
```

Parameters

fbs_id — a unique positive integer value representing the identifier for a frequency-based scheduler. This value can be obtained using **fbsid(3)** (page 6-18) or **fbsconfigure(3)** (page 6-10).

Enter MY_FBS to reference the frequency-based scheduler on which the calling process is scheduled without knowing the identifier.

frame — an integer value indicating the major frame in which you wish scheduling of processes to be resumed on the specified scheduler.

cycle — an integer value indicating the minor cycle in which you wish scheduling of processes to be resumed on the specified scheduler.

This value can range from 0 to the total number of minor cycles per frame minus 1. The total number of minor cycles per frame was specified when the scheduler was created by making a call to **fbsconfigure** (see page 6-10).

overruns — an integer value indicating the value to which you wish the overrun count to be set when scheduling resumes on the specified scheduler.

Specify the value -1 if you do not wish to change the overrun count.

Return Value

A return value of `0` indicates that the call has been successful. A return value of -1 indicates that an error has occurred; `errno` is set to indicate the error. Refer to the **fbsresume(3)** man page for a listing of the types of errors that may occur.

fbsrunrtc – Start/Stop a Real-Time Clock

This routine starts or stops the counting of a real-time clock that has been attached to a frequency-based scheduler.

Synopsis

```
#include <fbsched.h>

int fbsrunrtc(fbs_id, runflag)
int fbs_id, runflag;
```

Parameters

fbs_id — a unique positive integer value representing the identifier for a frequency-based scheduler. This value can be obtained using **fbsid(3)** (page 6-18) or **fbsconfigure(3)** (page 6-10).

Enter MY_FBS to reference the frequency-based scheduler on which the calling process is scheduled without knowing the identifier.

runflag — an integer value indicating whether the real-time clock is to be started or stopped. A nonzero value indicates that the clock is to be started. A zero value indicates that the clock is to be stopped.

Return Value

A return value of 0 indicates that the call has been successful. A return value of -1 indicates that an error has occurred; `errno` is set to indicate the error. Refer to the **fbsrunrtc(3)** man page for a listing of the types of errors that may occur.

fbsschedself – Add a Calling Process to a Frequency-Based Scheduler

This routine schedules the calling process on a frequency-based scheduler.

It is important to note that **fbsschedself** does not allow a process to set its scheduling policy and priority or its CPU bias. These tasks must be performed prior to invoking **fbsschedself**.

Note that you cannot use this routine to add **/idle** to a frequency-based scheduler.

To schedule the calling process on a frequency-based scheduler, the calling process must have alter permission for the scheduler.

To change the scheduling policy or priority of a frequency-based scheduled process, use the **sched_pgmresched** routine (see page 6-65 for an explanation of this routine).

Synopsis

```
#include <fbsched.h>

int fbsschedself(fbs_id, name, sched_buf)
int fbs_id;
char *name;
struct fbssched_buf {
    int version;
    int param;
    int period;
    int cycle;
    int ab;
    int fpid;
} *sched_buf;
```

Parameters

fbs_id a unique positive integer value representing the identifier for a frequency-based scheduler. This value can be obtained using **fbsid(3)** (page 6-18) or **fbsconfigure(3)** (page 6-10).

name a standard path name or arbitrary content identifying the program associated with the calling process. A full or relative path name of up to 1023 characters can be specified.

sched_buf a `fbssched_buf` structure that contains the scheduling information for the process that is to be scheduled. The type of information specified in each component is presented below.

- *version* an integer value indicating the version of *sched_buf* that is being passed to **fbsschedself**. Specify the symbolic constant **FBSSCHED_BUF_V1**, which is defined in **<fbsched.h>** for this purpose.
- *param* an integer value to be passed to a process that is scheduled on a frequency-based scheduler. This value can be retrieved by the frequency-based scheduled process through a call to **sched_pgmqry** (see page 6-61 for an explanation of this routine).

period — an integer value indicating the frequency with which the calling process is to be awakened in each major frame. A period of 1 indicates that the calling process is to be awakened every minor cycle; a period of 2 indicates that it is to be awakened once every two minor cycles and so on.

This value can range from 1 to the number of minor cycles that compose a frame on the specified scheduler as defined in a call to **`fbsconfigure`** (see page 6-10 for an explanation of this routine).

cycle — an integer value indicating the first minor cycle in which the calling process is scheduled to be awakened in each frame. This value can range from 0 to the total number of minor cycles per frame minus 1. The total number of minor cycles per frame is specified in a call to **`fbsconfigure`** (see page 6-10 for an explanation of this routine).

ab — an integer value indicating whether or not the scheduler should be stopped in the event that the calling process causes a frame overrun. A nonzero value indicates that the scheduler will be stopped.

fpid — an integer value returned by **`fbsschedself`** that is the unique frequency-based scheduler process identifier for the scheduled process.

Return Value

A return value of 0 indicates that the call has been successful. A return value of -1 indicates that an error has occurred; `errno` is set to indicate the error. Refer to the **`fbsschedself(3)`** man page for a listing of the types of errors that may occur.

fbssetrtc – Set a Real-Time Clock

This routine establishes the duration of a minor cycle by setting the count and the resolution values for a real-time clock to be used as an FBS timing device.

Synopsis

```
#include <fbsched.h>

int fbssetrtc(fbs_id, count, resolution)
int fbs_id, count, resolution;
```

Parameters

fbs_id — a unique positive integer value representing the identifier for a frequency-based scheduler. This value can be obtained using **fbsid(3)** (page 6-18) or **fbsconfigure(3)** (page 6-10).

Enter MY_FBS to reference the frequency-based scheduler on which the calling process is scheduled without knowing the identifier.

count — an integer value indicating the number of clock counts per minor cycle. This value can range from 2 to 65535.

resolution — an integer value indicating the duration in microseconds of one clock count. This value must be one of the following: 1, 10, 100, 1000, or 10000.

NOTE: Although a *count* of 1 cannot be used, a timing interval equal to a *resolution* value can be set by using the next lower resolution value and a count of 10; e.g., a resolution of 1,000 and count value of 10 results in a timing interval of 10,000 microseconds.

Return Value

A return value of 0 indicates that the call has been successful. A return value of -1 indicates that an error has occurred; `errno` is set to indicate the error. Refer to the **fbssetrtc(3)** man page for a listing of the types of errors that may occur.

fbstrig – Make a Sleeping Frequency-Based Scheduler Process Runnable

This routine makes a process scheduled on a frequency-based scheduler runnable if it is in the **fbswait(3)** sleep state. In addition, a context switch can be forced on the processor on which the process is executing.

Synopsis

```
#include <fbsched.h>

int fbstrig (int fpid, int tgrflg);
int fpid, tgrflg;
```

Parameters

fpid — an integer value providing the unique frequency-based scheduler process identifier for the sleeping process. This value is obtained when you make a call to **sched_pgmadd** (see page 6-51 for an explanation of this routine).

tgrflg — an integer value indicating whether or not a context switch is to be forced on the processor on which the awakened process is executing. A nonzero value indicates that a context switch is to be forced.

Return Value

A return value of 0 indicates that the call has been successful. A return value of -1 indicates that an error has occurred; `errno` is set to indicate the error. Refer to the **fbstrig(3)** man page for a listing of the types of errors that may occur.

fbswait – Wait on a Frequency-Based Scheduler

This routine enables a process that is scheduled on a frequency-based scheduler to sleep until its next scheduled minor cycle.

When the scheduled process does not call this service by its next scheduled minor cycle, either a soft overrun or hard overrun is incurred.

A soft overrun occurs if the per-process count of consecutively missed scheduled minor cycles does not reach or exceed the per-process soft overrun limit. When a soft overrun occurs, the process returns immediately from the **fbswait** call instead of blocking to wait for the next scheduled minor cycle.

When the count of consecutively missed scheduled minor cycles reaches or exceeds the per-process overrun limit, a hard overrun occurs. In this case, the process is blocked in **fbswait** until the next scheduled minor cycle.

A process' consecutive soft overrun limit may be changed from the default value of 0 using **sched_pgm_set_soft_overrun_limit(3)**. The hard overrun count, which can be read using **pmqrypgm(3)**, and the soft overrun count, which can be read using **sched_pgm_soft_overrun_query(3)**, indicate if the process is actually running at its assigned frequency.

When the scheduled process is subject to a deadline and the scheduled process calls this service after its deadline time has passed, a deadline violation is detected, and the scheduler may be halted.

Synopsis

```
#include <fbsched.h>
int fbswait(void)
```

Return Value

A return value of 0 indicates that the process has been awakened by the frequency-based scheduler. A return value of 1 indicates that the process has been awakened by **fbstrig(3)**. A return value of 2 indicates that the process did not sleep because the kernel detected a soft overrun and is allowing the process to attempt to recover from it. A return value of -1 indicates that an error has occurred; `errno` is set to indicate the error. Refer to the **fbswait(3)** man page for a listing of the types of errors that can occur.

nametopid, namepid, nametopid_big, namepid_big – Return the Process ID for a Specified Process Name

The **namepid** and **namepid_big** routines return the process ID (pid) of the specified process. The **nametopid** and **nametopid_big** routines perform the same function but allows the search to be restricted to processes scheduled on a particular frequency-based scheduler.

The 'big' routines are identical to their non-big counterparts except for their treatment of CPU masks. For a complete description of the differences, see "The Big-SMP FBS Interface" on page 6-2.

Synopsis

```
#include <fbsched.h>

int nametopid(name, fbs_key, cpu)
int namepid(name, cpu)
int  fbs_key, cpu;
char *name;

int nametopid_big(name, fbs_key, cpuset)
int namepid_big(name, cpuset)
int  fbs_key;
char *name;
cpuset_t *cpuset;
```

Parameters

name — the path name that identifies the process

fbs_key — an integer value identifying the frequency-based scheduler. This value is the same value specified for *key* when the scheduler was created using **fbsconfigure** (see page 6-10 for an explanation of this routine).

cpu — an integer value indicating the processor(s) to be used in conjunction with the value of the *name* parameter to identify the process. Acceptable values and corresponding results are:

0	the first process named by *name* currently running on the processor from which the call is made is requested
-1	the first process named by *name* currently running on any processor is requested
bit mask	those processors with (*cpu* & (1<<*i*)) set (where *i* is an integer ranging from 0 to 15 and representing a CPU) are requested

cpuset — a pointer that directly or indirectly indicates the set of processors, to be used in conjunction with *name*, to identify the process whose pid is to be returned. The pointer may either point to a valid `cpuset`, as created by **cpuset_alloc(3)** and filled with some set of CPU IDs by other **cpuset(3)** operations, or it may take on one of the following special pointer values:

`FBS_CPUSET_CURCPU` the first process by *name* currently running on the processor from which the call is made.

`FBS_CPUSET_ALLCPUS` the first process named by *name* running on any processor.

`FBS_CPUSET_ONECPU(`*cpu*`)` the first process named by *name* running on the processor with a CPU ID of *cpu*.

If *cpuset* points to a **`cpuset(3)`** that is empty, then the behavior is as described for `FBS_CPUSET_CURCPU`.

Return Value

On success these functions return the process ID. On failure these functions return -1 and `errno` is set to indicate the error. Refer to the **`nametopid(3)`** and **`nametopid_big(3)`** man pages for a listing of the types of errors that can occur

pgmremove, pgmremove_big – Remove a Process from a Frequency-Based Scheduler

These routines remove a process from a frequency-based scheduler. Identify the process to remove by using one of the following methods:

- Specify the name of the process and the CPU on which it is scheduled.
- Specify the process' frequency-based scheduler process identifier.
- Specify the name of the process, the CPU on which it is scheduled, and its frequency-based scheduler process identifier.

NOTE

The only method that can be used to identify a process that has been scheduled multiple times on the same CPU is to specify its frequency-based scheduler process identifier.

These two routines are identical except for their treatment of CPU masks. For details, see "The Big-SMP FBS Interface" on page 6-2.

Synopsis

```
#include <fbsched.h>

int pgmremove(fbs_id, name, cpu, fpid, ab)
int  fbs_id, cpu, fpid, ab;
char *name;

int pgmremove_big(fbs_id, name, cpuset, fpid, ab)
int  fbs_id, fpid, ab;
char *name;
cpuset_t *cpuset;
```

Parameters

fbs_id — a unique positive integer value representing the identifier for a frequency-based scheduler. This value can be obtained using **fbsid(3)** (page 6-18) or **fbsconfigure(3)** (page 6-10).

Enter MY_FBS to reference the frequency-based scheduler on which the calling process is scheduled without knowing the identifier.

name — a standard path name identifying the process to be removed from the specified scheduler. A full or relative path name of up to 1024 characters can be specified. If this variable contains the null string, you must provide the frequency-based scheduler process identifier in the *fpid* parameter.

cpu — an integer value indicating the processor(s) to be used in conjunction with the value of the *name* parameter to identify the process to be removed from the specified scheduler. Acceptable values and corresponding results are:

0	the first process named by *name* currently running on the processor from which the call is made is removed
-1	the first process named by *name* currently running on any processor is removed

bit mask	if (*cpu* & (1<<*i*)) is set (where *i* is an integer ranging from 0 to 15 and representing a CPU) and it is the only bit set, the first process named by *name* that is running on CPU *i* is removed
	
	If (*cpu* & (1<<*i*)) is set and it is not the only bit set, the first process named by *name* currently running on any of the selected CPUs is removed.

cpuset a pointer that directly or indirectly indicates the set of processors, to be used in conjunction with *name*, to identify the process to be removed from the specified scheduler. The pointer may either point to a valid `cpuset`, as created by **`cpuset_alloc(3)`** and filled with some set of CPU IDs by other **`cpuset(3)`** operations, or it may take on one of the following special pointer values:

`FBS_CPUSET_CURCPU`	the first process found that is named by *name* and is currently running on the processor from which the call is made.
`FBS_CPUSET_ALLCPUS`	the first process found that is named by *name* that is running on any processor.
`FBS_CPUSET_ONECPU(`*cpu*`)`	the first process that is named by *name* and is running on the processor with a CPU ID of *cpu*.

If *cpuset* points to a **`cpuset(3)`** that is empty, then the behavior is as described for `FBS_CPUSET_CURCPU`.

fpid an integer value providing the unique frequency-based scheduler process identifier for the process to be removed. This value is obtained when you make a call to **`sched_pgmadd`** (see page 6-51 for an explanation of this routine). This value must be -1 if you choose to identify the program to be removed only by specifying *name* and *cpu*.

ab a flag that contains an integer value indicating the manner in which the specified process is be removed from the specified scheduler. A positive value indicates that the process is to be removed from the scheduler but allowed to continue executing. A negative value indicates that the process is to be removed from the scheduler and terminated.

Return Value

A return value of 0 indicates that the call has been successful. A return value of -1 indicates that an error has occurred; `errno` is set to indicate the error. Refer to the **`pgmremove(3)`** and **`pgmremove_big(3)`** man pages for a listing of the types of errors that can occur.

pgmtrigger – Trigger a Process on a Frequency-Based Scheduler

This routine enables a process to wake a process that is in the **`fbswait`** sleep state. It is important to note that the calling process does not have to be scheduled on a frequency-based scheduler; the target process must be.

Synopsis

```
#include <fbsched.h>

int pgmtrigger(fpid, tgrflg)
int fpid, tgrflg;
```

Parameters

fpid — an integer value providing the unique frequency-based scheduler process identifier for the sleeping process. This value is obtained when you make a call to **`sched_pgmadd`** (see page 6-51 for an explanation of this routine).

tgrflg — an integer value indicating whether or not a context switch is to be forced on the processor on which the awakened process is executing. A nonzero value indicates that a context switch is to be forced.

Return Value

A return value of 0 indicates that the call has been successful. A return value of -1 indicates that an error has occurred; `errno` is set to indicate the error. Refer to the **`pgmtrigger(3)`** man page for a listing of the types of errors that may occur.

sched_fbsqry, sched_fbsqry_big – Query Processes on a Frequency-Based Scheduler

These routines obtain information about processes scheduled on a frequency-based scheduler. Information is returned for all processes scheduled on the user-specified processor(s). Information provided for each process includes the following:

- a mask of the CPU(s) on which the process can execute
- the frequency-based scheduler process identifier
- the policy under which the process has been scheduled
- the scheduling priority
- the period (the number of minor cycles indicating the frequency with which the process is awakened in each major frame)
- the starting base cycle (the first minor cycle in which the process is scheduled to be awakened in each major frame)
- the value of the "halt on overrun" flag
- the current state of the process

These two routines are identical except for their treatment of CPU masks. For details, see "The Big-SMP FBS Interface" on page 6-2.

Synopsis

```
#include <fbsched.h>

int sched_fbsqry(fbs_id, cpu, fbs_buf, buf_cnt)
int fbs_id, cpu, buf_cnt;
struct pgm2_ds {
    char *name_ptr;
    int cpu;
    int fpid;
    int cid;
    int prior;
    int param;
    int period;
    int cycle;
    int halt;
    int status;
} *fbs_buf;

int sched_fbsqry_big(fbs_id, cpuset, fbs_buf, buf_cnt)
int fbs_id, cpu, buf_cnt;
struct pgm2_big_ds {
    char *name_ptr;
    cpuset_t *cpuset;
    void *vfill[9]
    int fpid;
    int cid;
    int prior;
    int param;
    int period;
    int cycle;
```

```
    int halt;
    int status;
    int ifill[9];
} *fbs_buf;
```

Parameters

fbs_id — a unique positive integer value representing the identifier for a frequency-based scheduler. This value can be obtained using **fbsid(3)** (page 6-18) or **fbsconfigure(3)** (page 6-10).

Enter MY_FBS to reference the frequency-based scheduler on which the calling process is scheduled without knowing the identifier.

cpu — an integer value indicating the processor(s) for which scheduling information is to be obtained. The acceptable values and corresponding results are:

0	scheduling information for processes executing on the processor from which the call is made is returned
-1	scheduling information for all processes on the scheduler is returned
bit mask	if (*cpu* & (1<<*i*)) is set (where *i* is an integer ranging from 0 to 15 and representing a CPU), scheduling information for processes executing on CPU *i* is returned

cpuset — a pointer that directly or indirectly indicates the set of processors, to be used in conjunction with *name_ptr*, to identify the set of FBS-scheduled processes about which information is to be obtained. The pointer may either point to a valid cpuset, as created by **cpuset_alloc(3)** and filled with some set of CPU IDs by other **cpuset(3)** operations, or it may take on one of the following special pointer values:

FBS_CPUSET_CURCPU	the set of process found that are named by *name_ptr* and currently running on the processor from which the call is made.
FBS_CPUSET_ALLCPUS	the set of process found that are named by *name_ptr* that arerunning on any processor.
FBS_CPUSET_ONECPU(*cpu*)	the set of process that are named by *name_ptr* and are running on the processor with a CPU ID of *cpu*.

If *cpuset* points to a **cpuset(3)** that is empty, then the behavior is as described for FBS_CPUSET_CURCPU.

fbs_buf — a pointer to an array of pgm2_ds or pgm2_big_ds structures to which **sched_fbsqry** or **sched_fbsqry_big**, respectively, returns scheduling information for each process on the processor(s) specified with the *cpu* or *cpuset* parameter. The type of information returned in each component of the structure for a single process is presented below.

name_ptr a pointer to a variable that contains a path name identifying the process for which information is returned. Note that the application must deallocate the memory associated with *name_ptr* for all valid entries using **free** (see **malloc(3)**).

cpu a bit mask indicating the processor(s) on which the process can execute

cpuset a pointer to the cpuset into which the set of processors that the process can execute on is returned. If this is the NULL pointer, then no cpuset information is returned about the process.

fpid the process' frequency-based scheduler process identifier

cid the process' scheduling policy: SCHED_FIFO, SCHED_RR or SCHED_OTHER

prior an integer value indicating the specified process' scheduling priority

param the process' initiation parameter (optional)

period the number of minor cycles indicating the frequency with which the process is to be awakened in each major frame

cycle the first minor cycle in which the process is scheduled to be awakened in each major frame (starting base cycle)

halt the value of the "halt on overrun" flag. A nonzero value indicates that the flag is set. A value of zero indicates that the flag is not set.

status the current state of the process as defined in **<fbsched.h>**.

buf_cnt an integer value indicating the number of structures in the array to which *fbs_buf* points.

Return Value

A return value of 0 indicates that the call has been successful. A return value of -1 indicates that an error has occurred; errno is set to indicate the error. Refer to the **sched_fbsqry(3)** or **sched_fbsqry_big(3)** man pages for a listing of the types of errors that may occur.

sched_pgm_deadline_query, sched_pgm_deadline_query_big – Query the Assigned Deadline Time for a Process

These routines query the deadline parameters for a currently scheduled process on the frequency-based scheduler.

It is important to note that this function will not detect new deadline violations. Use **`sched_pgm_deadline_test(3)`** to trigger the detection of new deadline violations.

These two routines are identical except for their treatment of CPU masks. For details, see "The Big-SMP FBS Interface" on page 6-2.

Synopsis

```
#include <fbsched.h>

int sched_pgm_deadline_query (fbs_id, deadline_info_buf)
int fbs_id ;
struct deadline_info_ds
{
    char *name_ptr;
    int cpu;
    int fpid;
    int halt;
    deadline_kind kind;
    deadline_origin origin;
    struct timespec deadline;
    int total_violations;
} *deadline_info_buf ;

int sched_pgm_deadline_query_big (fbs_id, deadline_info_buf)
int fbs_id ;
struct deadline_info_big_ds
{
    char *name_ptr;
    cpuset_t cpuset;
    void *vfill[9];
    int fpid;
    int halt;
    int ifill[9];
    deadline_kind kind;
    deadline_origin origin;
    struct timespec deadline;
    int total_violations;
} *deadline_info_buf ;
```

Parameters

fbs_id — a unique positive integer value representing the identifier for a frequency-based scheduler. This value can be obtained using **`fbsid(3)`** (page 6-18) or **`fbsconfigure(3)`** (page 6-10).

deadline_info_buf — a pointer to a `deadline_info_ds` or `deadline_info_big_ds` structure that contains the attributes of any deadline assigned to the process. The structure definition is:

name_ptr a pointer to the path name that identifies the process. If the path name is the null string, the *fpid* field must be given.

cpu an integer value indicating the processors on which the specified program can be scheduled to run. Acceptable values and corresponding results are:

- 0 — the program pointed to by *name_ptr* can be scheduled on the processor from which the call is made.
- -1 — the program pointed to by *name_ptr* can be scheduled on any processor.
- bit mask — if (*cpu* & (1<<*i*)) is set (where *i* is an integer ranging from 0 to 15 and representing a CPU), the program pointed to by *name_ptr* can be scheduled on CPU *i*.

cpuset a pointer that directly or indirectly indicates the set of processors, to be used in conjunction with *name_ptr*, to identify the process to be queried. The pointer may either point to a valid `cpuset`, as created by **`cpuset_alloc(3)`** and filled with some set of CPU IDs by other **`cpuset(3)`** operations, or it may take on one of the following special pointer values:

- `FBS_CPUSET_CURCPU` — the first process found that is named by *name_ptr* and is currently running on the processor from which the call is made.
- `FBS_CPUSET_ALLCPUS` — the first process found that is named by *name_ptr* that is running on any processor.
- `FBS_CPUSET_ONECPU(`*cpu*`)` — the first process that is named by *name_ptr* and is running on the processor with a CPU ID of *cpu*.

If *cpuset* points to a **`cpuset(3)`** that is empty, then the behavior is as described for `FBS_CPUSET_CURCPU`.

fpid an integer value providing the unique frequency-based scheduler process identifier. If this value is -1, *name_ptr* must be supplied.

halt indicates whether the scheduler will be stopped upon detection of a deadline violation for the process. The value will be DL_HALT to indicate that the scheduler is to be stopped, or DL_NOHALT.

kind the kind of deadline set by **`sched_pgm_set_deadline(3)`**. Its value affects the interpretation of

deadline. *kind* may be either DEADLINE_CLEAR indicating that no deadline applies to the specified process, or DEADLINE_WALL_TIME indicating that a CLOCK_MONOTONIC deadline time is assigned to the specified process.

origin the point from which the deadline time is measured. *origin* may either be DL_CYCLE_RELATIVE indicating that the deadline time is measured from the beginning of a cycle in which the task is scheduled, or DL_TASK_RELATIVE, indicating that the deadline time is measured from the time the scheduled process exits **fbswait(3)** and begins execution.

deadline the maximum time that the process is expected to spend executing before returning to **fbswait(3)**.

total_violations
the total number of deadline violations that have occurred for the specified process or task.

Return Value

A return value of 0 indicates that the call has been successful. A return value of -1 indicates that an error has occurred; `errno` is set to indicate the error.
Refer to the **sched_pgm_deadline_query(3)** man page and the **sched_pgm_deadline_query_big(3)** man page for a listing of the types of errors that may occur.

sched_pgm_deadline_test, sched_pgm_deadline_test_big – Test for the Presence of a Deadline Violation

These routines tests for the occurrence of a deadline violation by the scheduled process on the frequency-based scheduler.

These two routines are identical except for their treatment of CPU masks. For details, see "The Big-SMP FBS Interface" on page 6-2.

Synopsis

```
#include <fbsched.h>

int sched_pgm_deadline_test (fbs_id, violation_buf)
int fbs_id ;
struct violation_ds
{
    char *name_ptr;
    int cpu;
    int fpid;
    int violated;
    int total_violations;
    deadline_kind kind;
    deadline_origin origin;
    struct timespec remaining;
} *violation_buf ;

int sched_pgm_deadline_test_big (fbs_id, violation_buf)
int fbs_id ;
struct violation_big_ds
{
    char *name_ptr;
    cpuset_t *cpuset;
    void *vfill[9];
    int fpid;
    int violated;
    int total_violations;
    int ifill[9];
    deadline_kind kind;
    deadline_origin origin;
    struct timespec remaining;
} *violation_buf ;
```

Parameters

fbs_id — a unique positive integer value representing the identifier for a frequency-based scheduler. This value can be obtained using **fbsid(3)** (page 6-18) or **fbsconfigure(3)** (page 6-10).

violation_buf — a pointer to a `violation_ds` or `violation_big_ds` structure that contains the deadline violation state information for the scheduled process. The structure definition is:

name_ptr a pointer to the path name that identifies the process. If the path name is the null string, the *fpid* field must be given.

cpu an integer value indicating the processors on which the specified program can be scheduled to run. Acceptable values and corresponding results are:

0	the program pointed to by *name_ptr* can be scheduled on the processor from which the call is made.
-1	the program pointed to by *name_ptr* can be scheduled on any processor.
bit mask	if (*cpu* & (1<<*i*)) is set (where *i* is an integer ranging from 0 to 15 and representing a CPU), the program pointed to by *name_ptr* can be scheduled on CPU *i*.

cpuset a pointer that directly or indirectly indicates the set of processors, to be used in conjunction with *name_ptr*, to identify the process to be tested. The pointer may either point to a valid `cpuset`, as created by **`cpuset_alloc(3)`** and filled in with some set of CPU IDs by other **`cpuset(3)`** operations, or it may take on one of the following special pointer values:

`FBS_CPUSET_CURCPU`	the first process named by *name_ptr* currently running on the processor from which the call is made.
`FBS_CPUSET_ALLCPUS`	the first process named by *name_ptr* running on any processor.
`FBS_CPUSET_ONECPU(`*`cpu`*`)`	the first process named by *name_ptr* running on the processor with a CPU ID of *cpu*.

If *cpuset* points to a **`cpuset(3)`** that is empty, then the behavior is as described for `FBS_CPUSET_CURCPU`.

fpid an integer value providing the unique frequency-based scheduler process identifier. If this value is -1, *name_ptr* must be supplied.

violated indicates whether the scheduler will be stopped upon detection of a deadline violation for the process. The value will be DEADLINE_VIOLATION to indicate that the scheduler is to be stopped, or NO_VIOLATION otherwise.

total_violations
specifies the number of deadline violations detected for the process since the scheduler was started.

kind the kind of deadline set by **`sched_pgm_set_deadline(3)`**. Its value affects

the interpretation of *remaining*. *kind* may be either DEADLINE_CLEAR indicating that no deadline applies to the specified process, or DEADLINE_WALL_TIME indicating that a CLOCK_MONOTONIC deadline time is assigned to the specified process.

origin — the deadline origin set for the process by **sched_pgm_set_deadline(3)**. Its value affects the interpretation of *remaining*. *origin* may either be DL_TASK_RELATIVE, indicating that the remaining time is measured from the beginning of task execution, or DL_CYCLE_RELATIVE indicating that the remaining time is measured from the beginning of a cycle.

remaining — the remaining time until the expiration of the deadline.

Return Value

A return value of 0 indicates that the call has been successful. A return value of -1 indicates that an error has occurred; `errno` is set to indicate the error. Refer to the **sched_pgm_deadline_test(3)** and **sched_pgm_deadline_test_big(3)** man pages for a listing of the types of errors that may occur.

sched_pgm_set_deadline, sched_pgm_set_deadline_big – Set or Clear the Process Deadline Time

These routines set or clear the deadline time for a currently scheduled process on the frequency-based scheduler.

These two routines are identical except for their treatment of CPU masks. For details, see "The Big-SMP FBS Interface" on page 6-2.

The deadline time for a process indicates the maximum amount of time the process is expected to spend executing before returning to **`fbswait(3)`**. If the deadline time is exceeded, a deadline violation is incurred by the process. The scheduler may optionally be halted upon detection of a deadline violation.

To set or clear a deadline, the calling process must have alter permission for the scheduler.

Identify the process by using one of the following methods:

- Specify the name of the process and the CPU on which it is scheduled.
- Specify the process' frequency-based scheduler process identifier.
- Specify the name of the process, the CPU on which it is scheduled, and its frequency-based scheduler process identifier.

Synopsis

```
#include <fbsched.h>

int sched_pgm_set_deadline (fbs_id, deadline_buf)
int fbs_id ;
struct deadline_ds
{
    char *name_ptr;
    int cpu;
    int fpid;
    int halt;
    deadline_kind kind;
    deadline_origin origin;
    struct timespec deadline;
}*deadline_buf ;

int sched_pgm_set_deadline_big (fbs_id, deadline_buf)
int fbs_id ;
struct deadline_big_ds
{
    char *name_ptr;
    cpuset_t *cpuset;
    void *vfill[9];
    int fpid;
    int halt;
    int ifill[9];
    deadline_kind kind;
    deadline_origin origin;
    struct timespec deadline;
}*deadline_buf ;
```

Parameters

fbs_id a unique positive integer value representing the identifier for a frequency-based scheduler. This value can be obtained using **fbsid(3)** (page 6-18) or **fbsconfigure(3)** (page 6-10).

deadline_buf

a pointer to a `deadline_ds` or `deadline_big_ds` structure that contains the deadline configuration information for the scheduled process. The structure definition is:

name_ptr a pointer to the path name that identifies the process. If the path name is the null string, the *fpid* field must be given.

cpu an integer value indicating the processors on which the specified program can be scheduled to run. Acceptable values and corresponding results are:

0 the program pointed to by *name_ptr* can be scheduled on the processor from which the call is made.

-1 the program pointed to by *name_ptr* can be scheduled on any processor.

bit mask if (*cpu* & (1<<*i*)) is set (where *i* is an integer ranging from 0 to 15 and representing a CPU), the program pointed to by *name_ptr* can be scheduled on CPU *i*.

cpuset a pointer that directly or indirectly indicates the set of processors, to be used in conjunction with *name_ptr*, to identify the process whose deadline attributes are to be changed. The pointer may either point to a valid `cpuset`, created by **cpuset_alloc(3)** and filled in with some set of CPU IDs by other **cpuset(3)** operations, or it may take on one of the following special pointer values:

`FBS_CPUSET_CURCPU` the first process named by *name_ptr* currently running on the processor from which the call is made.

`FBS_CPUSET_ALLCPUS` the first process named by *name_ptr* running on any processor.

`FBS_CPUSET_ONECPU(`*cpu*`)` the first process named by *name_ptr* running on the processor with a CPU ID of *cpu*.

If *cpuset* points to a **cpuset(3)** that is empty, then the behavior is as described for `FBS_CPUSET_CURCPU`.

fpid — an integer value providing the unique frequency-based scheduler process identifier. If this value is -1, *name_ptr* must be supplied.

halt — indicates whether the scheduler will be stopped upon detection of a deadline violation for the process. The value will be DL_HALT to indicate that the scheduler is to be stopped, or DL_NOHALT.

kind — the kind of deadline to be set, affecting the interpretation of *deadline*: DEADLINE_CLEAR indicating that no deadline applies to the specified process, or DEADLINE_WALL_TIME indicating that a CLOCK_MONOTONIC deadline time is assigned to the specified process.

origin — the point from which the deadline time is measured. *origin* may either be DL_CYCLE_RELATIVE indicating that the deadline time is measured from the beginning of a cycle in which the task is scheduled, or DL_TASK_RELATIVE, indicating that the deadline time is measured from the time the scheduled task exits **`fbswait(3)`** and begins execution.

deadline — must be non-negative and represent a value less than INT_MAX microseconds. By default this value is zero seconds, zero nanoseconds; i.e., if the process never sets a deadline time, it is zero.

Return Value

A return value of 0 indicates that the call has been successful. A return value of -1 indicates that an error has occurred; `errno` is set to indicate the error. Refer to the **`sched_pgm_set_deadline(3)`** and **`sched_pgm_set_deadline_big(3)`** man pages for a listing of the types of errors that may occur.

sched_pgm_set_soft_overrun_limit, sched_pgm_set_soft_overrun_limit_big – Set Soft Overrun Limit

These routines set the consecutive soft overrun limit for a currently scheduled process on the frequency-based scheduler. To set the consecutive soft overrun limit, the calling process must have alter permission for the scheduler.

Identify the process by using one of the following methods:

- Specify the name of the process and the CPU on which it is scheduled.
- Specify the process' frequency-based scheduler process identifier.
- Specify the name of the process, the CPU on which it is scheduled, and its frequency-based scheduler process identifier.

These routines are identical except for their treatment of CPU masks. For details, see "The Big-SMP FBS Interface" on page 6-2.

Synopsis

```
#include <fbsched.h>

int sched_pgm_set_soft_overrun_limit (fbs_id, soft_overrun_buf)
int fbs_id;
struct soft_overrun_ds{
    char *name_ptr;
    int cpu;
    int fpid;
    int soft_limit;
} *soft_overrun_buf;

int sched_pgm_set_soft_overrun_limit_big (fbs_id, soft_overrun_buf)
int fbs_id;
struct soft_overrun_big_ds{
    char *name_ptr;
    cpuset_t *cpuset;
    void *vfill[9];
    int fpid;
    int soft_limit;
    int ifill[9];
} *soft_overrun_buf;
```

Parameters

fbs_id a unique positive integer value representing the identifier for a frequency-based scheduler. This value can be obtained using **fbsid(3)** (page 6-18) or **fbsconfigure(3)** (page 6-10).

Enter MY_FBS to reference the frequency-based scheduler on which the calling process is scheduled without knowing the identifier.

soft_overrun_buf
a pointer to a `soft_overrun_ds` or `soft_overrun_big_ds`

structure that contains the soft overrun status for the scheduled process. The structure definition is:

name_ptr a pointer to the path name that identifies the process. If the path name is the null string, the *fpid* field must be given.

cpu an integer value indicating the processors on which the specified program can be scheduled to run. Acceptable values and corresponding results are:

- 0 — the program pointed to by *name_ptr* can be scheduled on the processor from which the call is made.
- -1 — the program pointed to by *name_ptr* can be scheduled on any processor.
- bit mask — if (*cpu* & (1<<*i*)) is set (where *i* is an integer ranging from 0 to 15 and representing a CPU), the program pointed to by *name_ptr* can be scheduled on CPU *i*.

cpuset a pointer that directly or indirectly indicates the set of processors, to be used in conjunction with *name_ptr*, to identify the process whose overrun attributes are to be changed. The pointer may either point to a valid `cpuset`, as created by **`cpuset_alloc(3)`** and filled in with some set of CPU IDs by other **`cpuset(3)`** operations, or it may take on one of the following special pointer values:

- `FBS_CPUSET_CURCPU` — the first process named by *name_ptr* currently running on the procesor from which the call is made.
- `FBS_CPUSET_ALLCPUS` — the first process named by *name_ptr* running on any processor.
- `FBS_CPUSET_ONECPU(`*cpu*`)` — the first process named by *name_ptr* running on the processor with a CPU ID of *cpu*.

If *cpuset* points to a **`cpuset(3)`** that is empty, then the behavior is as described for `FBS_CPUSET_CURCPU`.

fpid an integer value providing the unique frequency-based scheduler process identifier. If this value is -1, *name_ptr* must be supplied.

soft_limit the number of consecutive soft overruns allowed to occur before failure. This value must be non-negative and less than INT_MAX (see **`/usr/include/limits.h`**). By default, this value is 0.

Return Value

A return value of 0 indicates that the call has been successful. A return value of -1 indicates that an error has occurred; `errno` is set to indicate the error.

Refer to the **`sched_pgm_set_soft_overrun_limit(3)`** man page and the **`sched_pgm_set_soft_overrun_limit_big(3)`** man page for a listing of the types of errors that may occur.

sched_pgmadd, sched_pgmadd_big – Schedule a Process on a Frequency-Based Scheduler

These routines create a new process and schedules it on a frequency-based scheduler. To supply arguments for the new process, see **sched_pgmadd_args** and **sched_pgmadd_args_big** on page 6-54. To supply arguments and also soft overrun and/or deadline attributes for the new process, see **sched_pgmadd_attr** and **sched_pgmadd_attr_big** on page 6-58.

These routines are identical except for their treatment of CPU masks. For details, see "The Big-SMP FBS Interface" on page 6-2.

Synopsis

```
#include <fbsched.h>

int sched_pgmadd(fbs_id, sched_buf)
int fbs_id;
struct pgm2_ds {
    char *name_ptr;
    int cpu;
    int fpid;
    int cid;
    int prior;
    int param;
    int period;
    int cycle;
    int halt;
    int status;
} *sched_buf;

int sched_pgmadd_big(fbs_id, sched_buf)
int fbs_id;
struct pgm2_big_ds {
    char *name_ptr;
    cpuset_t *cpuset;
    void *vfill[9];
    int fpid;
    int cid;
    int prior;
    int param;
    int period;
    int cycle;
    int halt;
    int status;
    int ifill[9];
} *sched_buf;
```

Parameters

fbs_id — a unique positive integer value representing the identifier for a frequency-based scheduler. This value can be obtained using **fbsid(3)** (page 6-18) or **fbsconfigure(3)** (page 6-10).

Enter MY_FBS to reference the frequency-based scheduler on which the calling process is scheduled without knowing the identifier.

sched_buf a `pgm2_ds` or `pgm2_big_ds` structure that contains the scheduling parameters with which you wish to schedule the process. The type of information specified in each component is presented below. Note that the *status* component is ignored on this call.

name_ptr a pointer to a variable that contains a path name identifying the program to be scheduled. A full or relative path name of up to 1024 characters can be specified.

cpu an integer value indicating the processors on which the specified program can be scheduled to run. Acceptable values and corresponding results ares:

0 the program pointed to by *name_ptr* can be scheduled on the processor from which the call is made.

-1 the program pointed to by *name_ptr* can be scheduled on any processor.

bit mask if (*cpu* & (1<<*i*)) is set (where *i* is an integer ranging from 0 to 15 and representing a CPU), the program pointed to by *name_ptr* can be scheduled on CPU *i*.

cpuset a pointer that directly or indirectly indicates the set of processors on which the specified program can be scheduled to run. The pointer may either point to a valid `cpuset`, as created by **`cpuset_alloc(3)`** and filled in with some set of CPU IDsby other **`cpuset(3)`** operations, or it may take on one of the following special pointer values:

`FBS_CPUSET_CURCPU` the process can run only on the processor from which this call is made.

`FBS_CPUSET_ALLCPUS` the process can run on any processor.

`FBS_CPUSET_ONECPU(`*`cpu`*`)` the process can run only on the processor whose CPU ID is *cpu*.

If *cpuset* points to a **`cpuset(3)`** that is empty, then the behavior is as described for `FBS_CPUSET_CURCPU`.

fpid a returned integer value that is the unique frequency-based scheduler process identifier for the scheduled process

cid an integer value indicating the POSIX scheduling policy under which the specified program is to be scheduled.

Scheduling policies are defined in the file **<sched.h>**: SCHED_FIFO, SCHED_RR or SCHED_OTHER.

prior — an integer value indicating the scheduling priority of the specified process. The range of acceptable priority values is governed by the scheduling policy specified.

param — an integer value to be passed to a process scheduled on a frequency-based scheduler. This value can be retrieved by the frequency-based scheduled process through a call to **sched_pgmqry** or **sched_pgmqry_big** (see page 6-51 for an explanation of this routine).

period — an integer value indicating the frequency with which the specified program is to be awakened in each major frame.

A period of 1 indicates that the specified program is to be awakened every minor cycle; a period of 2 indicates that it is to be awakened once every two minor cycles, and so on.

This value can range from 1 to the number of minor cycles that compose a frame on the specified scheduler as defined in a call to **fbsconfigure** (see page 6-10).

cycle — an integer value indicating the first minor cycle in which the specified program is scheduled to be awakened in each frame. This value can range from 0 to the total number of minor cycles per frame minus 1. The total number of minor cycles per frame is specified in a call to **fbsconfigure** (see page 6-10).

halt — an integer value indicating whether or not the scheduler should be stopped in the event that the specified program causes a frame overrun. A nonzero value indicates that the scheduler will be stopped.

Return Value

A return value of 0 indicates that the call has been successful. A return value of -1 indicates that an error has occurred; `errno` is set to indicate the error. Refer to the **sched_pgmadd(3)** and **sched_pgmadd_big(3)** man pages for a listing of the types of errors that may occur.

sched_pgmadd_args, sched_pgmadd_args_big– Schedule a Process on a Frequency-Based Scheduler with Arguments

These routines create a new process with arguments and schedules it on a frequency-based scheduler. To supply arguments and also soft overrun and/or deadline attributes for the new process, see **sched_pgmadd_attr** and **sched_pgmadd_attr_big** on page 6-58.

These two routines are identical except for their treatment of CPU masks. For details, see "The Big-SMP FBS Interface" on page 6-2.

Synopsis

```
#include <fbsched.h>

int sched_pgmadd_args(fbs_id, sched_buf, argv[])
int fbs_id;
struct pgm2_ds {
    char *name_ptr;
    int cpu;
    int fpid;
    int cid;
    int prior;
    int param;
    int period;
    int cycle;
    int halt;
    int status;
} *sched_buf;
char * const argv[];

int sched_pgmadd_args_big(fbs_id, sched_buf, argv[])
int fbs_id;
struct pgm2_big_ds {
    char *name_ptr;
    cpuset_t* cpuset;
    void *vfill[9];
    int fpid;
    int cid;
    int prior;
    int param;
    int period;
    int cycle;
    int halt;
    int status;
    int ifill[9];
} *sched_buf;
char * const argv[];
```

Parameters

fbs_id a unique positive integer value representing the identifier for a frequency-based scheduler. This value can be obtained using **`fbsid(3)`** (page 6-18) or **`fbsconfigure(3)`** (page 6-10).

Enter MY_FBS to reference the frequency-based scheduler on which the calling process is scheduled without knowing the identifier.

sched_buf a `pgm2_ds` or `pgm2_big_ds` structure that contains the scheduling parameters with which you wish to schedule the process. The type of information specified in each component is presented below. Note that the *status* component is ignored on this call.

name_ptr a pointer to a variable that contains a path name identifying the program to be scheduled. A full or relative path name of up to 1024 characters can be specified.

cpu an integer value indicating the processors on which the specified program can be scheduled to run. Acceptable values and corresponding results ares:

0 the program pointed to by *name_ptr* can be scheduled on the processor from which the call is made.

-1 the program pointed to by *name_ptr* can be scheduled on any processor.

bit mask if (*cpu* & (1<<*i*)) is set (where *i* is an integer ranging from 0 to 15 and representing a CPU), the program pointed to by *name_ptr* can be scheduled on CPU *i*.

cpuset a pointer that directly or indirectly indicates the set of processors on which the specified program can be scheduled to run. The pointer may either point to a valid `cpuset`, as created by **`cpuset_alloc(3)`** and filled in with some set of CPU IDsby other **`cpuset(3)`** operations, or it may take on one of the following special pointer values:

`FBS_CPUSET_CURCPU` the process can run only on the processor from which this call is made.

`FBS_CPUSET_ALLCPUS` the process can run on any processor.

`FBS_CPUSET_ONECPU(`*`cpu`*`)` the process can run only on the processor whose CPU ID is *cpu*.

If *cpuset* points to a **`cpuset(3)`** that is empty, then the behavior is as described for `FBS_CPUSET_CURCPU`.

fpid — a returned integer value that is the unique frequency-based scheduler process identifier for the scheduled process

cid — an integer value indicating the POSIX scheduling policy under which the specified program is to be scheduled. Scheduling policies are defined in the file **<sched.h>**: SCHED_FIFO, SCHED_RR or SCHED_OTHER.

prior — an integer value indicating the scheduling priority of the specified process. The range of acceptable priority values is governed by the scheduling policy specified.

param — an integer value to be passed to a process scheduled on a frequency-based scheduler. This value can be retrieved by the frequency-based scheduled process through a call to **sched_pgmqry** or **sched_pgmqry_big** (see page 6-51 for an explanation of this routine).

period — an integer value indicating the frequency with which the specified program is to be awakened in each major frame.

A period of 1 indicates that the specified program is to be awakened every minor cycle; a period of 2 indicates that it is to be awakened once every two minor cycles, and so on.

This value can range from 1 to the number of minor cycles that compose a frame on the specified scheduler as defined in a call to **fbsconfigure** (see page 6-10).

cycle — an integer value indicating the first minor cycle in which the specified program is scheduled to be awakened in each frame. This value can range from 0 to the total number of minor cycles per frame minus 1. The total number of minor cycles per frame is specified in a call to **fbsconfigure** (see page 6-10).

halt — an integer value indicating whether or not the scheduler should be stopped in the event that the specified program causes a frame overrun. A nonzero value indicates that the scheduler will be stopped.

argv[] — an array of pointers to null-terminated strings that represent the argument list available to the scheduled program. By convention, the first argument, in argv[0], should point to the file name associated with the program being scheduled. The array of pointers must be terminated by a NULL pointer.

Return Value

A return value of 0 indicates that the call has been successful. A return value of -1 indicates that an error has occurred; `errno` is set to indicate the error. Refer to the

sched_pgmadd_args(3) and **sched_pgmadd_args_big(3)** man pages for a listing of the types of errors that may occur.

sched_pgmadd_attr, sched_pgmadd_attr_big – Schedule a Process on a Frequency-Based Scheduler with Arguments and Attributes

These routines create a new process with arguments and soft overrun attributes and schedules it on a frequency-based scheduler. The specified soft overrun attributes for the new process are guaranteed to be initialized before the new process begins execution.

These two routines are identical except for their treatment of CPU masks. For details, see "The Big-SMP FBS Interface" on page 6-2.

Synopsis

```
#include <fbsched.h>

int sched_pgmadd_attr (fbsid, sched_attr)
int fbs_id;
struct pgmadd_attr_ds *sched_attr;

struct pgmadd_attr_v1 {
   struct pgm2_ds *pgm2;
   char * const *argv;
   int so_soft_limit;
   int dl_halt;
   deadline_kind dl_kind;
   deadline_origin dl_origin;
   struct timespec dl_deadline;
};
struct pgmadd_attr_ds {
   int version;
   union {
     struct pgmadd_attr_v1 v1;
   } attr;
};

int sched_pgmadd_attr_big (fbsid, sched_attr)
int fbs_id;
struct pgmadd_attr_big_ds *sched_attr;

struct pgmadd_attr_big_v1 {
   struct pgm2_big_ds *pgm2;
   char * const *argv;
   void *vfill[9];
   int so_soft_limit;
   int dl_halt;
   int ifill[9];
   deadline_kind dl_kind;
   deadline_origin dl_origin;
   struct timespec dl_deadline;
};
struct pgmadd_attr_big_ds {
   int version;
   union {
     struct pgmadd_attr_big_v1 v1;
```

```
    } attr;
};
```

Parameters

fbs_id a unique positive integer value representing the identifier for a frequency-based scheduler. This value can be obtained using **`fbsid(3)`** (page 6-18) or **`fbsconfigure(3)`** (page 6-10).

Enter MY_FBS to reference the frequency-based scheduler on which the calling process is scheduled without knowing the identifier.

sched_attr a pointer to a `pgmadd_attr_ds` or `pgmadd_attr_big_ds` structure that contains the scheduling parameters and attributes for the process. The type of information specified in each field of this structure is described below.

version the version of the `attr` structure being used. Currently only version 1 is supported, so this field must be set to PGMADD_ATTR_VERSION1 to indicate that the `pgmadd_attr_v1` structure, `v1`, is being used.

pgm2 a pointer to the `pgm2_ds` or `pgm2_big_ds` structure that contains the scheduling parameters used to schedule the process. Refer to the *sched_buf* parameters in the **`sched_pgmadd_args`** section on page 6-54 for more details on the `pgm2_ds` structure fields.

argv a pointer which may be set to a value of NULL if arguments are not being used, or should otherwise point to an array of pointers of null-terminated strings that represent the argument list available to the scheduled program. By convention, the first argument, in `argv[0]`, should point to the file name associated with the program being scheduled. The array of pointers must be terminated by a NULL pointer.

so_soft_limit may be set to zero if not used, or may contain a soft overrun process limit value. Refer to **`sched_pgm_set_soft_overrun_limit`** on page 6-48 for more details on soft overruns.

dl_halt must be set to DL_HALT to specify that the scheduler is to be stopped when a deadline violation is detected for the process, or DL_NOHALT if the scheduler should not be halted.

dl_kind may be set to DEADLINE_CLEAR to indicate that no deadline is to be applied to the scheduled process, or DEADLINE_WALL_TIME indicating that a CLOCK_MONOTONIC deadline time is to be applied. When set to DEADLINE_WALL_TIME, *dl_halt*, *dl_origin* and *dl_deadline* must also be set.

dl_origin May be set to DL_CYCLE_RELATIVE to measure the deadline time from the beginning of the cycle in which the process is scheduled to run, or DL_TASK_RELATIVE to

measure deadline times from the time the process exits **fbswait(3)** and begins execution.

dl_deadline when *dl_kind* is set to DEADLINE_WALL_TIME, this timespec structure indicates the amount of CLOCK_MONOTONIC time the process is expected to execute before returning to **fbswait(3)**. See **sched_pgm_set_deadline** on page 6-45 for more details.

Return Value

A return value of 0 indicates that the call has been successful. A return value of -1 indicates that an error has occurred; errno is set to indicate the error. Refer to the **sched_pgmadd_attr(3)** and **sched_pgmadd_attr_big(3)** man page for a listing of the types of errors that may occur.

sched_pgmqry, sched_pgmqry_big – Query a Process

These routines obtain information for a particular process that has been scheduled on a frequency-based scheduler. You can identify the process by using one of the following methods:

- Specify the name of the process and the CPU(s) on which it is scheduled.
- Specify the process' frequency-based scheduler process identifier.
- Specify the name of the process, the CPU on which it is scheduled, and its frequency-based scheduler process identifier.

NOTE

The only method that can be used to identify a process that has been scheduled multiple times on the same CPU is to specify its frequency-based scheduler process identifier.

Information returned includes the following:

- the process' path name
- the CPU on which the process can execute
- the frequency-based scheduler process identifier
- the scheduling policy under which the process has been scheduled
- the scheduling priority
- the period (the number of minor cycles indicating the frequency with which the process is awakened in each major frame)
- the starting base cycle (the first minor cycle in which the process is scheduled to be awakened in each major frame)
- the value of the "halt on overrun" flag

These two routines are identical except for their treatment of CPU masks. For details, see "The Big-SMP FBS Interface" on page 6-2.

Synopsis

```
#include <fbsched.h>

int sched_pgmqry(fbs_id, qry_buf)
int fbs_id;
struct pgm2_ds {
    char *name_ptr;
    int cpu;
    int fpid;
    int cid;
    int prior;
    int param;
    int period;
    int cycle;
    int halt;
    int status;
} *qry_buf;
```

```
int sched_pgmqry_big(fbs_id, qry_buf)
int fbs_id;
struct pgm2_big_ds {
    char *name_ptr;
    cpuset_t cpuset;
    void* vfill[9];
    int fpid;
    int cid;
    int prior;
    int param;
    int period;
    int cycle;
    int halt;
    int status;
    int ifill[9];
} *qry_buf;
```

Parameters

fbs_id a unique positive integer value representing the identifier for a frequency-based scheduler. This value can be obtained using **fbsid(3)** (page 6-18) or **fbsconfigure(3)** (page 6-10).

Enter MY_FBS to reference the frequency-based scheduler on which the calling process is scheduled without knowing the identifier.

qry_buf a pointer to a pgm2_ds or pgm2_big_ds structure that contains scheduling information for the process. **sched_pgmqry** will return to this structure the scheduling information for a specified process. The information contained in each component of the structure to which *qry_buf* points is presented below.

name_ptr a pointer to the path name identifying the process for which information is to be returned. A full or relative path name of up to 1024 characters can be specified. If the pointer points to a null string, you must provide the frequency-based scheduler process identifier in the *fpid* component.

cpu an integer value indicating the processor(s) to be used with the value of *name_ptr* to identify the program for which information is to be obtained. Acceptable values and corresponding results follow:

0 the first process whose name matches the name pointed to by *name_ptr* currently running on the processor from which the call is made is requested.

-1 the first process whose name matches the name pointed to by *name_ptr* currently running on any processor is requested.

bit mask if (*cpu* & (1<<*i*)) is set (where *i* is an integer

ranging from 0 to 15 and representing a CPU) and it is the only bit set, the first process whose name matches the name pointed to by *name_ptr* running on CPU *i* is requested.

If (*cpu* & (1<<*i*)) is set and it is not the only bit set, the first process whose name matches the name pointed to by *name_ptr* currently running on any of the selected CPUs is requested.

cpuset a pointer that directly or indirectly indicates the set of processors, to be used in conjunction with *name_ptr*, to identify the process to be queried. The pointer may either point to a valid `cpuset`, as created by **`cpuset_alloc(3)`** and filled in with some set of CPU IDs by other **`cpuset(3)`** operations, or it may take on one of the following special pointer values:

`FBS_CPUSET_CURCPU` the first process named by *name_ptr* currently running on the processor from which call is made.

`FBS_CPUSET_ALLCPUS` the first process named by *name_ptr* running on any processor.

`FBS_CPUSET_ONECPU(`*cpu*`)` the first process named by *name_ptr* running on the processor with a CPU ID of *cpu*.

If *cpuset* points to a **`cpuset(3)`** that is empty, then the behavior is as described for `FBS_CPUSET_CURCPU`.

fpid — an integer value providing the frequency-based scheduler process identifier for the process for which information is to be returned. This value is obtained when you make a call to **sched_pgmadd** (see page 6-51). This value must be -1 if you wish to identify the program to be queried only by specifying *name_ptr* and *cpu*.

cid — an integer value indicating the specified process' scheduling policy.

prior — an integer value indicating the process' scheduling priority.

param — an integer value indicating the value passed to the process via a call to **sched_pgmadd** or **sched_pgmresched**.

period — an integer value indicating the frequency with which the specified program is to be awakened in each major frame.

cycle — an integer value indicating the first minor cycle in which the specified process is scheduled to be awakened in each frame.

halt — an integer value indicating the value of the "halt on overrun" flag. A nonzero value indicates that the flag is set. A value of zero indicates that the flag is not set.

status — an integer value indicating the current state of the specified process as defined in **<fbsched.h>**

Return Value

A return value of 0 indicates that the call has been successful. A return value of -1 indicates that an error has occurred; `errno` is set to indicate the error. Refer to the **sched_pgmqry(3)** and **sched_pgmqry_big(3)** man pages for a listing of the types of errors that may occur.

sched_pgmresched, sched_pgmresched_big– Reschedule a Process

These routines change the scheduling parameters for a process that is scheduled on a frequency-based scheduler. You may wish, for example, to change a program's policy or priority or the frequency with which it is scheduled to run. You cannot, however, change the CPU on which it has been scheduled.

It is important to note that to use this routine to (1) change a process' scheduling policy to the SCHED_FIFO or the SCHED_RR policy or (2) change the priority of a process scheduled under SCHED_FIFO or SCHED_RR, or (3) raise the priority of a process scheduled under SCHED_OTHER above a per-process limit, the effective user ID of the calling process must match the effective user ID of the target process (the process for which the scheduling policy and priority are being set).

Identify the process to reschedule using one of the following methods:

- Specify the name of the process and the CPU on which it is scheduled.
- Specify the process' frequency-based scheduler process identifier.
- Specify the name of the process, the CPU on which it is scheduled, and its frequency-based scheduler process identifier.

NOTE

The only method that can be used to identify a process that has been scheduled multiple times on the same CPU is to specify its frequency-based scheduler process identifier.

These two routines are identical except for their treatment of CPU masks. For details, see "The Big-SMP FBS Interface" on page 6-2.

Synopsis

```
#include <fbsched.h>

int sched_pgmresched(fbs_id, rsch_buf)
int fbs_id;
struct pgm2_ds {
    char *name_ptr;
    int cpu;
    int fpid;
    int cid;
    int prior;
    int param;
    int period;
    int cycle;
    int halt;
    int status;
} *rsch_buf;

int sched_pgmresched_big(fbs_id, rsch_buf)
int fbs_id;
struct pgm2_big_ds {
    char *name_ptr;
    cpuset_t *cpuset;
```

```
    void * vfill[9];
    int fpid;
    int cid;
    int prior;
    int param;
    int period;
    int cycle;
    int halt;
    int status;
    int ifill[9];
} *rsch_buf;
```

Parameters

fbs_id a unique positive integer value representing the identifier for a frequency-based scheduler. This value can be obtained using **fbsid(3)** (page 6-18) or **fbsconfigure(3)** (page 6-10).

Enter MY_FBS to reference the frequency-based scheduler on which the calling process is scheduled without knowing the identifier.

rsch_buf A pgm2_ds or pgm2_big_ds structure that contains the scheduling parameters with which to reschedule the process. The type of information specified in each component is presented below. Note that the *status* component is ignored on this call.

name_ptr a pointer to a variable that contains a path name identifying the process for which information is to be rescheduled. A full or relative path name of up to 1024 characters can be specified. If the pointer points to a null string, you must provide the frequency-based scheduler process identifier in the *fpid* component.

cpu an integer value indicating the processor(s) to be used with the value of *name_ptr* to identify the process to be rescheduled. Acceptable values and corresponding results follow:

0 the first process whose name matches the name pointed to by *name_ptr* currently running on the processor from which the call is made is rescheduled.

-1 the first process whose name matches the name pointed to by *name_ptr* currently running on any processor is rescheduled.

bit mask if (*cpu* & (1<<*i*)) is set (where *i* is an integer ranging from 0 to 15 and representing a CPU) and it is the only bit set, the first process whose name matches the name pointed to by *name_ptr* running on CPU *i* is rescheduled.

If (*cpu* & (1<<*i*)) is set and it is not the only bit

set, the first process whose name matches the name pointed to by *name_ptr* currently running on any of the selected CPUs is rescheduled.

cpuset — a pointer that directly or indirectly indicates the set of processors, to be used in connection with *name_ptr*, to identify the process to be rescheduled. The pointer may either point to a valid `cpuset`, as created by **`cpuset_alloc(3)`** and filled in with some set of CPU IDs by other **`cpuset(3)`** operations, or it may take on one of the following special pointer values:

`FBS_CPUSET_CURCPU` — the first process named by *name_ptr* currently running on the processor from which call is made.

`FBS_CPUSET_ALLCPUS` — the first process named by *name_ptr* running on any processor.

`FBS_CPUSET_ONECPU(`*cpu*`)` — the first process named by *name_ptr* running on the processor with a CPU ID of *cpu*.

If *cpuset* points to a **`cpuset(3)`** that is empty, then the behavior is as described for `FBS_CPUSET_CURCPU`.

fpid — an integer value providing the frequency-based scheduler process identifier for the process to be rescheduled. This value is obtained when you make a call to **`sched_pgmadd`** (see page 6-51). This value must be -1 if you wish to identify the program to be queried only by specifying *name_ptr* and *cpu*.

cid — an integer value indicating the specified process' scheduling policy: SCHED_FIFO, SCHED_RR or SCHED_OTHER.

prior — an integer value indicating the process' scheduling priority. The range of acceptable priority values is governed by the scheduling policy specified.

param — an integer value indicating the value passed to the process scheduled on a frequency-based scheduler.

period — an integer value indicating the frequency with which the specified program is to be awakened in each major frame.

A period of 1 indicates that the specified program is to be awakened every minor cycle; a period of 2 indicates that it is to be awakened once every two minor cycles, and so on.

This value can range from 1 to the number of minor cycles that compose a frame on the specified scheduler as defined in a call to **`fbsconfigure`** (see page 6-10)

cycle — an integer value indicating the first minor cycle in which the specified process is scheduled to be awakened in each frame.

This value can range from 0 to the total number of minor cycles per frame minus 1. The total number of minor cycles per frame is specified in a call to **fbsconfigure** (see page 6-10).

halt — an integer value indicating whether or not the scheduler should be stopped in the event that the specified process causes a frame overrun. A nonzero value indicates that the scheduler will be stopped.

Return Value

A return value of 0 indicates that the call has been successful. A return value of -1 indicates that an error has occurred; `errno` is set to indicate the error. Refer to the **sched_pgmresched(3)** and **sched_pgmresched_big(3)** man pages for a listing of the types of errors that may occur.

Performance Monitor Routines

The Performance Monitor routines provide access to the key features of the Performance Monitor. They enable you to perform such basic operations as:

- clearing performance monitor values for a process or processor
- starting and stopping performance monitoring for a process or processor
- obtaining performance monitor values for a process or processor

Routine Summary

Performance Monitor routines are summarized in Table 6-3. Complete information about each routine is provided under the section “Using Performance Monitor Routines.”

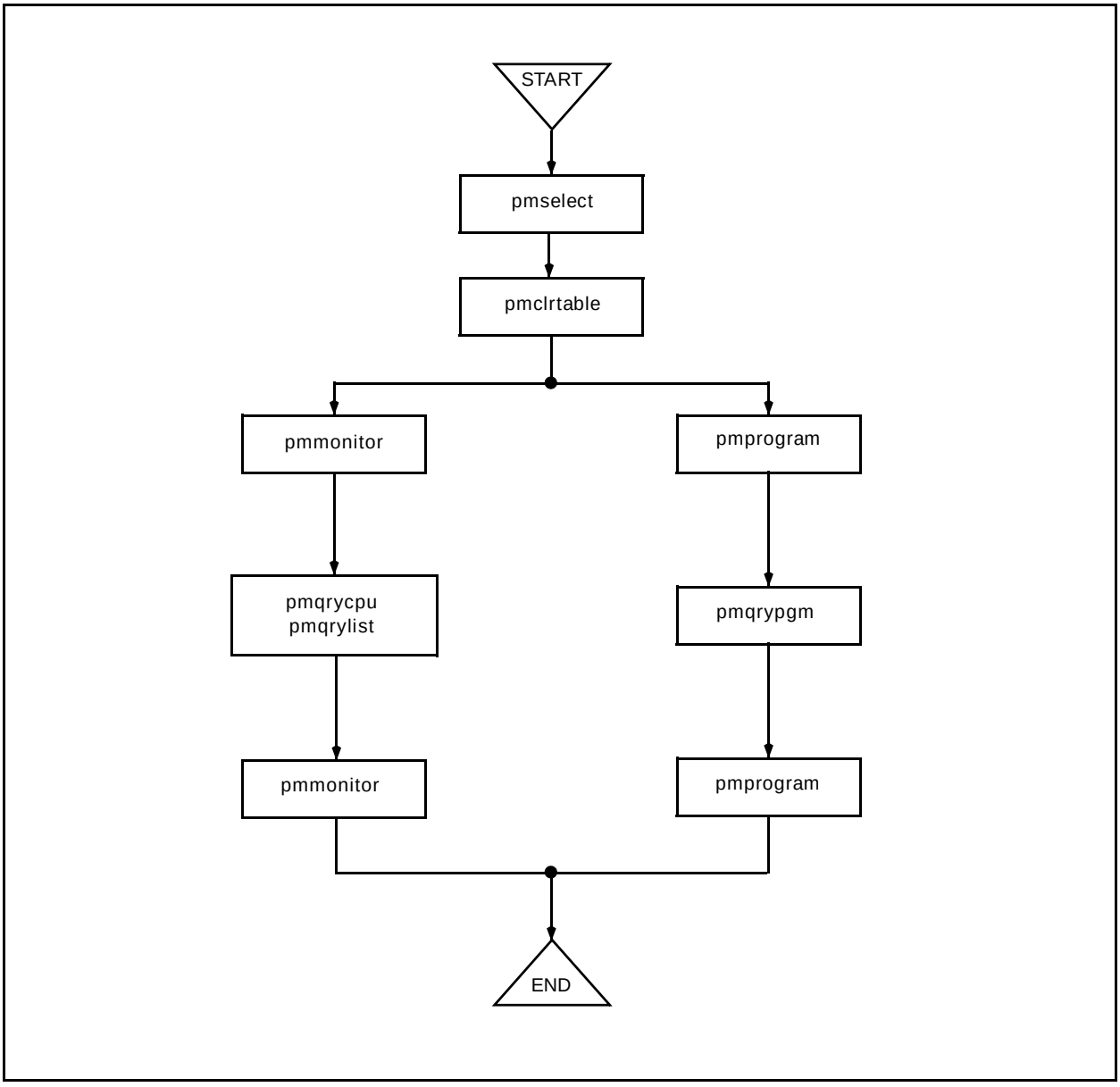
Table 6-3. Performance Monitor C Library Routines

Routine	Page	Description
pmclrpgm pmclrpgm_big	6-71	Clear values for a process
pmclrtable pmclrtable_big	6-73	Clear values for processor(s)
pmmonitor pmmonitor_big	6-75	Start/stop performance monitoring on processor(s)
pmprogram pmprogram_big	6-77	Start/stop performance monitoring on a process
pmqrycpu pmqrycpu_big	6-79	Query values for selected processor(s)
pmqrylist	6-83	Query values for a list of processes
pmqrypgm pmqrypgm_big	6-85	Query values for a selected process
pmqrytimer	6-88	Query Performance Monitor mode
pmselect	6-89	Select Performance Monitor mode

C Library Call Sequence

The approximate order in which you might call the Performance Monitor routines from an application program is illustrated in Figure 6-2..

Figure 6-2. C Library Call Sequence: Performance Monitor



Using Performance Monitor Routines

In the sections that follow, the Performance Monitor routines contained in the **`libccur_fbsched`** library are presented in alphabetical order.

pmclrpgm, pmclrpgm_big, – Clear Values for a Process

These routines clear performance monitor values for a particular process that has been scheduled on a frequency-based scheduler. You can identify the process using one of the following methods:

- Specify the name of the process and the CPU(s) on which it is scheduled.
- Specify the process' frequency-based scheduler process identifier.
- Specify the name of the process, the CPU on which it is scheduled, and its frequency-based scheduler process identifier.

NOTE

This routine clears the process' total soft overrun count.

These routines are identical except for their treatment of CPU masks. For details, see "The Big-SMP FBS Interface" on page 6-2.

Synopsis

```
#include <fbsched.h>

int  pmclrpgm(fbs_id, name, cpu, fpid)
int  fbs_id, cpu, fpid;
char *name;

int  pmclrpgm_big(fbs_id, name, cpuset, fpid)
int  fbs_id, fpid;
char *name;
cpuset_t *cpuset;
```

Parameters

fbs_id — a unique positive integer value representing the identifier for a frequency-based scheduler. This value can be obtained using **`fbsid(3)`** (page 6-18) or **`fbsconfigure(3)`** (page 6-10).

Enter MY_FBS to reference the frequency-based scheduler on which the calling process is scheduled without knowing the identifier.

name — a path name identifying the process for which values are to be cleared. A full or relative path name of up to 1024 characters can be specified. If this variable is the null string, you must provide the frequency-based scheduler process identifier in the *fpid* parameter.

cpu — an integer value indicating the processor(s) to be used in conjunction with the value of the *name* parameter to identify the process for which values are to be cleared. Acceptable values and corresponding results are presented below.

0	the first process named by *name* currently running on the processor from which the call is made is specified
-1	the first process named by *name* currently running on any processor is specified

bit mask — if (*cpu* & (1<<*i*)) is set (where *i* is an integer ranging from 0 to 15 and representing a CPU) and it is the only bit set, the first process named by *name* currently running on CPU *i* is specified

If (*cpu* & (1<<*i*)) is set and it is not the only bit set, the first process named by *name* currently running on any of the selected CPUs is specified.

cpuset — a pointer that directly or indirectly indicates the set of processors, to be used in conjunction with *name*, to identify the process for which values are to be cleared. The pointer may either point to a valid `cpuset`, as created by **`cpuset_alloc(3)`** and filled in with some set of CPU IDs by other **`cpuset(3)`** operations, or it may take on one of the following special pointer values:

`FBS_CPUSET_CURCPU` — the first process named by *name* currently running on the processor from which the call is made.

`FBS_CPUSET_ALLCPUS` — the first process named by *name* running on any processor.

`FBS_CPUSET_ONECPU(`*cpu*`)` — the first process named by *name* running on the processor with a CPU ID of *cpu*.

If *cpuset* points to a **`cpuset(3)`** that is empty, then the behavior is as described for `FBS_CPUSET_CURCPU`.

fpid — an integer value providing the unique frequency-based scheduler process identifier for the process for which values are to be cleared. This value is obtained when you make a call to **`sched_pgmadd`** (see page 6-51). This value must be -1 to identify the process only by specifying *name* and *cpu*.

Return Value

A return value of 0 indicates that the call has been successful. A return value of -1 indicates that an error has occurred; `errno` is set to indicate the error. Refer to the **`pmclrpgm(3)`** and **`pmclrpgm_big(3)`** man pages for a listing of the types of errors that may occur.

pmclrtable, pmclrtable_big – Clear Values for Processor(s)

These routines clear performance monitor values for frequency-based scheduled processes on one or more specified processors on a selected scheduler.

NOTE

These routines clear the total soft overrun count for all related processes.

These two routines are identical except for their treatment of CPU masks. For details, see "The Big-SMP FBS Interface" on page 6-2.

Synopsis

```
#include <fbsched.h>

int  pmclrtable(fbs_id, cpu)
int  fbs_id, cpu;

int  pmclrtable_big(fbs_id, cpuset)
int  fbs_id;
cpuset_t *cpuset;
```

Parameters

fbs_id — a unique positive integer value representing the identifier for a frequency-based scheduler. This value can be obtained using **`fbsid(3)`** (page 6-18) or **`fbsconfigure(3)`** (page 6-10).

Enter MY_FBS to reference the frequency-based scheduler on which the calling process is scheduled without knowing the identifier.

cpu — an integer value indicating the processor or processors for which performance monitor values are to be cleared. Acceptable values and corresponding results are presented below.

0	performance monitor values for frequency-based scheduled processes executing on the processor from which the call is made are cleared
-1	performance monitor values for all processes on the scheduler are cleared
bit mask	if (*cpu* & (1<<*i*)) is set (where *i* is an integer ranging from 0 to 15 and representing a CPU), performance monitor values for processes executing on CPU *i* are cleared

cpuset — a pointer that directly or indirectly indicates the set of processors for which performance monitor values are to be cleared. The pointer may either point to a valid `cpuset`, as created by **`cpuset_alloc(3)`** and filled in with some set of CPU IDs by other **`cpuset(3)`** operations, or it may take on one of the following special pointer values:

FBS_CPUSET_CURCPU	the processor from which the call is made.
FBS_CPUSET_ALLCPUS	all processors.
FBS_CPUSET_ONECPU(*cpu*)	the processor with a CPU ID of *cpu*.

If *cpuset* points to a **cpuset(3)** that is empty, then the behavior is as described for FBS_CPUSET_CURCPU.

Return Value

A return value of 0 indicates that the call has been successful. A return value of -1 indicates that an error has occurred; errno is set to indicate the error. Refer to the **pmclrtable(3)** and **pmclrtable_big(3)** man pages for a listing of the types of errors that may occur.

pmmonitor, pmmonitor_big – Start/Stop Performance Monitoring on Processor(s)

These routines start or stop performance monitoring for frequency-based scheduled processes on one or more processors on a selected scheduler.

These two routines are identical except in their treatment of CPU masks. For a complete description of these differences, see "The Big-SMP FBS Interface" on page 6-2.

Synopsis

```
#include <fbsched.h>

int  pmmonitor(fbs_id, pmflag, cpu)
int  fbs_id, pmflag,cpu;

int  pmmonitor_big(fbs_id, pmflag, cpuset)
int  fbs_id, pmflag;
cpuset_t* cpuset;
```

Parameters

fbs_id — a unique positive integer value representing the identifier for a frequency-based scheduler. This value can be obtained using **fbsid(3)** (page 6-18) or **fbsconfigure(3)** (page 6-10).

Enter MY_FBS to reference the frequency-based scheduler on which the calling process is scheduled without knowing the identifier.

pmflag — an integer value indicating whether performance monitoring is to be started or stopped. A nonzero value indicates that performance monitoring is to be started. A zero value indicates that performance monitoring is to be stopped.

cpu — an integer that indicates the processor or processors for which performance monitoring is to be started or stopped. Acceptable values and corresponding results are presented below.

0	performance monitoring for frequency-based scheduled processes executing on the processor from which the call is made is started or stopped.
-1	performance monitoring for all processes on the scheduler is started or stopped.
bit mask	if (*cpu* & (1<<*i*)) is set (where *i* is an integer ranging from 0 to 15 and representing a CPU), performance monitoring for processes executing on CPU *i* is started or stopped.

cpuset — a pointer that directly or indirectly indicates the set of processors for which performance monitoring is to be started or stopped. The pointer may either point to a valid cpuset, as created by **cpuset_alloc(3)** and filled in with some set of CPU IDs by other **cpuset(3)** operations, or it may take on one of the following special pointer values:

FBS_CPUSET_CURCPU	the processor from which the call is made.
FBS_CPUSET_ALLCPUS	all processors.
FBS_CPUSET_ONECPU(*cpu*)	the processor with a CPU ID of *cpu*.

If *cpuset* points to a **cpuset(3)** that is empty, then the behavior is as described for FBS_CPUSET_CURCPU.

Return Value

A return value of 0 indicates that the call has been successful. A return value of -1 indicates that an error has occurred; errno is set to indicate the error. Refer to the **pmmonitor(3)** and **pmmonitor_big(3)** man pages for a listing of the types of errors that may occur.

pmprogram, pmprogram_big – Start/Stop Performance Monitoring on a Process

These routines start or stop performance monitoring for a particular process that has been scheduled on a frequency-based scheduler. Identify the process by using one of the following methods:

- Specify the name of the process and the CPU(s) on which it is scheduled.
- Specify the process' frequency-based scheduler process identifier.
- Specify the name of the process, the CPU(s) on which it is scheduled, and its frequency-based scheduler process identifier.

These two routines are identical except for their treatment of CPU masks. For details, see "The Big-SMP FBS Interface" on page 6-2.

Synopsis

```
#include <fbsched.h>

int  pmprogram(fbs_id, name, cpu, fpid, pmflag)
int  fbs_id, cpu, fpid, pmflag;
char *name;

int  pmprogram_big(fbs_id, name, cpuset, fpid, pmflag)
int  fbs_id, fpid, pmflag;
char *name;
cpuset_t *cpuset;
```

Parameters

fbs_id — a unique positive integer value representing the identifier for a frequency-based scheduler. This value can be obtained using **`fbsid(3)`** (page 6-18) or **`fbsconfigure(3)`** (page 6-10).

Enter MY_FBS to reference the frequency-based scheduler on which the calling process is scheduled without knowing the identifier.

name — a path name identifying the process for which performance monitoring is to be started or stopped. A full or relative path name of up to 1024 characters can be specified. If this variable is the null string, you must provide the frequency-based scheduler process identifier in the *fpid* parameter.

cpu — an integer value indicating the processor(s) to be used in conjunction with the value of the *name* parameter to identify the process for which performance monitoring is to be started or stopped. Valid values are presented below:

0	the first process named by *name* currently running on the processor from which the call is made is specified
-1	the first process named by *name* currently running on any processor is specified
bit mask	if (*cpu* & (1<<*i*)) is set (where *i* is an integer ranging from 0 to 15 and representing a CPU) and it is the only bit set, the first process named by *name* currently

running on CPU *i* is specified.

If (*cpu* & (1<<*i*)) is set and it is not the only bit set, the first process named by *name* currently running on any of the selected CPUs is specified.

cpuset a pointer that directly or indirectly indicates the set of processors, to be used in conjunction with *name*, to identify the process for which performance monitoring is to be started or stopped. The pointer may either point to a valid `cpuset`, as created by **`cpuset_alloc(3)`** and filled in with some set of CPU IDs by other **`cpuset(3)`** operations, or it may take on one of the following special pointer values:

`FBS_CPUSET_CURCPU` the first process named by *name* currently running on the processor from which the call is made.

`FBS_CPUSET_ALLCPUS` the first process named by *name* running on any processor.

`FBS_CPUSET_ONECPU(`*`cpu`*`)` the first process named by *name* running on the processor with a CPU ID of *cpu*.

If *cpuset* points to a **`cpuset(3)`** that is empty, the the behavior is as described for `FBS_CPUSET_CURCPU`.

fpid an integer value providing the unique frequency-based scheduler process identifier for the process for which performance monitoring is to be started or stopped.

Get this value by making a call to **`sched_pgmadd`** (see page 6-51). This value must be -1 if you wish to identify the process only by specifying *name* and *cpu*.

pmflag an integer value indicating whether performance monitoring is to be started or stopped. A nonzero value indicates that performance monitoring is to be started. A zero value indicates that performance monitoring is to be stopped.

Return Value

A return value of 0 indicates that the call has been successful. A return value of -1 indicates that an error has occurred; `errno` is set to indicate the error. Refer to the **`pmprogram(3)`** and **`pmprogram_big(3)`** man pages for a listing of the types of errors that may occur.

pmqrycpu, pmqrycpu_big – Query Values for Selected Processor(s)

These routines obtain performance monitor values for frequency-based scheduled processes on one or more specified processors on a selected scheduler.

These two routines are identical except for their treatment of CPU masks. For details, see "The Big-SMP FBS Interface" on page 6-2.

Synopsis

```
#include <fbsched.h>

int  pmqrycpu(fbs_id, cpu, pm_buf, buf_cnt)
int  fbs_id, cpu, buf_cnt;
struct pmqry_ds {
  int                fpid;
  struct timespec    lastcyc_tm;
  int                tot_cycles;
  struct timespec    tot_cycles_tm;
  int                overruns;
  struct timespec    mincyc_tm;
  int                mincyc_cycle;
  int                mincyc_frame;
  struct timespec    maxcyc_tm;
  int                maxcyc_cycle;
  int                maxcyc_frame;
  struct timespec    minframe_tm;
  int                minframe;
  struct timespec    maxframe_tm;
  int                maxframe;
} *pm_buf;

int  pmqrycpu_big(fbs_id, cpuset, pm_buf, buf_cnt)
int  fbs_id, buf_cnt;
cpuset_t *cpuset;
struct pmqry_ds *pm_buf;
```

Parameters

fbs_id — a unique positive integer value representing the identifier for a frequency-based scheduler. This value can be obtained using **fbsid(3)** (page 6-18) or **fbsconfigure(3)** (page 6-10).

Enter MY_FBS to reference the frequency-based scheduler on which the calling process is scheduled without knowing the identifier.

cpu — an integer value indicating the processor(s) for which performance monitor values are to be obtained. Acceptable values and corresponding results are presented below.

- 0 — performance monitor values for frequency-based scheduled processes executing on the processor from which the call is made are returned.
- -1 — performance monitor values for all processes on the scheduler are returned.

bit mask — if (*cpu* & (1<<*i*)) is set (where *i* is an integer ranging from 0 to 15 and representing a CPU), performance monitor values for processes executing on CPU *i* are returned.

cpuset — a pointer that directly or indirectly indicates the set of processors for which performance monitor values are to be obtained. The pointer may either point to a valid `cpuset`, as created by **`cpuset_alloc(3)`** and filled in with some set of CPU IDs by other **`cpuset(3)`** operations, or it may take on one of the following special pointer values:

`FBS_CPUSET_CURCPU` — the processor from which the call is made.

`FBS_CPUSET_ALLCPUS` — all processors.

`FBS_CPUSET_ONECPU(`*`cpu`*`)` — the processor with a CPU ID of *cpu*.

If *cpuset* points to a **`cpuset(3)`** that is empty, then the behavior is as described for `FBS_CPUSET_CURCPU`.

pm_buf — a pointer to an array of `pmqry_ds` structures to which **`pmqrycpu`** returns the performance monitor values for each frequency-based scheduled process on the processor(s) specified with the *cpu* parameter.

The number of processes for which these values are returned is bound by the value of the *buf_cnt* parameter. The type of information returned in each component of the structure for a single process is presented below.

fpid — the process' frequency-based scheduler process identifier.

lastcyc_tm — the amount of time that the process has spent running from the last time that it was awakened by the scheduler until it called **`fbswait`**.

tot_cycles — the number of times that the process has been awakened by the scheduler (total iterations, or cycles).

tot_cycles_tm — the time that the process has spent running in all cycles.

overruns — the number of hard frame overruns caused by the process.

mincyc_tm — the least amount of time that the process has spent running in a cycle (minimum cycle time).

mincyc_cycle — the number of the minor cycle in which the minimum cycle time has occurred (minimum cycle cycle).

mincyc_frame — the number of the major frame in which the minimum cycle time has occurred (minimum cycle frame).

maxcyc_tm the greatest amount of time that the process has spent running in a cycle (maximum cycle time).

maxcyc_cycle the number of the minor cycle in which the maximum cycle time has occurred (maximum cycle cycle).

maxcyc_frame the number of the major frame in which the maximum cycle time has occurred (maximum cycle frame).

minframe_tm the least amount of time that the process has spent running during a major frame (minimum frame time).

minframe the number of the major frame in which the minimum frame time has occurred (minimum frame frame).

maxframe_tm the greatest amount of time that the process has spent running during a major frame (maximum frame time).

maxframe the number of the major frame in which the maximum frame time has occurred (maximum frame frame).

buf_cnt an integer value indicating the number of structures in the array to which *pm_buf* points.

Return Value

A return value of 0 indicates that the call has been successful. A return value of -1 indicates that an error has occurred; `errno` is set to indicate the error. Refer to the **`pmqrycpu(3)`** and **`pmqrycpu_big(3)`** man pages for a listing of the types of errors that may occur.

pmqrylist – Query Values for a List of Processes

This routine gets performance monitor values for a list of processes scheduled on a frequency-based scheduler.

Synopsis

```
#include <fbsched.h>

int  pmqrylist(fbs_id, pm_buf, buf_cnt)
int fbs_id, buf_cnt;
struct pmqry_ds {
  int                fpid;
  struct timespec    lastcyc_tm;
  int                tot_cycles;
  struct timespec    tot_cycles_tm;
  int                overruns;
  struct timespec    mincyc_tm;
  int                mincyc_cycle;
  int                mincyc_frame;
  struct timespec    maxcyc_tm;
  int                maxcyc_cycle;
  int                maxcyc_frame;
  struct timespec    minframe_tm;
  int                minframe;
  struct timespec    maxframe_tm;
  int                maxframe;
} *pm_buf;
```

Parameters

fbs_id — a unique positive integer value representing the identifier for a frequency-based scheduler. This value can be obtained using **fbsid(3)** (page 6-18) or **fbsconfigure(3)** (page 6-10).

Enter MY_FBS to reference the frequency-based scheduler on which the calling process is scheduled without knowing the identifier.

pm_buf — a pointer to an array of pmqry_ds structures to which **pmqrylist** returns the performance monitor values for a list of frequency-based scheduled processes.

The list of processes for which values are returned is created by placing the frequency-based scheduler identifier in the *fpid* component of each structure in the array. The type of information contained in each component of the structure for a single process is presented below.

fpid — the process' frequency-based scheduler process identifier.

lastcyc_tm — the amount of time that the process has spent running from the last time that it was awakened by the scheduler until it called **fbswait**.

tot_cycles — the number of times that the process has been awakened by the scheduler (total iterations, or cycles).

tot_cycles_tm the time that the process has spent running in all cycles.

overruns the number of hard frame overruns caused by the process.

mincyc_tm the least amount of time that the process has spent running in a cycle (minimum cycle time).

mincyc_cycle the number of the minor cycle in which the minimum cycle time has occurred (minimum cycle cycle).

mincyc_frame the number of the major frame in which the minimum cycle time has occurred (minimum cycle frame).

maxcyc_tm the greatest amount of time that the process has spent running in a cycle (maximum cycle time).

maxcyc_cycle the number of the minor cycle in which the maximum cycle time has occurred (maximum cycle cycle).

maxcyc_frame the number of the major frame in which the maximum cycle time has occurred (maximum cycle frame).

minframe_tm the least amount of time that the process has spent running during a major frame (minimum frame time).

minframe the number of the major frame in which the minimum frame time has occurred (minimum frame frame).

maxframe_tm the greatest amount of time that the process has spent running during a major frame (maximum frame time).

maxframe the number of the major frame in which the maximum frame time has occurred (maximum frame frame).

buf_cnt an integer value indicating the number of structures in the array to which *pm_buf* points.

Return Value

A return value of 0 indicates that the call has been successful. A return value of -1 indicates that an error has occurred; `errno` is set to indicate the error. Refer to the **`pmqrylist(3)`** man page for a listing of the types of errors that may occur.

pmqrypgm, pmqrypgm_big – Query Values for a Selected Process

These routines get performance monitor values for a particular process scheduled on a frequency-based scheduler. Identify the process by using one of the following methods:

- Specify the name of the process and the CPU(s) on which it is scheduled.
- Specify the process' frequency-based scheduler process identifier.
- Specify the name of the process, the CPU(s) on which it is scheduled, and its frequency-based scheduler process identifier.

These two routines are identical except for their treatment of CPU masks. For details, see "The Big-SMP FBS Interface" on page 6-2.

Synopsis

```
#include <fbsched.h>

int  pmqrypgm(fbs_id, name, cpu, pm_buf)
int  fbs_id, cpu;
char *name;
struct pmqry_ds {
  int               fpid;
  struct timespec   lastcyc_tm;
  int               tot_cycles;
  struct timespec   tot_cycles_tm;
  int               overruns;
  struct timespec   mincyc_tm;
  int               mincyc_cycle;
  int               mincyc_frame;
  struct timespec   maxcyc_tm;
  int               maxcyc_cycle;
  int               maxcyc_frame;
  struct timespec   minframe_tm;
  int               minframe;
  struct timespec   maxframe_tm;
  int               maxframe;
} *pm_buf;

int  pmqrypgm_big(fbs_id, name, cpuset, pm_buf)
int  fbs_id;
char *name;
cpuset_t *cpuset;
struct pmqry_ds *pm_buf;
```

Parameters

fbs_id — a unique positive integer value representing the identifier for a frequency-based scheduler. This value can be obtained using **fbsid(3)** (page 6-18) or **fbsconfigure(3)** (page 6-10).

Enter MY_FBS to reference the frequency-based scheduler on which the calling process is scheduled without knowing the identifier.

name — a pointer to a variable that contains a path name identifying the process for which performance monitoring values are to be returned. A full or

relative path name of up to 1024 characters can be specified. If this variable is the null string, you must provide the frequency-based scheduler process identifier in the *fpid* component of the structure to which *pm_buf* points.

cpu an integer value indicating the processor(s) to be used in conjunction with the value of the *name* parameter to identify the process for which performance monitoring values are to be returned. Acceptable values and corresponding results are presented below.

0	the first process named by *name* currently running on the processor from which the call is made is specified.
-1	the first process named by *name* currently running on any processor is specified.

bit mask	if (*cpu* & (1<<*i*)) is set (where *i* is an integer ranging from 0 to 15 and representing a CPU) and it is the only bit set, the first process named by *name* currently running on CPU *i* is specified.
	
	If (*cpu* & (1<<*i*)) is set and it is not the only bit set, the first process named by *name* currently running on any of the selected CPUs is specified.

cpuset a pointer that directly or indirectly indicates the set of porocessors, to be used in conjunction with *name*, to identify the process for which performance monitoring values are to be returned. The pointer may either point to a valid `cpuset`, as created by **`cpuset_alloc(3)`** and filled in with some set of CPU IDs by other **`cpuset(3)`** operations, or it may take on one of the following special pointer values:

`FBS_CPUSET_CURCPU`	the first process named by *name* currently running on the processor from which the call is made.
`FBS_CPUSET_ALLCPUS`	the first process named by *name* running on any processor.
`FBS_CPUSET_ONECPU(`*`cpu`*`)`	the first process named by *name* running on the processor with a CPU ID of *cpu*.

If *cpuset* points to a **`cpuset(3)`** that is empty, the the behavior is as described for `FBS_CPUSET_CURCPU`.

pm_buf a pointer to a `pmqry_ds` structure to which **`pmqrypgm`** returns the performance monitor values for the frequency-based scheduled process pointed to by the *name* parameter.

The type of information contained in each component of the structure is presented below:

fpid an integer value providing the unique frequency-based scheduler process identifier for which performance monitor values are to be returned.

Get this value by calling **`sched_pgmadd`** (see page 6-51). This value must be -1 if you wish to identify the process only by specifying *name* and *cpu*.

lastcyc_tm the amount of time that the process has spent running from the last time that it was awakened by the scheduler until it called **`fbswait`**.

tot_cycles the number of times that the process has been awakened by the scheduler since the last time that performance monitor values have been cleared and performance monitoring has been enabled (total iterations, or cycles).

tot_cycles_tm the time that the process has spent running in all cycles.

overruns the number of hard frame overruns caused by the process.

mincyc_tm the least amount of time that the process has spent running in a cycle (minimum cycle time).

mincyc_cycle the number of the minor cycle in which the minimum cycle time has occurred (minimum cycle cycle).

mincyc_frame the number of the major frame in which the minimum cycle time has occurred (minimum cycle frame).

maxcyc_tm the greatest amount of time that the process has spent running in a cycle (maximum cycle time).

maxcyc_cycle the number of the minor cycle in which the maximum cycle time has occurred (maximum cycle cycle).

maxcyc_frame the number of the major frame in which the maximum cycle time has occurred (maximum cycle frame).

minframe_tm the least amount of time that the process has spent running during a major frame (minimum frame time).

minframe the number of the major frame in which the minimum frame time has occurred (minimum frame frame).

maxframe_tm the greatest amount of time that the process has spent running during a major frame (maximum frame time).

maxframe the number of the major frame in which the maximum frame time has occurred (maximum frame frame).

Return Value

A return value of 0 indicates that the call has been successful. A return value of -1 indicates that an error has occurred; `errno` is set to indicate the error. Refer to the **`pmqrypgm(3)`** and **`pmqrypgm_big(3)`** man pages for a listing of the types of errors that may occur.

pmqrytimer – Query Performance Monitor Mode

This routine determines whether performance monitor timing values include or exclude time spent servicing interrupts.

Synopsis

```
#include <fbsched.h>

int pmqrytimer()
```

Return Value

A return value of 0 indicates that interrupt time is excluded from performance monitor timing values. A return value of 1 indicates that interrupt time is included in timing values. A return value of -1 indicates that an error has occurred; `errno` is set to indicate the error. Refer to the **`pmqrytimer(3)`** man page for a listing of the types of errors that may occur.

pmselect – Select Performance Monitor Mode

This routine selects the timing mode under which the Performance Monitor is to run. The timing mode can be set to include or exclude time spent servicing interrupts.

When interrupt time is included, a process' user and system times will total the elapsed time which accrues when the process is the currently running process, including all time spent servicing interrupts. The time spent servicing interrupts is added to the process' system time.

When excluding interrupt time, a process' user and system times will total the time which accrues when the process is the currently running process, excluding all time spent servicing interrupts.

Synopsis

```
#include <fbsched.h>

int pmselect(mode)
int mode;
```

Parameters

mode — an integer value indicating whether time spent servicing interrupts is to be included in or excluded from performance monitor timing values.

A nonzero value indicates that interrupt time is to be included. A value of zero indicates that interrupt time is to be excluded.

Return Value

A return value of 0 indicates that the call has been successful. A return value of -1 indicates that an error has occurred; `errno` is set to indicate the error. Refer to the **`pmselect(3)`** man page for a listing of the types of errors that may occur.

7
The FORTRAN Library Interface

This chapter describes the FORTRAN library interface to the Frequency-Based Scheduler and the Performance Monitor. Library information, call sequences for using the scheduler and the Performance Monitor, and details for each of the library routines are provided.

Overview

Access to the functions associated with the Frequency-Based Scheduler and the Performance Monitor is provided through libraries of routines that can be called from application programs written in FORTRAN.

The following information is provided in this chapter for each routine:

- a description of the routine
- the FORTRAN variable declarations and call statement needed to reference the routine in an application program
- detailed descriptions of each parameter

Information about the individual routines, including an illustration of the sequence in which you would call the routines during frequency-based scheduling, begins on page 7-3. The same information is provided for performance monitoring beginning on page 7-65.

For compatibility with the PowerMAX operating system, **`usermap(3f)`** is provided to map a target process' memory into address space. Refer to the man page for details.

Compiling and Linking Procedures

When statically linking a FORTRAN program, the following libraries are required:

```
/usr/lib/libccur_fbsched.a
/usr/lib/libccur_rt.a
/usr/lib/libF77rt.a
```

When dynamically linking a C program, the following libraries are used:

```
/usr/lib/libccur_fbsched.so
/usr/lib/libccur_rt.so
/usr/lib/libF77rt.so
```

To compile and link a FORTRAN program, the command line instruction is as follows:

`cf77` [*options ...*] *file*`.f -lF77rt -lccur_fbsched -lccur_rt`

The Big-SMP FBS Interface for Fortran

The Fortran language has not yet been given an API to the Big-SMP FBS interface. Therefore, Fortran applications that need access to the Big-SMP API should do so indirectly; for example, by invoking FBS services through the **`rtcp(1)`** command, or by writing and linking C routines that invoke the desired Big-SMP FBS services for the application

Frequency-Based Scheduler Routines

Frequency-Based Scheduler routines provide access to the key features of the scheduler. They enable you to perform such basic operations as:

- configuring a scheduler
- scheduling programs on a scheduler
- setting up and connecting a timing source to a scheduler
- starting, stopping and resuming scheduling on a scheduler
- getting information about scheduled processes
- rescheduling and removing scheduled processes
- disconnecting a timing source
- removing a scheduler

Routine Summary

Frequency-Based Scheduler routines are summarized in Table 7-1. Complete information about each routine is provided under the section "Using Frequency-Based Scheduler Routines."

Table 7-1. Frequency-Based Scheduler FORTRAN Library Routines

Routine	Page	Description
fbsaccess	7-6	Change permissions for a frequency-based scheduler
fbsattach	7-8	Attach timing source to a frequency-based scheduler
fbsconfigure	7-9	Configure a frequency-based scheduler
fbscycle	7-11	Return minor cycle/major frame count
fbsdetach	7-12	Detach timing source from a frequency-based scheduler
fbsgetrtc	7-13	Get real-time clock values
fbsid	7-14	Return the frequency-based scheduler identifier for a key
fbsinfo	7-15	Return information for a frequency-based scheduler
fbsintrpt	7-17	Start/stop/resume scheduling
fbsquery	7-18	Query processes on a frequency-based scheduler
fbsremove	7-21	Remove a frequency-based scheduler
(continued on next page)		
fbsresume	7-22	Resume scheduling on a frequency-based scheduler

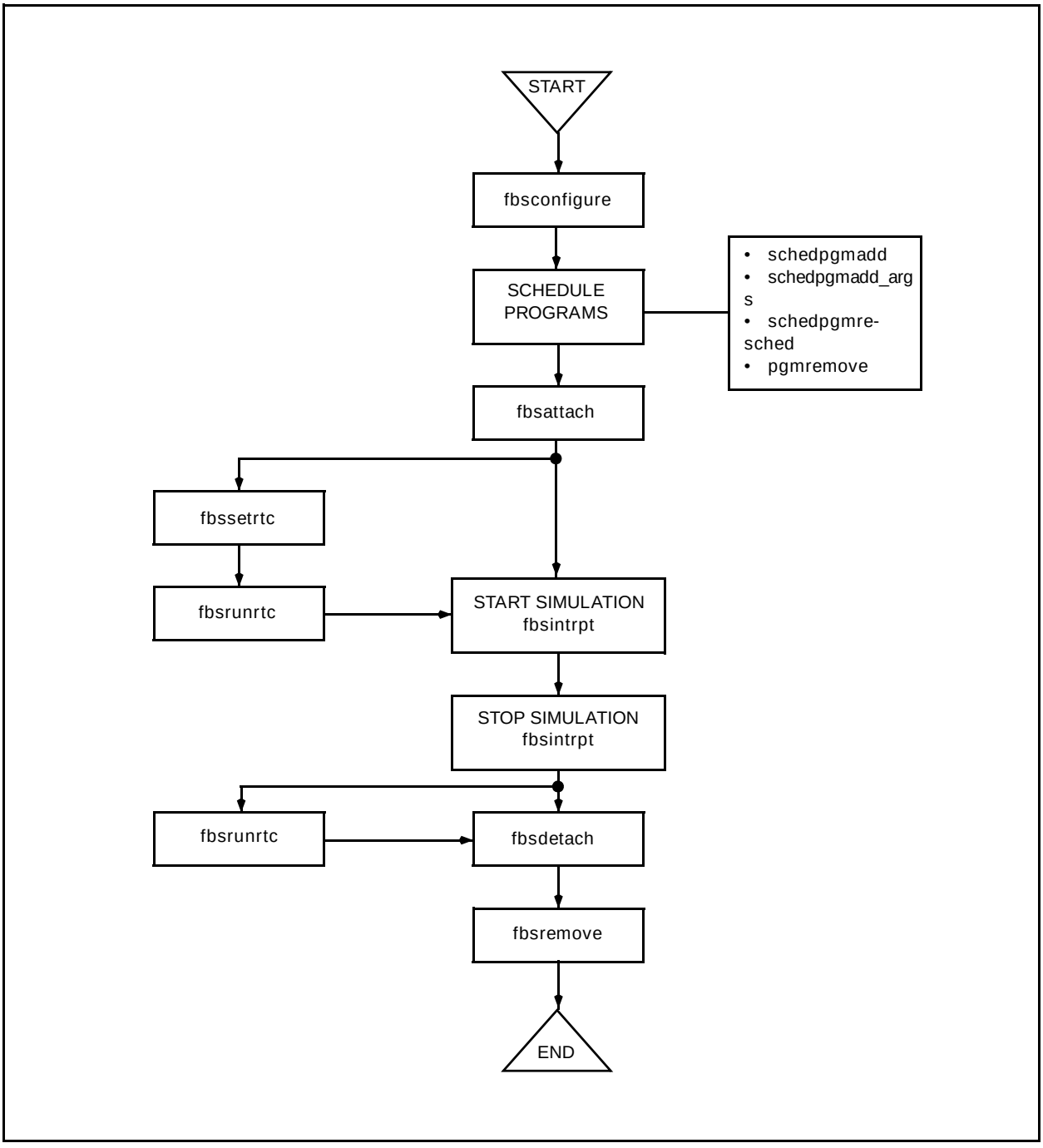
Table 7-1. Frequency-Based Scheduler FORTRAN Library Routines (Cont.)

Routine	Page	Description
fbsrunrtc	7-24	Start/stop real-time clock
fbsschedself	7-25	Add a calling process to a frequency-based scheduler
fbssetrtc	7-27	Set real-time clock
fbswait	7-28	Wait on a frequency-based scheduler
nametopid	7-29	Return the process ID
pgmquery	7-30	Query processes on a frequency-based scheduler
pgmremove	7-32	Remove a process from a frequency-based scheduler
pgmreschedule	7-34	Reschedule a process
pgmschedule	7-37	Schedule a process on a frequency-based scheduler
pgmstat	7-40	Query the state of an FBS-scheduled process
pgmtrigger	7-42	Trigger a process on a frequency-based scheduler
rtparm	7-43	Return initiation parameter
sched_pgm_deadline_query	7-44	Query assigned deadline for a process
sched_pgm_deadline_test	7-46	Test for the presence of a deadline violation
sched_pgm_set_deadline	7-48	Set or clear process deadline time
sched_pgm_set_soft_overrun_limit	7-50	Set soft overrun limit
sched_pgm_soft_overrun_query	7-51	Query soft overrun processing
schedfbsqry	7-52	Query processes on a frequency-based scheduler
schedpgmadd	7-55	Schedule a process on a frequency-based scheduler
schedpgmadd_args	7-57	Schedule a process on a frequency-based scheduler with arguments
schedpgmqry	7-59	Query a process
schedpgmresched	7-62	Reschedule a process

FORTRAN Library Call Sequence

The approximate order in which you might call the routines from an application program is illustrated in Figure 7-1.

Figure 7-1. FORTRAN Library Call Sequence: Frequency-Based Scheduler



Using Frequency-Based Scheduler Routines

In the sections that follow, the Frequency-Based Scheduler routines contained in the **libF77rt** library are presented in alphabetical order.

fbsaccess – Change Permissions for a Frequency-Based Scheduler

This subroutine changes the permissions assigned for a selected frequency–based scheduler. Permissions can be changed only by a process that has an effective user ID that is equal to that of the owner/creator of the frequency–based scheduler.

Call Statement

```
call fbsaccess (schdle, uid, gid, permissions, istat)
integer schdle, uid, gid, permissions, istat
```

Parameters

schdle — a unique, positive integer value representing the identifier for a frequency–based scheduler. You can obtain this value by making a call to **fbsconfigure(3f)** (see page 7-9 for an explanation of this subroutine) or **fbsid(3f)** (see page 7-14). To reference the frequency–based scheduler on which the calling process is scheduled without knowing the identifier, you can specify the value **−1**.

uid — an integer value representing the effective user ID of the specified frequency–based scheduler.

gid — an integer value representing the effective group ID of the specified frequency–based scheduler.

permissions — a bit pattern used to set the permissions associated with the specified frequency–based scheduler. Bit patterns and corresponding permissions are presented in Table 7-2.

Table 7-2. Frequency-Based Scheduler Permissions

Bit Pattern	Permissions
400	Read by user
200	Alter by user
060	Read, alter by group
006	Read, alter by others

istat — an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A nonzero value indicates that an error of a specific type has occurred. Refer to the **fbsaccess(3f)** man page for a

listing of the nonzero values that may be returned and the types of errors that they represent.

fbsattach – Attach Timing Source to a Frequency-Based Scheduler

This subroutine is invoked to attach a timing source to a frequency–based scheduler or to specify end–of–cycle scheduling. The timing source can be a real–time clock or an edge–triggered interrupt device.

Call Statement

`call fbsattach(`*schdle, cpu, devname, istat*`)`
`integer` *schdle, cpu, istat*
`character* (*)` *devname*

Parameters

schdle — a unique, positive integer value representing the identifier for the frequency–based scheduler for which the timing source is to be attached or end–of–cycle scheduling specified. You can obtain this value by making a call to **`fbsconfigure(3f)`** (see page 7-9 for an explanation of this subroutine) or **`fbsid(3f)`** (see page 7-14). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing its identifier, you can specify a value of – **`1`**.

cpu — a variable that must contain the value **`0`**.

devname — a null string or the path name of the device that is to be used as the timing source for the specified scheduler. If *devname* contains a null string, end–of–cycle scheduling is specified; that is, execution of the processes in the next minor cycle will occur when the last process scheduled to execute in the current minor cycle finishes its execution for that cycle.

If *devname* contains a path name, it may refer to a real-time clock or an edge-triggered interrupt. See Chapter 3 for information about timing source device files.

istat — an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A nonzero value indicates that an error of a specific type has occurred. Refer to the **`fbsattach(3f)`** man page for a listing of the nonzero values that may be returned and the types of errors that they represent.

fbsconfigure – Configure a Frequency-Based Scheduler

This subroutine is invoked to configure a frequency–based scheduler or to obtain configuration details for a frequency–based scheduler that has already been configured. Note that to configure a scheduler, the calling process must have the capabilities associated with the *fbscheduser* role (for additional information on privileges, refer to Chapter 1).

If you wish to configure a scheduler, you must specify a *key*, which is a user–chosen numeric identifier for a frequency–based scheduler. The value for *cycles* indicates the function to be performed:

- If the value specified in *cycles* is NOT equal to 0, and if the number of existing frequency-based schedulers is less than the system imposed limit (default = 10) a frequency-based scheduler is created and an identifier is returned.
- If the value of *cycles* is equal to 0, the current configuration for the frequency-based scheduler is returned.

Call Statement

`call fbsconfigure(`*key, cycles, progs, max, reset, configflg, schdle, istat*`)`
`integer` *key, cycles, progs, max, reset, configflg, schdle, istat*

Parameters

To create a frequency-based scheduler, you must specify the following parameters as described.

key — an integer value identifying the frequency–based scheduler that is to be created.

cycles — an integer value indicating the number of minor cycles that compose a frame on the specified scheduler.

progs — an integer value indicating the maximum number of programs that can be scheduled to execute during one minor cycle.

max — an integer value indicating the maximum number of programs that can be scheduled on the specified scheduler at one time. This value must be less than or equal to the product that is obtained by multiplying the values specified for the *cycles* and *progs* parameters.

reset — an integer value indicating whether or not processes currently scheduled on the specified scheduler are to be killed before the scheduler is reconfigured. Acceptable values and corresponding results are as follows:

<0	Kill and remove all processes currently scheduled on the specified scheduler
0	Ignore all processes currently scheduled on the specified scheduler
>0	Remove all processes currently scheduled on the specified scheduler

configflg — an integer value indicating the permissions assigned to the specified scheduler.

schdle — a unique, positive integer value representing the identifier for the specified frequency–based scheduler. It is important to note that this identifier is required by most of the library subroutines for the FBS and the performance monitor.

istat — an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A nonzero value indicates that an error of a specific type has occurred. Refer to the **`fbsconfigure(3f)`** man page for a listing of the nonzero values that may be returned and the types of errors that they represent.

To obtain information for an existing frequency–based scheduler, you must specify the following parameters as described.

key — an integer value identifying the frequency–based scheduler for which configuration information is to be returned. If this value is zero, the frequency–based scheduler identifier associated with this scheduler must also be provided by using the *schdle* parameter.

cycles — the integer value zero, indicating that current configuration information for the specified scheduler is to be returned. **`Fbsconfigure`** will return to this variable an integer value indicating the number of minor cycles that compose a frame on the specified scheduler.

progs — the maximum number of programs that can be scheduled to run during one minor cycle on the specified scheduler.

max — the maximum number of programs that can be scheduled on the specified scheduler at one time.

configflg — the permissions assigned to the specified scheduler.

schdle — a unique, positive integer value representing the identifier for the specified frequency–based scheduler. If you specify a key of **`0`**, this variable must contain the related frequency–based scheduler identifier.

fbscycle – Return Minor Cycle/Major Frame Count

This subroutine is invoked to obtain the current minor cycle and major frame count values for a frequency–based scheduler. These values enable you to determine the progress of a simulation.

Call Statement

`call fbscycle(`*schdle, count, istat*`)`
`integer` *schdle, count(2), istat*

Parameters

schdle	a unique, positive integer value representing the identifier for the frequency–based scheduler for which you wish to obtain the current cycle and frame counts. You can obtain this value by making a call to **`fbsconfigure`** (see page 7-9 for an explanation of this subroutine) or **`fbsid(3f)`** (see page 7-14). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing its identifier, you can specify a value of –1.
count	an array containing integer values indicating the current minor cycle and major frame for the specified scheduler. *Count(1)* will contain the current minor cycle. *Count(2)* will contain the major frame count.
istat	an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A nonzero value indicates that an error of a specific type has occurred. Refer to the **`fbscycle(3f)`** man page for a listing of the nonzero values that may be returned and the types of errors that they represent.

fbsdetach – Detach Timing Source from a Frequency-Based Scheduler

This subroutine is invoked to detach the currently attached timing source from a frequency–based scheduler or to disable end–of–cycle scheduling. If the timing source is a real-time clock, it is recommended that you stop the clock prior to invoking this subroutine. You can do so by making a call to **fbsrunrtc** (see page 7-24 for an explanation of this subroutine).

Call Statement

```
call fbsdetach(schdle, istat)
integer schdle, istat
```

Parameters

schdle — a unique, positive integer value representing the identifier for the frequency–based scheduler from which you wish to detach the currently attached timing source or for which you wish to disable end–of–cycle scheduling. You can obtain this value by making a call to **fbsconfigure(3f)** (see page 7-9 for an explanation of this subroutine) or **fbsid(3f)** (see page 7-14). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing its identifier, you can specify a value of **– 1**.

istat — an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A nonzero value indicates that an error of a specific type has occurred. Refer to the **fbsdetach(3f)** man page for a listing of the nonzero values that may be returned and the types of errors that they represent.

fbsgetrtc – Obtain Current Values for Real–Time Clock

This subroutine is invoked to obtain the current count and resolution values for the real–time clock that is attached to a specified frequency–based scheduler.

Call Statement

`call fbsgetrtc(`*schdle, count, resolution, istat1, istat2*`)`
`integer` *schdle, count, resolution, istat1, istat2*

Parameters

Parameters must be specified in the order indicated. They are described as follows.

schdle — a unique, positive integer value representing the identifier for the frequency–based scheduler to which the real–time clock is attached. You can obtain this value by making a call to **`fbsconfigure(3f)`** (see page 7-9 for an explanation of this subroutine) or **`fbsid(3f)`** (see page 7-14). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing its identifier, you can specify a value of **`–1`**.

count — an integer value indicating the current number of clock counts per minor cycle. This value can range from one to 65535.

resolution — an integer value indicating the current duration in microseconds of one clock count. This value will be one of the following: 1, 10, 100, 1000, or 10000.

istat1 — an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A nonzero value indicates that an error of a specific type has occurred. Refer to the **`fbsgetrtc(3f)`** man page for a listing of the nonzero values that may be returned and the types of errors that they represent. If *istat1* contains a value indicating that an error has occurred on an **`open`** or **`ioctl`** call, the error status of that call is returned in *istat2*.

istat2 — the error status of an **`open`** or **`ioctl`** call. See the include file **`<errno.h>`** for a description of the errors.

fbsid – Return the FBS Identifier for a Key

This subroutine is invoked to obtain the frequency–based scheduler identifier associated with a particular user–specified key. The key must match the key that was specified when the scheduler was created by making a call to **`fbsconfigure(3f)`**.

Call Statement

`call fbsid(`*key, schdle, istat*`)`
`integer` *key, schdle, istat*

Parameters

Parameters must be specified in the order indicated. They are described as follows.

key	an integer value identifying a frequency–based scheduler; this value must be the same value that was specified for *key* when the scheduler was created by making a call to **`fbsconfigure`** (see page 7-9 for an explanation of this subroutine).
schdle	an integer value representing the unique frequency–based scheduler identifier associated with the key.
istat	an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A nonzero value indicates that an error of a specific type has occurred. Refer to the **`fbsid(3f)`** man page for a listing of the nonzero values that may be returned and the types of errors that they represent.

fbsinfo – Return Information for a Frequency-Based Scheduler

This subroutine is invoked to obtain information that is related to a selected frequency–based scheduler but cannot be obtained by invoking other subroutines (for example, **schedfbsqry**, **schedpgmqry**). Such information includes the following:

- The user and group IDs of the owner and the creator of the scheduler
- The permissions assigned for the scheduler
- The key associated with the scheduler's identifier
- The total number of overruns for all processes on the scheduler
- The CPUs that are active in the system
- The CPUs on which performance monitoring has been enabled
- The FBS–enabled flag
- The path name of the device that has been attached to the scheduler

Call Statement

```
call fbsinfo(schdle, buf, devname, istat)
integer schdle, buf(41), istat
character* (*) devname
```

Parameters

Parameters are described as follows.

schdle a unique, positive integer value representing the identifier for a frequency–based scheduler. You can obtain this value by making a call to **fbsconfigure(3f)** (see page 7-9 for an explanation of this subroutine) or **fbsid(3f)** (see page 7-14). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing its identifier, you can specify a value of **−1**.

buf an array containing information about the specified scheduler. The information returned in each element of the array is presented in Table 7-3.

Table 7-3. Contents of Array Elements: fbsinfo

Element	Contents
buf(1)	owner's user ID
buf(2)	owner's group ID
buf(3)	creator's user ID
buf(4)	creator's group ID
buf(5)	access modes
buf(6)	key
buf(7)	flags word
buf(8)	reserved for future use

Table 7-3. Contents of Array Elements: fbsinfo (Cont.)

Element	Contents
buf(9)	total number of hard overruns for all processes on the scheduler
buf(10)	mask of CPUs active in the system
buf(11)	mask of CPUs on which performance monitoring has been enabled
buf(12)	FBS–enabled flag
buf(13)–(41)	reserved for future use

devname — the path name of the device that is being used as the timing source for the specified frequency–based scheduler. If end–of–cycle scheduling has been specified, *devname* will contain a null string.

istat — an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A nonzero value indicates that an error of a specific type has occurred. Refer to the **`fbsinfo(3f)`** man page for a listing of the nonzero values that may be returned and the types of errors that they represent.

fbsintrpt – Start/Stop/Resume Scheduling on a Frequency-Based Scheduler

This subroutine is invoked to start, stop, or resume scheduling on a frequency–based scheduler. If you invoke this subroutine to start scheduling, the minor cycle, major frame, and overrun count values are reset. If you invoke it to resume scheduling, these values are not reset.

Prior to invoking **fbsintrpt**, you must have invoked **fbsattach** to specify end–of–cycle scheduling or attach a timing source to the frequency–based scheduler on which you are starting scheduling (see page 7-8 for an explanation of **fbsattach**). If you have specified a real–time clock as the timing source, scheduling will not begin until you have set and started the clock (see pages 7-27 and 7-24 for explanations of **fbssetrtc** and **fbsrunrtc**, respectively). If you have specified an edge–triggered interrupt device as the timing source, it must already be generating interrupts in order for scheduling to start.

Call Statement

`CALL fbsintrpt(`*schdle, intrflag, istat*`)`
`integer` *schdle, intrflag, istat*

Parameters

schdle — a unique, positive integer value representing the identifier for the frequency–based scheduler on which you wish to start, stop, or resume scheduling of processes. You can obtain this value by making a call to **fbsconfigure(3f)** (see page 7-9 for an explanation of this subroutine) or **fbsid(3f)** (see page 7-14). If you wish to reference the frequency-based scheduler on which the calling process is scheduled without knowing its identifier, you can specify a value of **–1**.

intrflag — an integer value indicating whether scheduling of processes on the specified scheduler is to be started, stopped, or resumed. Acceptable values and results are as follows:

- <0 Start scheduling of processes with the initial frame, cycle, and overrun count values set to zero
- 0 Stop scheduling of processes, and save the count values for the current frame and cycle
- >0 Resume scheduling of processes with the frame, cycle, and overrun count values set to the values that were saved when the scheduler was last stopped

istat — an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A nonzero value indicates that an error of a specific type has occurred. Refer to the **fbsintrpt(3f)** man page for a listing of the nonzero values that may be returned and the types of errors that they represent.

fbsquery – Query Processes on a Frequency-Based Scheduler

CAUTION

This interface is obsolete. It is maintained for compatibility with CX/UX, but it returns processes' scheduling priorities without any indication of the scheduling policies with which they are associated. If you have an existing application that uses this interface, it is recommended that you change your application to use **schedfbsqry(3f)** (see p. 7-52).

This subroutine is invoked to obtain information about processes that have been scheduled on a frequency–based scheduler. Information is returned for all processes scheduled on the user–specified processor(s). Information provided for each process includes the following:

- A mask of the CPU(s) on which the process can execute
- The frequency–based scheduler process identifier
- The scheduling priority
- The period (the number of minor cycles indicating the frequency with which the process is wakened in each major frame)
- The starting base cycle (the first minor cycle in which the process is scheduled to be wakened in each major frame)
- The value of the "halt on overrun" flag

Call Statement

```
CALL fbsquery(schdle, cpu, buf1size, buf1, maxsize, buf2size, buf2, istat)
integer schdle, cpu, buf1size, buf1(buf1size), maxsize, buf2size, istat
character* (*) buf2
```

Parameters

schdle — a unique, positive integer value representing the identifier for the frequency–based scheduler for which you wish to obtain scheduling information. You can obtain this value by making a call to **fbsconfigure(3f)** (see page 7-9 for an explanation of this subroutine) or **fbsid(3f)** (see page 7-14). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing its identifier, you can specify a value of **–1**.

cpu — an integer value indicating the processor(s) for which scheduling information is to be obtained. Acceptable values and corresponding results are as follows:

0	Scheduling information for processes executing on the processor from which the call is made is returned
-1	Scheduling information for all processes on the scheduler is returned

Bit mask If (*cpu* & (1<<*i*)) is set (where *i* is an integer ranging from zero to 15 and representing a CPU), scheduling information for processes executing on CPU *i* is returned

buf1size an integer value indicating the size in 32–bit words of the array represented by *buf1*. Because 10 words of information are returned for each process, it is recommended that this value be a multiple of 10.

buf1 an array containing a series of 10 integer values for each process on the processor(s) specified with the *cpu* parameter. The number of processes for which these values are returned is bound by the value of the *buf1size* parameter. If, for example, the value of *buf1size* is 145, values for 14 processes will be returned. These values represent the scheduling information for the process(es). The type of information returned in each array element for a single process is presented in Table 7-4.

Table 7-4. Contents of Array Elements: fbsquery

Element	Contents
1	Byte offset of the process' path name in *buf2*
2	Length in bytes of the process' path name
3	Zero
4	Zero
5	Mask of the CPU(s) on which the process can execute
6	The process' frequency–based scheduler process identifier
7	The process' scheduling priority
8	The number of minor cycles indicating the frequency with which the process is to be wakened in each major frame (period)
9	The first minor cycle in which the process is scheduled to be wakened in each major frame (starting base cycle)
10	The value of the "halt on overrun" flag. A nonzero value indicates that the flag is set. A value of zero indicates that the flag is not set.

maxsize an integer value indicating the maximum length of a path name to be returned in *buf2*

buf2size an integer value indicating the size in bytes of the character string represented by *buf2*. To ensure that *buf2* is large enough to accommodate the names of all processes that you wish to query, you may find it helpful to compute the number of bytes needed by multiplying the maximum number of processes allowed on the scheduler (see the information on **`fbsconfigure`** presented on page 7-9) by 32.

buf2	the path names for each process on the processor(s) specified with the *cpu* parameter. Path names are returned as a series of strings. The length of each string is less than or equal to the value of *maxsize*. Where *maxsize* is not large enough to accommodate a full path name, the concluding component names are returned. The number of path names returned is bound by the value of the *buf2size* parameter.
istat	an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A nonzero value indicates that an error of a specific type has occurred. Refer to the **`fbsquery(3f)`** man page for a listing of the nonzero values that may be returned and the types of errors that they represent.

fbsremove – Remove a Frequency-Based Scheduler

This subroutine is invoked to remove a frequency–based scheduler and to free the data structure associated with it. It is important to note that prior to invoking **fbsremove**, you must ensure that the timing source is detached from the scheduler or that end–of–cycle scheduling is disabled (see page 7-12 for information on the use of **fbsdetach**). It is important to note that **fbsremove** will remove all processes scheduled on the specified scheduler. It is recommended, however, that you remove all scheduled processes prior to invoking **fbsremove**. You can do so by making a call to **pgmremove** (see page 7-32 for information on the use of this subroutine).

Note that to remove a frequency-based scheduler, the calling process must have an effective user ID that is equal to that of the owner/creator of the scheduler.

Call Statement

```
call fbsremove(schdle, ab, istat)
integer schdle, ab, istat
```

Parameters

schdle — a unique, positive integer value representing the identifier for the frequency–based scheduler that you wish to remove. You can obtain this value by making a call to **fbsconfigure(3f)** (see page 7-9 for an explanation of this subroutine) or **fbsid(3f)** (see page 7-14). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing its identifier, you can specify a value of **−1**.

ab — an integer value indicating the manner in which processes scheduled on the scheduler are to be handled. Acceptable values and corresponding results are as follows:

<0	Kill and remove all processes currently scheduled on the specified scheduler
≥0	Remove all processes currently scheduled on the specified scheduler

istat — an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A nonzero value indicates that an error of a specific type has occurred. Refer to the **fbsremove(3f)** man page for a listing of the nonzero values that may be returned and the types of errors that they represent.

fbsresume – Resume Scheduling on a Frequency-Based Scheduler

The **fbsresume** subroutine is invoked to resume scheduling of processes on a frequency-based scheduler at the specified minor cycle, major frame, and overrun count.

Note that to resume scheduling of processes on a frequency-based scheduler, the calling process must have alter permission for the scheduler.

If you wish to resume scheduling of processes on a frequency-based scheduler without altering the scheduler's current frame, cycle, and overrun values, it is recommended that you use the **fbsintrpt(3f)** subroutine (see page 7-17 for an explanation of this subroutine).

CAUTION

The **fbsresume** subroutine clears performance monitor values for all processes scheduled on the specified scheduler. Changing the frame and cycle count for the scheduler causes the values that are being maintained by the performance monitor to be inaccurate.

Call Statement

```
call fbsresume(schdle, frame, cycle, overruns, istat)
integer schdle, frame, cycle, overruns, istat
```

Parameters

schdle — a unique, positive integer value representing the identifier for the frequency–based scheduler on which you wish to resume scheduling of processes. You can obtain this value by making a call to **fbsconfigure(3f)** (see page 7-9 for an explanation of this subroutine) or **fbsid(3f)** (see page 7-14). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing the identifier, you can specify the value –1.

frame — an integer value indicating the major frame in which you wish scheduling of processes to be resumed on the specified scheduler

cycle — an integer value indicating the minor cycle in which you wish scheduling of processes to be resumed on the specified scheduler. This value can range from zero to the total number of minor cycles per frame minus one. The total number of minor cycles per frame was specified when the scheduler was created by making a call to **fbsconfigure** (see page 7-9 for an explanation of this subroutine).

overruns — an integer value indicating the value to which you wish the overrun count to be set when scheduling resumes on the specified scheduler. If you do not wish to change the overrun count, you can specify the value –1.

istat	an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A nonzero value indicates that an error of a specific type has occurred. Refer to the **fbsresume(3f)** man page for a listing of the nonzero values that may be returned and the types of errors that they represent.

fbsrunrtc – Start/Stop Real–Time Clock

This subroutine is invoked to start or stop the counting of a real–time clock that has been attached to a frequency–based scheduler.

Call Statement

`call fbsrunrtc(`*schdle, runflag, istat1, istat2*`)`
`integer` *schdle, runflag, istat1, istat2*

Parameters

schdle	a unique, positive integer value representing the identifier for the frequency–based scheduler for which you wish to start or stop the attached real–time clock. You can obtain this value by making a call to **`fbsconfigure(3f)`** (see page 7-9 for an explanation of this subroutine) or **`fbsid(3f)`** (see page 7-14). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing its identifier, you can specify a value of –1.
runflag	an integer value indicating whether the real–time clock is to be started or stopped. A nonzero value indicates that the clock is to be started. A zero value indicates that the clock is to be stopped.
istat1	an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A nonzero value indicates that an error of a specific type has occurred. Refer to the **`fbsrunrtc(3f)`** man page for a listing of the nonzero values that may be returned and the types of errors that they represent. If *istat1* contains a value indicating that an error has occurred on an **`open`** or **`ioctl`** call, the error status of that call is returned in *istat2*.
istat2	the error status of an **`open`** or **`ioctl`** call. See the include file **`<errno.h>`** for a description of the error.

fbsschedself – Schedule a Process/Thread on a Frequency-Based Scheduler

The **fbsschedself** subroutine is invoked to schedule the calling process or thread on a frequency-based scheduler.

It is important to note that **fbsschedself** does not allow a process to set its scheduling policy and priority or its CPU bias. These tasks must be performed prior to invoking **fbsschedself**.

A process can set its scheduling policy and priority by using the **sched_setscheduler(2)** library routine; it can set its CPU bias by using the the **mpadvise(3)** library routine or **run(1)** command. Procedures for using these functions are explained in the *RedHawk Linux User's Guide*.

Note that you cannot use this subroutine to add **/idle** to a frequency-based scheduler.

To schedule the calling process on a frequency-based scheduler, it must have alter permission for the scheduler.

You must not change the scheduling policy or priority of a process while it is scheduled on a scheduler by using **sched_setscheduler** or other program interfaces that allow you to change scheduling policy and priority. The frequency-based scheduler is not aware of changes in scheduling policy and priority that are made by using these interfaces.

If you need to change the scheduling policy or priority of a single-threaded FBS-scheduled process, you may do so by using **schedpgmresched** to reschedule it (see page 7-62 for an explanation of this routine).

Call Statement

```
call fbsschedself(schdle, name, sched_buf, istat)
integer schdle, istat
character* (*) name
integer (*) sched_buf
```

Parameters

schdle	a unique, positive integer value representing the identifier for a frequency–based scheduler. You can obtain this value by making a call to **fbsconfigure(3f)** (see page 7-9 for an explanation of this subroutine) or **fbsid(3f)** (see page 7-14). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing the identifier, you can specify the value –1.
name	a standard path name or arbitrary content identifying the program associated with the calling process. A full or relative path name of up to 1023 characters can be specified.
sched_buf	an integer array that contains the scheduling parameters with which you wish to schedule the process. The information that is specified in this array is presented in Table 7-5.
istat	an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A nonzero value indicates that an error of a specific type has occurred. Refer to the **fbsschedself(3f)** man page for a listing of the nonzero values that may be returned and the types of errors that they represent.

Table 7-5. Contents of Array Elements: fbssetrtc

Element	Contents
sched_buf(1)	an integer value indicating the version of *sched_buf* that is being passed to **fbsschedself**. Specify the symbolic constant **FBSSCHED_BUF_V1,** which is defined in **<fbsched.h>** for this purpose.
sched_buf(2)	an integer value to be passed to a process that is scheduled on a frequency-based scheduler. This value can be retrieved by the FBS–scheduled process through a call to **rtparm** (see page 7-43 for an explanation of this subroutine).
sched_buf(3)	an integer value indicating the frequency with which the calling process is to be wakened in each major frame. A period of one indicates that the calling process is to be wakened every minor cycle; a period of two indicates that it is to be wakened once every two minor cycles, a period of three once every three minor cycles, and so on. This value can range from one to the number of minor cycles that compose a frame on the specified scheduler as defined in a call to **fbsconfigure** (see page 7-9 for an explanation of this subroutine).
sched_buf(4)	an integer value indicating the first minor cycle in which the calling process is scheduled to be wakened in each frame. This value can range from zero to the total number of minor cycles per frame minus one. The total number of minor cycles per frame is specified in a call to **fbsconfigure** (see page 7-9 for an explanation of this routine).
sched_buf(5)	an integer value indicating whether or not the scheduler should be stopped in the event that the calling process causes a frame overrun. A nonzero value indicates that the scheduler will be stopped.
sched_buf(6)	an integer value that is returned by **fbsschedself** and is the unique frequency–based scheduler process identifier for the scheduled process

fbssetrtc – Set Real–Time Clock

This subroutine is invoked to establish the duration of a minor cycle by setting the count and the resolution values for a real–time clock.

Call Statement

`call fbssetrtc(`*schdle, count, resolution, istat1, istat2*`)`
`integer` *schdle, count, resolution, istat1, istat2*

Parameters

Parameters must be specified in the order indicated. They are described as follows.

schdle — a unique, positive integer value representing the identifier for the frequency–based scheduler to which a real–time clock has been attached. You can obtain this value by making a call to **`fbsconfigure(3f)`** (see page 7-9 for an explanation of this subroutine) or **`fbsid(3f)`** (see page 7-14). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing its identifier, you can specify a value of **`–1`**.

count — an integer value indicating the number of clock counts per minor cycle. This value can range from 2 to 65535.

resolution — an integer value indicating the duration in microseconds of one clock count. This value must be one of the following: 1, 10, 100, 1000, or 10000.

istat1 — an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A nonzero value indicates that an error of a specific type has occurred. Refer to the **`fbssetrtc(3f)`** man page for a listing of the nonzero values that may be returned and the types of errors that they represent. If *istat1* contains a value indicating that an error has occurred on an **`open`** or **`ioctl`** call, the error status of that call is returned in *istat2*.

istat2 — the error status of an **`open`** or **`ioctl`** call. See the include file **`<errno.h>`** for a description of the error.

NOTE: Although a *count* of 1 cannot be used, a timing interval equal to a *resolution* value can be set by using the next lower resolution value and a count of 10; e.g., a resolution of 1,000 and count value of 10 results in a timing interval of 10,000 microseconds.

fbswait – Wait on a Frequency-Based Scheduler

This subroutine enables a process that is scheduled on a frequency-based scheduler to sleep until its next scheduled minor cycle.

If the scheduled process does not call this library routine by its next scheduled minor cycle, either a soft overrun or a hard overrun is incurred.

A soft overrun occurs if the per-process count of consecutively missed scheduled minor cycles does not reach or exceed the per-process soft overrun limit. When a soft overrun occurs, the process returns immediately from the **fbswait** call instead of blocking to wait for the next scheduled minor cycle.

When the count of consecutively missed scheduled minor cycles reaches or exceeds the per-process overrun limits, a hard overrun occurs. In this case, the process is blocked in **fbswait** until the next scheduled minor cycle.

A process' consecutive soft overrun limit may be changed from the default value of 0 using **sched_pgm_set_soft_overrun_limit(3f)**. The hard overrun count, which can be read via **pmqrypgm(3f)**, and the soft overrun count, which can be read via **sched_pgm_soft_overrun_query(3f)**, indicate whether the process is actually running at its assigned frequency.

When the scheduled process is subject to a deadline and the scheduled process calls this service after its deadline time has passed, a deadline violation will be detected and the scheduler may be halted.

Call Statement

```
CALL fbswait(istat)
integer istat
```

Parameter

istat — an integer value indicating whether or not an error has occurred and whether the process has been wakened by the scheduler or by an **pgmtrigger(3f)** call from another process. Values that may be returned are as follows:

0	The process has been wakened normally
1	The process has been wakened as the result of a **pgmtrigger(3f)** call
2	The process did not sleep because the kernel detected a soft overrun and is allowing the process to attempt to recover from it
Other nonzero value	An error of a specific type has occurred. Refer to the **fbswait(3f)** man page for a listing of the nonzero values that may be returned and the types of errors that they represent.

nametopid – Return the Process ID for a Specified Process Name

The **nametopid** routine returns the process ID (pid) of the specified process.

Call Statement

```
call nametopid(name, key, cpu, pid, istat)
integer key, cpu, pid, istat
character* (*) name
```

Parameters

name — a standard path name identifying the process.

key — an integer value identifying a frequency–based scheduler; this value must be the same value that was specified for *key* when the scheduler was created by making a call to **fbsconfigure** (see page 7-9 for an explanation of this subroutine). It may also be set to **-1** to indicate that the process is not sheduled on a frequency-based scheduler.

cpu — an integer value indicating the processor(s) to be used in conjunction with the value of the *name* parameter to identify the program for which information is to be returned. Acceptable values and corresponding results are as follows:

0	The first process named by *name* that is currently running on the processor from which the call is made is specified
-1	The first process named by *name* that is currently running on any processor is specified
Bit mask	If (*cpu* & (1<<*i*)) is set (where *i* is an integer ranging from zero to 15 and representing a CPU) and it is the only bit set, the first process named by *name* that is running on CPU *i* is specified
	If (*cpu* & (1<<*i*)) is set and it is not the only bit set, the first process named by *name* that is currently running on any of the selected CPUs is rescheduled

pid — the process ID of the process specified in *name*

istat — an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A nonzero value indicates that an error of a specific type has occurred. Refer to the **nametopid(3f)** man page for a listing of the nonzero values that may be returned and the types of errors that they represent.

pgmquery – Query a Process on a Frequency-Based Scheduler

CAUTION

This interface is obsolete. It is recommended that you use **schedpgmqry(3f)** (see p. 7-59).

Information that is returned by this subroutine includes the following:

- The process' path name
- The CPU on which the process can execute
- The frequency–based scheduler process identifier
- The scheduling priority
- The period (the number of minor cycles indicating the frequency with which the process is wakened in each major frame)
- The starting base cycle (the first minor cycle in which the process is scheduled to be wakened in each major frame)
- The value of the "halt on overrun" flag

Call Statement

```
call pgmquery(schdle, name, cpu, slot, prior, period, cycle, ab, istat)
integer schdle, cpu, slot, prior, period, cycle, ab, istat
character* (*) name
```

Parameters

schdle — a unique, positive integer value representing the identifier for the frequency–based scheduler on which the process for which you wish to obtain scheduling information has been scheduled. You can obtain this value by making a call to **fbsconfigure(3f)** (see page 7-9 for an explanation of this subroutine) or **fbsid(3f)** (see page 7-14). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing its identifier, you can specify a value of – **1**.

name — a standard path name identifying the process for which information is to be returned. A full or relative path name of up to 1024 characters can be specified. If this variable contains blanks, you must provide the frequency–based scheduler process identifier in the *slot* parameter.

cpu — an integer value indicating the processor(s) to be used in conjunction with the value of the *name* parameter to identify the program for which information is to be returned. Acceptable values and corresponding results are as follows:

0	The first process named by *name* that is currently running on the processor from which the call is made is specified
-1	The first process named by *name* that is currently running on any processor is specified

Bit mask If (*cpu* & (1<<*i*)) is set (where *i* is an integer ranging from zero to 15 and representing a CPU) and it is the only bit set, the first process named by *name* that is running on CPU *i* is specified

If (*cpu* & (1<<*i*)) is set and it is not the only bit set, the first process named by *name* that is currently running on any of the selected CPUs is rescheduled

slot an integer value providing the unique frequency–based scheduler process identifier for the process for which information is to be returned. This value is obtained when you make a call to **pgmschedule** (see page 7-37 for an explanation of this subroutine). This value must be **– 1** if you wish to identify the program to be queried only by specifying *name* and *cpu*.

prior an integer value indicating the specified process' scheduling priority.

period an integer value indicating the frequency with which the specified program is to be wakened in each major frame.

cycle an integer value indicating the first minor cycle in which the specified process is scheduled to be wakened in each frame

ab an integer value indicating the value of the "halt on overrun" flag. A nonzero value indicates that the flag is set. A value of zero indicates that the flag is not set.

istat an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A nonzero value indicates that an error of a specific type has occurred. Refer to the **pgmquery(3f)** man page for a listing of the nonzero values that may be returned and the types of errors that they represent.

pgmremove – Remove a Process from a Frequency-Based Scheduler

This subroutine is invoked to remove a process from a frequency–based scheduler. You can identify the process that you wish to remove by using one of the following methods:

- Specify the name of the process and the CPU on which it is scheduled.
- Specify the process' frequency–based scheduler process identifier (slot number).
- Specify the name of the process, the CPU on which it is scheduled, and its frequency–based scheduler process identifier.

NOTE

The only method that can be used to identify a process that has been scheduled multiple times on the same CPU is to specify its frequency–based scheduler process identifier.

Call Statement

```
call pgmremove(schdle, name, cpu, slot, ab, istat)
integer schdle, cpu, slot, ab, istat
character* (*) name
```

Parameters

schdle — a unique, positive integer value representing the identifier for the frequency–based scheduler on which the process is scheduled. You can obtain this value by making a call to **`fbsconfigure(3f)`** (see page 7-9 for an explanation of this subroutine) or **`fbsid(3f)`** (see page 7-14). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing its identifier, you can specify a value of **`–1`**.

name — a standard path name identifying the process to be removed from the specified scheduler. A full or relative path name of up to 1024 characters can be specified. If this variable contains blanks, you must provide the frequency–based scheduler process identifier in the *slot* parameter.

cpu — an integer value indicating the processor(s) to be used in conjunction with the value of the *name* parameter to identify the process to be removed from the specified scheduler. Acceptable values and corresponding results are as follows:

0	The first process named by *name* that is currently running on the processor from which the call is made is removed
-1	The first process named by *name* that is currently running on any processor is removed
Bit mask	If (*cpu* & (1<<*i*)) is set (where *i* is an integer ranging from zero to 15 and representing a

CPU) and it is the only bit set, the first process named by *name* that is running on CPU *i* is specified

If (*cpu* & (1<<*i*)) is set and it is not the only bit set, the first process named by *name* that is currently running on any of the selected CPUs is specified

slot — an integer value providing the unique frequency–based scheduler process identifier for the process to be removed from the specified scheduler. This value is obtained when you make a call to **schedpgmadd** (see page 7-55 for an explanation of this subroutine). This value must be **–1** if you choose to identify the program to be removed only by specifying *name* and *cpu*.

ab — an integer value indicating the manner in which the specified process is be removed from the specified scheduler. A positive value indicates that the process is to be removed from the scheduler but allowed to continue executing. A negative value indicates that the process is to be removed from the scheduler and terminated.

istat — an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A nonzero value indicates that an error of a specific type has occurred. Refer to the **pgmremove(3f)** man page for a listing of the nonzero values that may be returned and the types of errors that they represent.

pgmreschedule – Reschedule a Process

CAUTION

This interface is obsolete. It is maintained for compatibility with CX/UX, but its behavior with respect to specification of a process' scheduling priority has changed. If you have an existing application that uses this interface, it is recommended that you change your application to use **schedpgmresched(3f)** (see p. 7-62).

This subroutine is invoked to change the scheduling parameters for a process that is scheduled on a frequency–based scheduler. You may wish, for example, to change a program's priority or the frequency with which it is scheduled to run. You cannot, however, change the CPU on which it has been scheduled.

To change a process' priority, the following conditions must be met:

- The calling process must have the capabilities associated with the *fbscheduser* role (see Chapter 1).
- The effective user ID of the calling process must match the effective user ID of the target process (the process for which the scheduling policy and priority are being set), or the calling process must have the P_OWNER privilege.

You can call **pgmreschedule** to change the parameters without having called **pgmremove** to remove the process from the scheduler (see page 7-32) or **fbsintrpt** to stop the simulation (see page 7-17).

You can identify the process that you wish to reschedule by using one of the following methods:

- Specify the name of the process and the CPU on which it is scheduled.
- Specify the process' frequency–based scheduler process identifier (slot number).
- Specify the name of the process, the CPU on which it is scheduled, and its frequency–based scheduler process identifier.

NOTE

The only method that can be used to identify a process that has been scheduled multiple times on the same CPU is to specify its frequency–based scheduler process identifier.

Call Statement

```
call pgmreschedule(schdle, name, cpu, slot, prior, param, period, cycle, ab, istat)
integer schdle, cpu, slot, prior, param, period, cycle, ab, istat
character* (*) name
```

Parameters

schdle — a unique, positive integer value representing the identifier for the frequency–based scheduler on which the process is scheduled. You can obtain this value by making a call to **fbsconfigure(3f)** (see page 7-9 for an explanation of this subroutine) or **fbsid(3f)** (see page 7-14). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing its identifier, you can specify a value of – **1**.

name — a standard path name identifying the process to be rescheduled. A full or relative path name of up to 1024 characters can be specified. If this variable contains blanks, you must provide the frequency–based scheduler process identifier in the *slot* parameter.

cpu — an integer value indicating the processor(s) to be used in conjunction with the value of the *name* parameter to identify the process to be rescheduled. Acceptable values and corresponding results are as follows:

0	The first process named by *name* that is currently running on the processor from which the call is made is rescheduled
-1	The first process named by *name* that is currently running on any processor is rescheduled
Bit mask	If (*cpu* & (1<<*i*)) is set (where *i* is an integer ranging from zero to 15 and representing a CPU) and it is the only bit set, the first process named by *name* that is running on CPU *i* is rescheduled
	If (*cpu* & (1<<*i*)) is set and it is not the only bit set, the first process named by *name* that is currently running on any of the selected CPUs is rescheduled

slot — an integer value providing the unique frequency–based scheduler process identifier for the process to be rescheduled. This value is obtained when you make a call to **pgmschedule** (see page 7-37 for an explanation of this subroutine). This value must be – **1** if you wish to identify the program to be rescheduled only by specifying *name* and *cpu*.

prior — an integer value indicating the specified process' scheduling priority. A process that has been scheduled using **pgmschedule** (see p. 7-37 for an explanation of this subroutine) is scheduled under the POSIX **SCHED_RR** scheduling policy. The value specified must lie in the range of priorities associated with this policy. You can obtain the allowable range of priorities by invoking the **run(1)** command from the shell and not specifying any options or arguments (see the corresponding man page for an explanation of this command). Higher numerical values correspond to more favorable scheduling priorities. For complete information on

scheduling policies and priorities, refer to the *RedHawk Linux User's Guide*.

param — an integer value to be passed to a process that is scheduled on a frequency–based scheduler.

period — an integer value indicating the frequency with which the specified program is to be wakened in each major frame. A period of one indicates that the specified program is to be wakened every minor cycle; two indicates that it is to be wakened once every two minor cycles, three once every three minor cycles, and so on. This value can range from one to the number of minor cycles that compose a frame on the specified scheduler as defined in a call to **fbsconfigure** (see page 7-9).

cycle — an integer value indicating the first minor cycle in which the specified process is scheduled to be wakened in each frame. This value can range from zero to the total number of minor cycles per frame minus one. The total number of minor cycles per frame is specified in a call to **fbsconfigure** (see page 7-9 for an explanation of this subroutine).

ab — an integer value indicating whether or not the scheduler should be stopped in the event that the specified process causes a frame overrun. A nonzero value indicates that the scheduler will be stopped.

istat — an integer value indicating whether or not an error has occurred. Zero indicates that no error has occurred. A nonzero value indicates that an error of a specific type has occurred. Refer to the **pgmreschedule(3f)** man page for a listing of the nonzero values that may be returned and the types of errors that they represent.

pgmschedule – Schedule a Process on a Frequency-Based Scheduler

CAUTION

This interface is obsolete. It is maintained for compatibility with CX/UX, but its behavior with respect to specification of a process' scheduling priority has changed. If you have an existing application that uses this interface, it is recommended that you change your application to use **schedpgmadd(3f)** (see p. 7-55).

This subroutine is invoked to create a new process and schedule it on a frequency–based scheduler. When a process is scheduled using this subroutine, it is scheduled under the POSIX **SCHED_RR** scheduling policy (for complete information on scheduling policies and priorities, refer to the *RedHawk Linux User's Guide*).

If you wish to set the process' scheduling priority, the following conditions must be met:

- The calling process must have the capabilities associated with the *fbscheduser* role (see Chapter 1).
- The effective user ID of the calling process must match the effective user ID of the target process (the process for which the scheduling policy and priority are being set), or the calling process must have the P_OWNER privilege.

If you wish to modify the process' CPU bias when you invoke this subroutine, the real or effective user ID of the calling process must match the real or saved user ID of the process for which the CPU assignment is being changed.

Call Statement

```
call pgmschedule(schdle, name, prior, param, period, cycle, ab, cpu, slot, istat)
integer schdle, prior, param, period, cycle, ab, cpu, slot, istat
character* (*) name
```

Parameters

schdle — a unique, positive integer value representing the identifier for a frequency–based scheduler. You can obtain this value by making a call to **fbsconfigure(3f)** (see page 7-9 for an explanation of this subroutine) or **fbsid(3f)** (see page 7-14). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing the identifier, you can specify the value **–1**.

name — a standard path name identifying the program to be scheduled on the scheduler. A full or relative path name of up to 1024 characters can be specified.

prior — an integer value indicating the specified process' scheduling priority. A process that is scheduled using **pgmschedule** is scheduled under the POSIX **SCHED_RR** scheduling policy. The value specified must lie in the range of priorities associated with this policy. You can obtain the allowable range of priorities by invoking the **run(1)** command from the shell and not specifying any options or arguments (see the

corresponding man page for an explanation of this command). Higher numerical values correspond to more favorable scheduling priorities.

For complete information on scheduling policies and priorities, refer to the *RedHawk Linux User's Guide*.

param — an integer value to be passed to a process that is scheduled on a frequency–based scheduler. This value can be retrieved by the FBS–scheduled process through a call to **`rtparm`** (see page 7-43 for an explanation of this subroutine).

period — an integer value indicating the frequency with which the specified program is to be wakened in each major frame. A period of one indicates that the specified program is to be wakened every minor cycle; a period of two indicates that it is to be wakened once every two minor cycles, a period of three once every three minor cycles, and so on. This value can range from one to the number of minor cycles that compose a frame on the specified scheduler as defined in a call to **`fbsconfigure`** (see page 7-9).

cycle — an integer value indicating the first minor cycle in which the specified program is scheduled to be wakened in each frame. This value can range from zero to the total number of minor cycles per frame minus one. (The total number of minor cycles per frame is specified in a call to **`fbsconfigure`**. See page 7-9 for an explanation of this subroutine.)

ab — refers to a flag that contains an integer value indicating whether or not the scheduler should be stopped in the event that the specified program causes a frame overrun. A nonzero value indicates that the scheduler will be stopped.

cpu — refers to a mask that identifies the processors on which the specified program can be scheduled to run. Acceptable values and corresponding results are as follows:

0	The program specified by *name* can be scheduled on the processor from which the call is made
-1	The program specified by *name* can be scheduled on any processor
Bit mask	If (*cpu* & (1<<*i*)) is set (where *i* is an integer ranging from zero to 15 and representing a CPU) the program specified by *name* can be scheduled on CPU *i*

slot — refers to a variable to which **`pgmschedule`** will return an integer value that is the unique frequency–based scheduler process identifier for the scheduled process.

istat refers to a variable to which **pgmschedule** will return an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A nonzero value indicates that an error of a specific type has occurred. Refer to the **pgmschedule(3f)** man page for a listing of the nonzero values that may be returned and the types of errors that they represent.

pgmstat – Query State of FBS–Scheduled Process

This subroutine is invoked to obtain information about the state of a particular process that has been scheduled on a frequency–based scheduler. The state of the process indicates whether it is in the **fbswait** sleep state or is in another state.

You can identify the process by using one of the following methods:

- Specify the name of the process and the CPU(s) on which it is scheduled.
- Specify the process' frequency–based scheduler process identifier (slot number).
- Specify the name of the process, the CPU on which it is scheduled, and its frequency–based scheduler process identifier.

NOTE

The only method that can be used to identify a process that has been scheduled multiple times on the same CPU is to specify its frequency–based scheduler process identifier.

Information that is returned includes the following:

- The process' path name
- A mask of the CPU(s) on which the process can run
- The frequency–based scheduler process identifier
- The current state of the process

Call Statement

```
call pgmstat(schdle, name, cpu, slot, state, istat)
integer schdle, cpu, slot, state, istat
character* (*) name
```

Parameters

schdle — a unique, positive integer value representing the identifier for the frequency–based scheduler on which the process for which you wish to obtain state information has been scheduled. You can obtain this value by making a call to **fbsconfigure(3f)** (see page 7-9 for an explanation of this subroutine) or **fbsid(3f)** (see page 7-14). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing its identifier, you can specify a value of **–1**.

name — a standard path name identifying the process for which state information is to be returned. A full or relative path name of up to 1024 characters can be specified. If this variable contains blanks, you must provide the frequency–based scheduler process identifier in the *slot* parameter. **Pgmstat** will return to this variable the path name of the specified FBS–scheduled process.

cpu an integer value indicating the processor(s) to be used in conjunction with the value of the *name* parameter to identify the program for which state information is to be returned. Acceptable values and corresponding results are as follows:

0	The first process named by *name* that is currently running on the processor from which the call is made is specified
-1	The first process named by *name* that is currently running on any processor is specified
Bit mask	If (*cpu* & (1<<*i*)) is set (where *i* is an integer ranging from zero to 15 and representing a CPU) and it is the only bit set, the first process named by *name* that is running on CPU *i* is specified
	If (*cpu* & (1<<*i*)) is set and it is not the only bit set, the first process named by *name* that is currently running on any of the selected CPUs is specified

Pgmstat will return to this variable the mask of the CPUs on which the specified process can run.

slot an integer value providing the unique frequency–based scheduler process identifier for the process for which status information is to be returned. This value is obtained when you make a call to **schedpgmadd** (see page 7-55 for an explanation of this subroutine). This value must be **– 1** if you wish to identify the program to be queried only by specifying *name* and *cpu*. **Pgmstat** will return to this variable the frequency–based scheduler process identifier for the specified process.

state an integer value indicating the current state of the specified process as defined in <**fbsched.h**.>

istat an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A nonzero value indicates that an error of a specific type has occurred. Refer to the **pgmstat(3f)** man page for a listing of the nonzero values that may be returned and the types of errors that they represent.

pgmtrigger – Trigger Process Waiting on FBS

This subroutine enables a process to wake a process that is in the **fbswait** sleep state. It is important to note that the calling process does not have to be scheduled on a frequency–based scheduler; the target process must be.

Call Statement

`call pgmtrigger(`*schdle, slot, tgrflg, istat*`)`
`integer` *schdle, slot, tgrflg, istat*

Parameters

schdle	a unique, positive integer value representing the identifier for a frequency–based scheduler on which the sleeping process is scheduled.
slot	an integer value providing the unique frequency–based scheduler process identifier for the sleeping process. This value is obtained when you make a call to **schedpgmadd** (see page 7-55 for an explanation of this subroutine).
tgrflg	an integer value indicating whether or not a context switch is to be forced on the processor on which the wakened process is executing. A nonzero value indicates that a context switch is to be forced.
istat	an integer value indicating whether or not an error has occurred. A value of zero indicates that the process is runnable. A nonzero value indicates that an error of a specific type has occurred. Refer to the **pgmtrigger(3f)** man page for a listing of the nonzero values that may be returned and the types of errors that they represent.

rtparm – Return Initiation Parameter

This subroutine enables a process that is scheduled on a frequency–based scheduler to obtain the value of a process initiation parameter that has been passed to it via a call to **schedpgmadd** (see page 7-55) or **schedpgmresched** (see 7-62).

Call Statement

```
call rtparm(param)
integer param
```

Parameter

param	the integer value passed to the process via a call to **schedpgmadd** or **schedpgmresched**.

sched_pgm_deadline_query – Query the Assigned Deadline for a Process

This routine queries the deadline parameters for a currently scheduled process or thread on a frequency-based scheduler.

It is important to note that this function will not detect new deadline violations. Use **sched_pgm_deadline_test(3f)** to trigger the detection of new deadline violations.

The process can be identified in one of the following ways:

- A slot only (if name is blank).
- A path name and processor id pair only (if *slot* is -1).
- Both a slot and the path name and processor id pair.

Call Statement

```
call sched_pgm_deadline_query (schdle, name, cpu, slot, ddln_halt, ddln_kind,
     ddln_origin, ddln_sec, ddln_nsec, violations, istat)
integer schdle, cpu, slot, ddln_halt, ddln_kind, ddln_origin, ddln_sec, ddln_nsec,
     violations, istat
character* (*) name
```

Parameters

schdle — obtained from an **fbconfigure(3f)** or **fbsid(3f)** library routine call or set to -1. -1 enables an FBS-scheduled process to reference the frequency-based scheduler on which it is scheduled without knowing the scheduler identifier.

name — path name that identifies the process (or thread in a multithreaded process). If the name is all blanks, then the *slot* field (frequency-based scheduler process identifier) must be given.

cpu — either a bit mask or set to 0 or -1. If a bit mask is specified, then those processors with (*cpu* & (1<<*i*)) set are requested. If *cpu* is 0, then the processor on which the call is made is requested. If *cpu* is -1, then all processors are requested. The first process named *name* that is currently running on one of the requested processors is returned.

slot — frequency-based scheduler process identifier for the process. If the slot number equals -1, then a name and processor id must be given.

ddln_halt — indicates whether the scheduler will be halted upon detection of a deadline violation. Its value may be one of the following:

0	DL_NOHALT	indicating that the scheduler should not be halted upon detection of a deadline violation.
1	DL_HALT	indicating that the scheduler should be halted upon detection of a deadline violation.

ddln_kind the type of deadline set for the process by a **sched_pgm_set_deadline(3f)** call. Its value affects the interpretation of the value of *ddln_sec* and *ddln_nsec*. *ddln_kind* may be one of the following:

0 DEADLINE_CLEAR indicating that no deadline is currently assigned to the specified process or thread

1 DEADLINE_WALL_TIME indicating that the values *ddln_sec* and *ddln_nsec* together specify the CLOCK_MONOTONIC deadline time value assigned to the process or thread.

ddln_origin denotes the point from which the deadline time is measured. It may be one of the following:

0 DL_CYCLE_RELATIVE indicating that the deadline time is measured from the beginning of the cycle in which the task is scheduled.

1 DL_TASK_RELATIVE indicating that the deadline time is measured from the time that the scheduled task exits **fbswait(3)** and begins execution.

ddln_sec
ddln_nsec together, these represent the maximum time, measured from the specified origin, until the process is expected to return to **fbswait(3f)**.

violations indicates the total number of deadline violations that have occurred for the specified process or task.

istat an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A nonzero value indicates that an error of a specific type has occurred. Refer to the man page for a listing of the nonzero values that may be returned and the types of errors that they represent.

sched_pgm_deadline_test – Test for the Presence of a Deadline Violation

This subroutine tests for the occurrence of a deadline violation by the currently scheduled process or thread on a frequency-based scheduler.

The process can be identified in one of the following ways:

- A slot only (if name is blank).
- A path name and processor id pair only (if slot is -1).
- Both a slot and the path name and processor id pair.

Call Statement

```
call sched_pgm_deadline_test (schdle, name, cpu, slot, violated, violations,
     ddln_kind, ddln_origin, remaining_sec, remaining_nsec, istat)
integer schdle, cpu, slot, violated, violations, ddln_kind, ddln_origin, remaining_sec,
     remaining_nsec, istat
character* (*) name
```

Parameters

schdle — obtained from an **fbconfigure(3f)** or **fbsid(3f)** library routine call or set to -1. -1 enables an FBS-scheduled process to reference the frequency-based scheduler on which it is scheduled without knowing the scheduler identifier.

name — path name that identifies the process. If the name is all blanks, then the slot field (frequency-based scheduler process identifier) must be given.

cpu — either a bit mask or set to 0 or -1. If a bit mask is specified, then those processors with (*cpu* & (1<<*i*)) set are requested. If cpu is 0, then the processor on which the call is made is requested. If cpu is -1, then all processors are requested. The first process named name that is currently running on one of the requested processors is returned.

slot — frequency-based scheduler process identifier for the process. If the slot number equals -1, then a name and processor id must be given.

violated — the occurrence of a deadline violation. It may be one of the following

0 NO_VIOLATION — indicating no deadline has occurred for the specified process or thread.

1 DEADLINE_VIOLATION indicating a violation has occurred.

violations — the number of deadline violations for the process since the scheduler was started.

ddln_kind — the type of deadline set for the process by a **sched_pgm_set_deadline(3f)** call. Its value affects the interpretation of the value of *remaining_sec* and *remaining_nsec*. *ddln_kind* may be one of the following:

0 DEADLINE_CLEAR indicating that no deadline is currently assigned to the specified process or thread

1 DEADLINE_WALL_TIME indicating that the values *remaining_sec* and *remaining_nsec* together specify the CLOCK_MONOTONIC time remaining until the expiration of the deadline.

ddln_origin describes the deadline origin set for the process by a **sched_pgm_set_deadline(3f)** call. Its value affects the interpretation of the value of *remaining_sec* and *remaining_nsec*. *ddln_origin* may be one of the following:

0 DL_CYCLE_RELATIVE indicating that the remaining time is measured from the beginning of a cycle.

1 DL_TASK_RELATIVE indicating that the remaining time is measured from the beginning of task execution.

remaining_sec
remaining_nsec together, these represent the remaining time that the process is expected to spend executing before returning to **fbswait(3f)**.

istat an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A nonzero value indicates that an error of a specific type has occurred. Refer to the man page for a listing of the nonzero values that may be returned and the types of errors that they represent.

sched_pgm_set_deadline – Set or Clear Deadline Time

This routine sets or clears the deadline time for a currently scheduled process or thread on a frequency-based scheduler.

The deadline time for a process indicates the maximum amount of time the process is expected to spend executing before returning to **fbswait(3f)**. If the deadline time is exceeded, a deadline violation is incurred by the process. The scheduler may optionally be halted upon detection of a deadline violation.

To set or clear a deadline, the calling process must have alter permission for the scheduler.

The process can be identified in one of the following ways:

- A slot only (if name is blank).
- A path name and processor id pair only (if *slot* is -1).
- Both a slot and the path name and processor id pair.

Call Statement

```
call sched_pgm_set_deadline (schdle, name, cpu, slot, ddln_halt, ddln_kind,
     ddln_origin, ddln_sec, ddln_nsec, istat)
integer schdle, cpu, slot, ddln_halt, ddln_kind, ddln_origin, ddln_sec, ddln_nsec, istat
character* (*) name
```

Parameters

schdle — obtained from an **fbconfigure(3f)** or **fbsid(3f)** library routine call or set to -1. -1 enables an FBS-scheduled process to reference the frequency-based scheduler on which it is scheduled without knowing the scheduler identifier.

name — path name that identifies the process (or thread in a multithreaded process). If the name is all blanks, then the *slot* field (frequency-based scheduler process identifier) must be given.

cpu — either a bit mask or set to 0 or -1. If a bit mask is specified, then those processors with (*cpu* & (1<<*i*)) set are requested. If *cpu* is 0, then the processor on which the call is made is requested. If *cpu* is -1, then all processors are requested. The first process named *name* that is currently running on one of the requested processors is returned.

slot — frequency-based scheduler process identifier for the process. If the slot number equals -1, then a name and processor id must be given.

ddln_halt — indicates whether the scheduler will be halted upon detection of a deadline violation. Its value may be one of the following:

0	DL_NOHALT	indicating that the scheduler should not be halted upon detection of a deadline violation.
1	DL_HALT	indicating that the scheduler should be

halted upon detection of a deadline violation.

ddln_kind the type of deadline to be set for the process. Its value affects the interpretation of the value of *ddln_sec* and *ddln_nsec*. *ddln_kind* must be one of the following:

0 DEADLINE_CLEAR indicating that any deadline currently assigned to the specified process is to be removed. The specified process is no longer subject to a deadline. This is the default.

1 DEADLINE_WALL_TIME indicating that the values *ddln_sec* and *ddln_nsec* together specify the CLOCK_MONOTONIC deadline time value assigned to the process or thread.

ddln_origin denotes the point from which the deadline time is measured. It may be one of the following:

0 DL_CYCLE_RELATIVE indicating that the deadline time is to be measured from the beginning of the cycle in which the task is scheduled. The measured time will include any delay before the process begins execution, including normal process dispatch latency, time spent servicing interrupts or time spent waiting for a higher priority task to yield the CPU.

1 DL_TASK_RELATIVE indicating that the deadline time is measured from the time that the scheduled task exits **`fbswait(3)`** and begins execution.

ddln_sec
ddln_nsec must be non-negative and together define the number of seconds and nanoseconds (as for a C struct timespec value) representing the deadline time. The deadline time is the maximum time, relative to the specified origin, until the specified process is expected to return to **`fbswait(3f)`**. By default, this value is zero seconds, zero nanoseconds; i.e., if the process never sets a deadline time, it is zero.

istat an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A nonzero value indicates that an error of a specific type has occurred. Refer to the man page for a listing of the nonzero values that may be returned and the types of errors that they represent.

sched_pgm_set_soft_overrun_limit – Set Soft Overrun Limit

This subroutine sets the consecutive soft overrun limit for a currently scheduled process or thread on a frequency-based scheduler.

To set the consecutive soft overrun limit, the calling process must have alter permission for the scheduler.

The process can be identified in one of the following ways:

- A slot only (if name is blank).
- A path name and processor id pair only (if slot is -1).
- Both a slot and the path name and processor id pair.

Call Statement

```
call sched_pgm_set_soft_overrun_limit (schdle, name, cpu, slot, soft_limit,
     istat)
integer schdle,cpu, slot, soft_limit, istat
character* (*) name
```

Parameters

schdle	obtained from an **fbconfigure(3f)** or **fbsid(3f)** library routine call or set to -1. -1 enables an FBS-scheduled process to reference the frequency-based scheduler on which it is scheduled without knowing the scheduler identifier.
name	path name that identifies the process. If the name is all blanks, then the *slot* field (frequency-based scheduler process identifier) must be given.
cpu	either a bit mask or set to 0 or -1. If a bit mask is specified, then those processors with (*cpu* & (1 << *i*)) set are requested. If cpu is 0, then the processor on which the call is made is requested. If cpu is -1, then all processors are requested. The first process named that is currently running on one of the requested processors has its soft overrun limit set.
slot	frequency-based scheduler process identifier for the process. If the slot number equals -1, then a name and processor ID must be given.
soft_limit	number of consecutive soft overruns allowed to occur before failure. soft_limit must be non-negative and must be less than INT_MAX. By default, this value is zero; i.e., if the process never sets a consecutive soft overrun limit, then it is zero.
istat	an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A nonzero value indicates that an error of a specific type has occurred. Refer to the man page for a listing of the nonzero values that may be returned and the types of errors that they represent.

sched_pgm_soft_overrun_query – Query Soft Overrun Processing

This subroutine queries the status of soft overrun processing for a currently scheduled process or thread on a frequency-based scheduler. The process can be identified in one of the following ways:

- A slot only (if name is blank).
- A path name and processor id pair only (if slot is -1).
- Both a slot and the path name and processor id pair.

Call Statement

```
call sched_pgm_soft_overrun_query (schdle, name, cpu, slot, soft_limit,
     soft_total, istat)
integer schdle, cpu, slot, soft_limit, soft_total, istat
character* (*) name
```

Parameters

schdle	obtained from an **fbconfigure(3f)** or **fbsid(3f)** library routine call or set to -1. -1 enables an FBS-scheduled process to reference the frequency-based scheduler on which it is scheduled without knowing the scheduler identifier.
name	path name that identifies the process. If the name is all blanks, then the slot field (frequency-based scheduler process identifier) must be given.
cpu	either a bit mask or set to 0 or -1. If a bit mask is specified, then those processors with (*cpu* & (1<<*i*)) set are requested. If cpu is 0, then the processor on which the call is made is requested. If cpu is -1, then all processors are requested. The first process named name that is currently running on one of the requested processors is returned.
slot	frequency-based scheduler process identifier for the process. If the slot number equals -1, then a name and processor id must be given.
soft_limit	number of consecutive soft overruns set by calling **sched_pgm_set_soft_overrun_limit(3f)**.
soft_total	total number of soft overruns incurred by the process.
istat	an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A nonzero value indicates that an error of a specific type has occurred. Refer to the man page for a listing of the nonzero values that may be returned and the types of errors that they represent.

schedfbsqry – Query Processes on a Frequency-Based Scheduler

The **schedfbsqry** subroutine is invoked to obtain information about processes that have been scheduled on a frequency–based scheduler. Information is returned for all processes scheduled on the user–specified processor(s). Information provided for each process includes the following:

- A mask of the CPU(s) on which the process can execute
- The frequency–based scheduler process identifier
- The scheduling policy under which the process has been scheduled
- The scheduling priority
- The period (the number of minor cycles indicating the frequency with which the process is wakened in each major frame)
- The starting base cycle (the first minor cycle in which the process is scheduled to be wakened in each major frame)
- The value of the "halt on overrun" flag

Call Statement

`CALL schedfbsqry(`*schdle, cpu, buf1size, buf1, maxsize, buf2size, buf2, istat*`)`
`integer` *schdle, cpu, buf1size, buf1(buf1size), maxsize, buf2size, istat*
`character* (*)` *buf2*

Parameters

schdle — a unique, positive integer value representing the identifier for the frequency–based scheduler for which you wish to obtain scheduling information. You can obtain this value by making a call to **fbsconfigure(3f)** (see page 7-9 for an explanation of this subroutine) or **fbsid(3f)** (see page 7-14). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing its identifier, you can specify a value of **– 1**.

cpu — an integer value indicating the processor(s) for which scheduling information is to be obtained. Acceptable values and corresponding results are as follows:

0	Scheduling information for processes executing on the processor from which the call is made is returned
-1	Scheduling information for all processes on the scheduler is returned
Bit mask	If (*cpu* & (1<<*i*)) is set (where *i* is an integer ranging from zero to 15 and representing a CPU), scheduling information for processes executing on CPU *i* is returned

buf1size — an integer value indicating the size in 32–bit words of the array represented by *buf1*. Because 9 words of information are returned for each process, it is recommended that this value be a multiple of 9.

buf1 — an array containing a series of 11 integer values for each process on the processor(s) specified with the *cpu* parameter. The number of processes for which these values are returned is bound by the value of the *buf1size* parameter. If, for example, the value of *buf1size* is 145, values for 16 processes will be returned. These values represent the scheduling information for the process(es). The type of information returned in each array element for a single process is presented in Table 7-6.

Table 7-6. Contents of Array Elements: schedfbsqry

Element	Contents
1	Byte offset of the process' path name in *buf2*
2	Length in bytes of the process' path name
3	Mask of the CPU(s) on which the process can execute
4	The process' frequency–based scheduler process identifier
5	The process' scheduling policy
6	The process' scheduling priority
7	The number of minor cycles indicating the frequency with which the process is to be wakened in each major frame (period)
8	The first minor cycle in which the process is scheduled to be wakened in each major frame (starting base cycle)
9	The value of the "halt on overrun" flag. A nonzero value indicates that the flag is set. A value of zero indicates that the flag is not set.

maxsize — an integer value indicating the maximum length of a path name to be returned in *buf2*

buf2size — an integer value indicating the size in bytes of the character string represented by *buf2*. To ensure that *buf2* is large enough to accommodate the names of all processes that you wish to query, you may find it helpful to compute the number of bytes needed by multiplying the maximum number of processes allowed on the scheduler (see the information on **`fbsconfigure`** presented on page 7-9) by 32.

buf2 — refers to a variable to which **`schedfbsqry`** will return the path names for each process on the processor(s) specified with the *cpu* parameter. Path names are returned as a series of strings. The length of each string is less than or equal to the value of *maxsize*. Where *maxsize* is not large enough to accommodate a full path name, the concluding component names are returned. The number of path names returned is bound by the value of the *buf2size* parameter.

istat	refers to a variable to which **schedfbsqry** will return an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A nonzero value indicates that an error of a specific type has occurred. Refer to the **schedfbsqry(3f)** man page for a listing of the nonzero values that may be returned and the types of errors that they represent.

schedpgmadd – Schedule a Process on a Frequency-Based Scheduler

The **schedpgmadd** subroutine is invoked to create a new process and schedule it on a frequency–based scheduler. To include arguments, see **schedpgmadd_args** on page 7-57. It is important to note that to use this subroutine, the calling process must have the capabilities associated with the *fbscheduser* role (see Chapter 1).

Call Statement

```
call schedpgmadd(schdle, name, cid, prior, param, period, cycle, ab, cpu, slot, istat)
integer schdle, cid, prior, param, period, cycle, ab, cpu, slot, istat
character* (*) name
```

Parameters

schdle — a unique, positive integer value representing the identifier for a frequency–based scheduler. You can obtain this value by making a call to **fbsconfigure(3f)** (see page 7-9 for an explanation of this subroutine) or **fbsid(3f)** (see page 7-14). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing the identifier, you can specify the value **– 1**.

name — a standard path name identifying the program to be scheduled on the scheduler. A full or relative path name of up to 1024 characters can be specified.

cid — an integer value indicating the POSIX scheduling policy under which the specified process is to be scheduled: SCHED_FIFO, SCHED_RR or SCHED_OTHER. Scheduling policies are defined in the file **<sched.h>**.

prior — an integer value indicating the scheduling priority of the specified program. The range of acceptable priority values is governed by the scheduling policy specified.

You can determine the allowable range of priorities associated with each policy (SCHED_FIFO, SCHED_RR, or SCHED_OTHER) by invoking the **run(1)** command from the shell and not specifying any options or arguments (see the corresponding man page for an explanation of this command). Higher numerical values correspond to more favorable priorities.

For complete information on scheduling policies and priorities, refer to the *RedHawk Linux User's Guide*.

param — an integer value to be passed to a process that is scheduled on a frequency–based scheduler. This value can be retrieved by the FBS–scheduled process through a call to **rtparm** (see page 7-43 for an explanation of this subroutine).

period — an integer value indicating the frequency with which the specified program is to be wakened in each major frame. A period of one indicates that the specified program is to be wakened every minor cycle; a period of two indicates that it is to be wakened once every two minor cycles, a period of three

once every three minor cycles, and so on. This value can range from one to the number of minor cycles that compose a frame on the specified scheduler as defined in a call to **fbsconfigure** (see page 7-9).

cycle — an integer value indicating the first minor cycle in which the specified program is scheduled to be wakened in each frame. This value can range from zero to the total number of minor cycles per frame minus one. (The total number of minor cycles per frame is specified in a call to **fbsconfigure**. See page 7-9 for an explanation of this subroutine).

ab — a flag that contains an integer value indicating whether or not the scheduler should be stopped in the event that the specified program causes a frame overrun. A nonzero value indicates that the scheduler will be stopped.

cpu — a mask that identifies the processors on which the specified program can be scheduled to run. Acceptable values and corresponding results are as follows:

0	The program specified by *name* can be scheduled on the processor from which the call is made
-1	The program specified by *name* can be scheduled on any processor
Bit mask	If (*cpu* & (1<<*i*)) is set (where *i* is an integer ranging from zero to 15 and representing a CPU) the program specified by *name* can be scheduled on CPU *i*

slot — an integer value that is the unique frequency–based scheduler process identifier for the scheduled process.

istat — an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A nonzero value indicates that an error of a specific type has occurred. Refer to the **schedpgmadd(3f)** man page for a listing of the nonzero values that may be returned and the types of errors that they represent.

schedpgmadd_args – Schedule a Process on a Frequency-Based Scheduler with Arguments

The **schedpgmadd_args** subroutine is invoked to create a new process with arguments and schedule it on a frequency–based scheduler. It is important to note that to use this subroutine, the calling process must have the capabilities associated with the *fbscheduser* role (see Chapter 1).

Call Statement

```
call schedpgmadd_args(schdle, name, arglist, arglen, argc, cid, prior, param,
period, cycle, ab, cpu, slot, istat)
integer schdle, cid, prior, param, period, cycle, ab, cpu, slot, istat
integer argc, arglen(0:argc-1)
character* (*) arglist(0:argc-1)
character* (*) name
```

Parameters

schdle — a unique, positive integer value representing the identifier for a frequency–based scheduler. You can obtain this value by making a call to **fbsconfigure(3f)** (see page 7-9 for an explanation of this subroutine) or **fbsid(3f)** (see page 7-14). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing the identifier, you can specify the value – **1**.

name — a standard path name identifying the program to be scheduled on the scheduler. A full or relative path name of up to 1024 characters can be specified.

cid — an integer value indicating the POSIX scheduling policy under which the specified process is to be scheduled: SCHED_FIFO, SCHED_RR or SCHED_OTHER. Scheduling policies are defined in the file **<sched.h>**.

prior — an integer value indicating the scheduling priority of the specified program. The range of acceptable priority values is governed by the scheduling policy specified.

You can determine the allowable range of priorities associated with each policy (SCHED_FIFO, SCHED_RR, or SCHED_OTHER) by invoking the **run(1)** command from the shell and not specifying any options or arguments (see the corresponding man page for an explanation of this command). Higher numerical values correspond to more favorable priorities.

For complete information on scheduling policies and priorities, refer to the *RedHawk Linux User's Guide*.

param — an integer value to be passed to a process that is scheduled on a frequency–based scheduler. This value can be retrieved by the FBS–scheduled process through a call to **rtparm** (see page 7-43 for an explanation of this subroutine).

period — an integer value indicating the frequency with which the specified program is to be wakened in each major frame. A period of one indicates that the specified program is to be wakened every minor cycle; a period of two indicates that it is to be wakened once every two minor cycles, a period of three once every three minor cycles, and so on. This value can range from one to the number of minor cycles that compose a frame on the specified scheduler as defined in a call to **fbsconfigure** (see page 7-9).

cycle — an integer value indicating the first minor cycle in which the specified program is scheduled to be wakened in each frame. This value can range from zero to the total number of minor cycles per frame minus one. (The total number of minor cycles per frame is specified in a call to **fbsconfigure**. See page 7-9 for an explanation of this subroutine).

ab — a flag that contains an integer value indicating whether or not the scheduler should be stopped in the event that the specified program causes a frame overrun. A nonzero value indicates that the scheduler will be stopped.

cpu — a mask that identifies the processors on which the specified program can be scheduled to run. Acceptable values and corresponding results are as follows:

0	The program specified by *name* can be scheduled on the processor from which the call is made
-1	The program specified by *name* can be scheduled on any processor
Bit mask	If (*cpu* & (1<<*i*)) is set (where *i* is an integer ranging from zero to 15 and representing a CPU) the program specified by *name* can be scheduled on CPU *i*

arglist, arglen, argc — the length of the *arglist* array is explicitly passsed in *argc*. *arglist* and *arglen* must be dimensioned at least as large as *argc*. While these arrays may be dimensioned greater than required, *argc* specifies the number of elements at the beginning of each array that are to be used by the routine. The string length of each element in *arglist* is passed in the corresponding element of the *arglen* array.

slot — an integer value that is the unique frequency–based scheduler process identifier for the scheduled process.

istat — an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A nonzero value indicates that an error of a specific type has occurred. Refer to the **schedpgmadd_args(3f)** man page for a listing of the nonzero values that may be returned and the types of errors that they represent.

schedpgmqry – Query a Process on a Frequency-Based Scheduler

The **schedpgmqry** subroutine is invoked to obtain information for a particular process that has been scheduled on a frequency–based scheduler. You can identify the process by using one of the following methods:

- Specify the name of the process and the CPU(s) on which it is scheduled.
- Specify the process' frequency–based scheduler process identifier (slot number).
- Specify the name of the process, the CPU on which it is scheduled, and its frequency–based scheduler process identifier.

NOTE

The only method that can be used to identify a process that has been scheduled multiple times on the same CPU is to specify its frequency–based scheduler process identifier.

Information that is returned includes the following:

- The process' path name
- The CPU on which the process can execute
- The frequency–based scheduler process identifier
- The scheduling policy
- The scheduling priority
- The period (the number of minor cycles indicating the frequency with which the process is wakened in each major frame)
- The starting base cycle (the first minor cycle in which the process is scheduled to be wakened in each major frame)
- The value of the "halt on overrun" flag

Call Statement

```
call schedpgmqry(schdle, name, cpu, slot, cid, prior, period, cycle, ab, istat)
integer schdle, cpu, slot, cid, prior, period, cycle, ab, istat
character* (*) name
```

Parameters

schdle — a unique, positive integer value representing the identifier for the frequency–based scheduler on which the process for which you wish to obtain scheduling information has been scheduled. You can obtain this value by making a call to **fbsconfigure(3f)** (see page 7-9 for an explanation of this subroutine) or **fbsid(3f)** (see page 7-14). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing its identifier, you can specify a value of **–1**.

name — a standard path name identifying the process for which information is to be returned. A full or relative path name of up to 1024 characters can be specified. If this variable contains blanks, you must provide the frequency–based scheduler process identifier in the *slot* parameter.

cpu — an integer value indicating the processor(s) to be used in conjunction with the value of the *name* parameter to identify the program for which information is to be returned. Acceptable values and corresponding results are as follows:

0	The first process named by *name* that is currently running on the processor from which the call is made is specified
-1	The first process named by *name* that is currently running on any processor is specified
Bit mask	If (*cpu* & (1<<*i*)) is set (where *i* is an integer ranging from zero to 15 and representing a CPU) and it is the only bit set, the first process named by *name* that is running on CPU *i* is specified
	If (*cpu* & (1<<*i*)) is set and it is not the only bit set, the first process named by *name* that is currently running on any of the selected CPUs is specified

slot — an integer value providing the unique frequency–based scheduler process identifier for the process for which information is to be returned. This value is obtained when you make a call to **schedpgmadd** (see page 7-55 for an explanation of this subroutine). This value must be **–1** if you wish to identify the program to be queried only by specifying *name* and *cpu*.

cid — an integer value indicating the scheduling policy under which the specified process has been scheduled

prior — an integer value indicating the specified process' scheduling priority

period — an integer value indicating the frequency with which the specified program is to be wakened in each major frame.

cycle	an integer value indicating the first minor cycle in which the specified process is scheduled to be wakened in each frame
ab	an integer value indicating the value of the “halt on overrun” flag. A nonzero value indicates that the flag is set. A value of zero indicates that the flag is not set.
istat	an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A nonzero value indicates that an error of a specific type has occurred. Refer to the **`schedpgmqry(3f)`** man page for a listing of the nonzero values that may be returned and the types of errors that they represent.

schedpgmresched – Reschedule a Process

The **schedpgmresched** subroutine is invoked to change the scheduling parameters for a process that is scheduled on a frequency–based scheduler. You may wish, for example, to change a program's scheduling policy or priority or the frequency with which it is scheduled to run. You cannot, however, change the CPU on which it has been scheduled.

If you wish to (1) change a process' scheduling policy to the SCHED_FIFO or the SCHED_RR policy, (2) change the priority of a process scheduled under the SCHED_FIFO or the SCHED_RR policy or (3) raise the priority of a process scheduled under the SCHED_OTHER policy above a per-process or process limit, the following conditions must be met:

- The calling process must have the capabilities associated with the *fbscheduser* role (see Chapter 1).
- The effective user ID of the calling process must match the effective user ID of the target process (the process for which the scheduling policy and priority are being set).

You can identify the process that you wish to reschedule by using one of the following methods:

- Specify the name of the process and the CPU on which it is scheduled.
- Specify the process' frequency–based scheduler process identifier (slot number).
- Specify the name of the process, the CPU on which it is scheduled, and its frequency–based scheduler process identifier.

NOTE

The only method that can be used to identify a process that has been scheduled multiple times on the same CPU is to specify its frequency–based scheduler process identifier.

Call Statement

```
call schedpgmresched(schdle, name, cpu, slot, cid, prior, param, period, cycle,
     ab, istat)
integer schdle,cpu, slot, cid, prior, param, period, cycle, ab, istat
character* (*) name
```

Parameters

schdle — a unique, positive integer value representing the identifier for the frequency–based scheduler on which the process is scheduled. You can obtain this value by making a call to **fbsconfigure(3f)** (see page 7-9 for an explanation of this subroutine) or **fbsid(3f)** (see page 7-14). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing its identifier, you can specify a value of **–1**.

name — a standard path name identifying the process to be rescheduled. A full or relative path name of up to 1024 characters can be specified. If this variable contains blanks,

you must provide the frequency–based scheduler process identifier in the *slot* parameter.

cpu — an integer value indicating the processor(s) to be used in conjunction with the value of the *name* parameter to identify the process to be rescheduled. Acceptable values and corresponding results are as follows:

0	The first process named by *name* that is currently running on the processor from which the call is made is rescheduled
-1	The first process named by *name* that is currently running on any processor is rescheduled
Bit mask	If (*cpu* & (1<<*i*)) is set (where *i* is an integer ranging from zero to 15 and representing a CPU) and it is the only bit set, the first process named by *name* that is running on CPU *i* is rescheduled
	If (*cpu* & (1<<*i*)) is set and it is not the only bit set, the first process named by *name* that is currently running on any of the selected CPUs is rescheduled

slot — an integer value providing the unique frequency–based scheduler process identifier for the process to be rescheduled. This value is obtained when you make a call to **schedpgmadd** (see page 7-55 for an explanation of this subroutine). This value must be –1 if you wish to identify the program to be rescheduled only by specifying *name* and *cpu*.

cid — an integer value indicating the scheduling policy under which the specified program is to be scheduled: SCHED_FIFO, SCHED_RR or SCHED_OTHER. Scheduling policies are defined in the file **<sched.h>**.

prior — an integer value indicating the scheduling priority of the specified program. The range of acceptable priority values is governed by the scheduling policy specified.

You can determine the allowable range of priorities associated with each policy (SCHED_FIFO, SCHED_RR, or SCHED_OTHER) by invoking the **run(1)** command from the shell and not specifying any options or arguments (see the corresponding man page for an explanation of this command). Higher numerical values correspond to more favorable priorities.For complete information on scheduling policies and priorities, refer to the *RedHawk Linux User's Guide*.

param — an integer value to be passed to a process that is scheduled on a frequency–based scheduler.

period — an integer value indicating the frequency with which the specified program is to be wakened in each major frame. A period of one indicates that the specified program is to be

wakened every minor cycle; a period of two indicates that it is to be wakened once every two minor cycles, a period of three once every three minor cycles, and so on. This value can range from one to the number of minor cycles that compose a frame on the specified scheduler as defined in a call to **fbsconfigure** (see page 7-9).

cycle — an integer value indicating the first minor cycle in which the specified process is scheduled to be wakened in each frame. This value can range from zero to the total number of minor cycles per frame minus one. The total number of minor cycles per frame is specified in a call to **fbsconfigure** (see page 7-9 for an explanation of this subroutine).

ab — a flag that contains an integer value indicating whether or not the scheduler should be stopped in the event that the specified process causes a frame overrun. A nonzero value indicates that the scheduler will be stopped.

istat — an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A nonzero value indicates that an error of a specific type has occurred. Refer to the **schedpgmresched(3f)** man page for a listing of the nonzero values that may be returned and the types of errors that they represent.

Performance Monitor Routines

The Performance Monitor routines provide access to the key features of the Performance Monitor. They enable you to perform such basic operations as:

- clearing performance monitor values for a process or processor
- starting and stopping performance monitoring for a process or processor
- obtaining performance monitor values for a process or processor

Routine Summary

Performance Monitor routines are summarized in Table 7-7. Complete information about each routine is provided under the section “Using Performance Monitor Routines.”

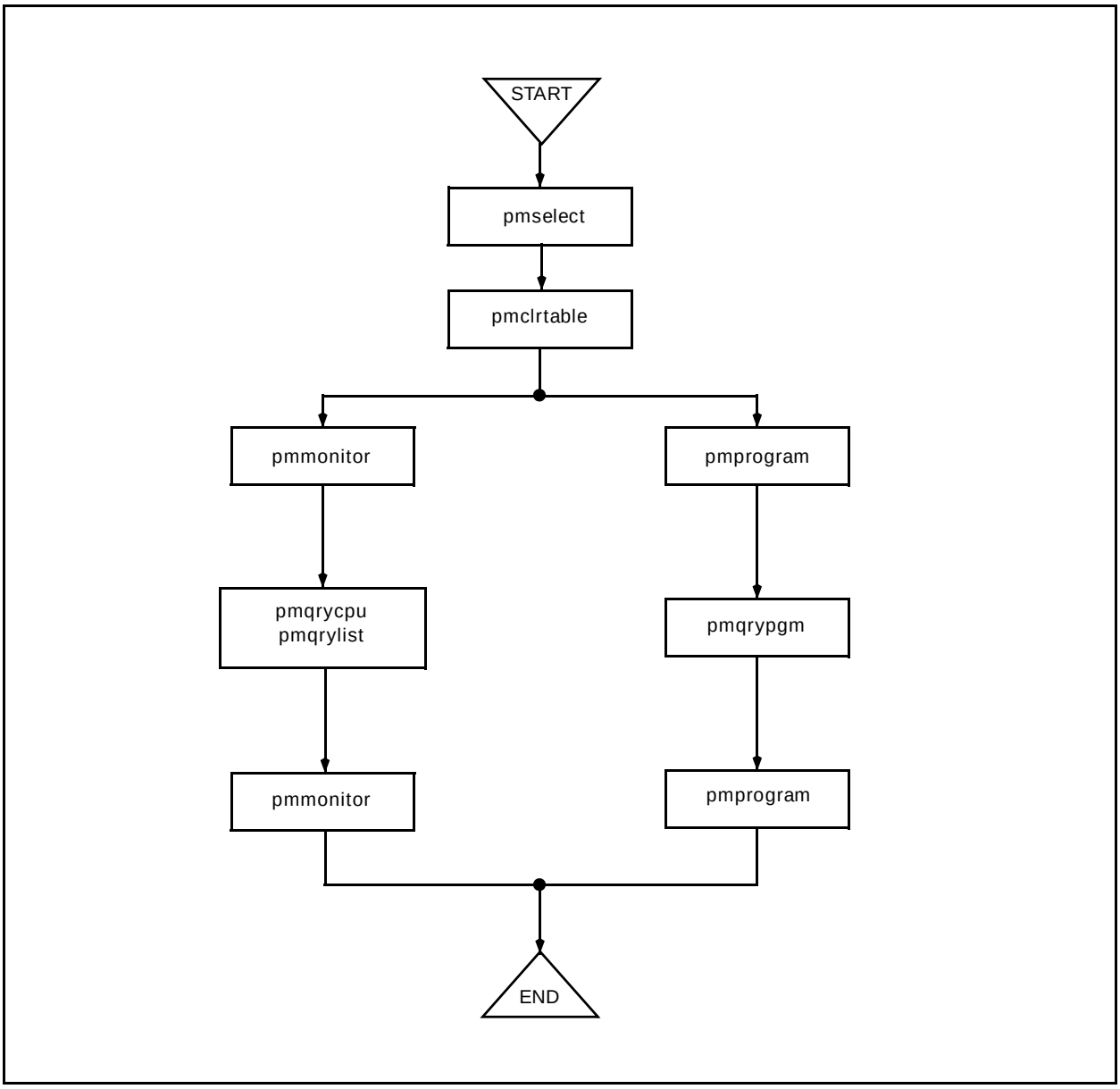
Table 7-7. Performance Monitor FORTRAN Library Routines

Routine	Page	Description
pmclrpgm	7-67	Clear values for a process
pmclrtable	7-69	Clear values for processor(s)
pmmonitor	7-70	Start/stop performance monitoring on processor(s)
pmprogram	7-71	Start/stop performance monitoring on a process
pmqrycpu	7-73	Query values for selected processor(s)
pmqrylist	7-75	Query values for a list of processes
pmqrypgm	7-77	Query values for a selected process
pmquerytimer	7-80	Query Performance Monitor mode
pmselect	7-81	Select Performance Monitor mode

FORTRAN Library Call Sequence

The approximate order in which you might call the Performance Monitor routines from an application program is illustrated in Figure 7-2.

Figure 7-2. FORTRAN Library Call Sequence: Performance Monitor



Using Performance Monitor Routines

In the sections that follow, the Performance Monitor routines contained in the **libF77rt** library are presented in alphabetical order.

pmclrpgm – Clear Values for a Process

This subroutine is invoked to clear performance monitor values for a particular process that has been scheduled on a frequency–based scheduler. You can identify the process by using one of the following methods:

- Specify the name of the process and the CPU(s) on which it is scheduled.
- Specify the process' frequency–based scheduler process identifier (slot number).
- Specify the name of the process, the CPU on which it is scheduled, and its frequency–based scheduler process identifier.

NOTE

This subroutine will clear the process' total soft overrun count.

Call Statement

```
call pmclrpgm(schdle, name, cpu, slot, istat)
integer schdle, cpu, slot, istat
character* (*) name
```

Parameters

schdle — a unique, positive integer value representing the identifier for the frequency–based scheduler on which the process is scheduled. You can obtain this value by making a call to **fbsconfigure(3f)** (see page 7-9 for an explanation of this subroutine) or **fbsid(3f)** (see page 7-14). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing its identifier, you can specify a value of – **1**.

name — a standard path name identifying the process for which values are to be cleared. A full or relative path name of up to 1024 characters can be specified. If this variable is filled with blanks, you must provide the frequency–based scheduler process identifier in the *slot* parameter.

cpu — an integer value indicating the processor(s) to be used in conjunction with the value of the *name* parameter to identify the process for which values are to be cleared. Acceptable values and corresponding results are as follows:

0	The first process named by *name* that is currently running on the processor from which the call is made is specified
-1	The first process named by *name* that is currently running on any processor is specified

Bit mask If (*cpu* & (1<<*i*)) is set (where *i* is an integer ranging from zero to 15 and representing a CPU) and it is the only bit set, the first process named by *name* that is running on CPU i is specified

If (*cpu* & (1<<*i*)) is set and it is not the only bit set, the first process named by *name* that is currently running on any of the selected CPUs is specified

slot an integer value providing the unique frequency–based scheduler process identifier for the process for which values are to be cleared. This value is obtained when you make a call to **schedpgmadd** (see page 7-55 for an explanation of this subroutine). This value must be **– 1** if you wish to identify the process only by specifying *name* and *cpu*.

istat an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A nonzero value indicates that an error of a specific type has occurred. Refer to the **pmclrpgm(3f)** man page for a listing of the nonzero values that may be returned and the types of errors that they represent.

pmclrtable – Clear Values for Processor(s)

This subroutine is invoked to clear performance monitor values for FBS–scheduled processes on one or more specified processors on a selected scheduler.

NOTE

This subroutine will clear the total soft overrun count for all related processes.

Call Statement

`call pmclrtable(`*schdle, cpucount, cpulist, istat*`)`
`integer` *schdle, cpucount, cpulist(cpucount), istat*

Parameters

schdle — a unique, positive integer value representing the identifier for the frequency–based scheduler on which the processes are scheduled. You can obtain this value by making a call to **`fbsconfigure(3f)`** (see page 7-9 for an explanation of this subroutine) or **`fbsid(3f)`** (see page 7-14). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing its identifier, you can specify a value of **`–1`**.

cpucount — an integer value indicating the number of elements contained in the array represented by *cpulist*.

cpulist — an array that consists of the number of elements specified by the *cpucount* parameter and contains one or more integer values indicating the processor or processors for which performance monitor values are to be cleared. Acceptable values and corresponding results are as follows:

0	Performance monitor values for FBS–scheduled processes executing on the processor from which the call is made are cleared
-1	Performance monitor values for all processes on the scheduler
Bit mask	If (*cpu* & (1<<*i*)) is set (where *i* is an integer ranging from zero to 15 and representing a CPU), performance monitor values for processes executing on CPU *i* are cleared

istat — an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A nonzero value indicates that an error of a specific type has occurred. Refer to the **`pmclrtable(3f)`** man page for a listing of the nonzero values that may be returned and the types of errors that they represent.

pmmonitor – Start/Stop Performance Monitoring on Processor(s)

This subroutine is invoked to start or stop performance monitoring for FBS–scheduled processes on one or more specified processors on a selected scheduler.

Call Statement

`call pmmonitor(`*schdle, pmflag, cpucount, cpulist, istat*`)`
`integer` *schdle, pmflag, cpucount, cpulist(cpucount), istat*

Parameters

schdle — a unique, positive integer value representing the identifier for the frequency–based scheduler on which the processes are scheduled. You can obtain this value by making a call to **`fbsconfigure(3f)`** (see page 7-9 for an explanation of this subroutine) or **`fbsid(3f)`** (see page 7-14). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing its identifier, you can specify a value of **`–1`**.

pmflag — an integer value indicating whether performance monitoring is to be started or stopped. A nonzero value indicates that performance monitoring is to be started. A zero value indicates that performance monitoring is to be stopped.

cpucount — an integer value indicating the number of elements in the array represented by *cpulist*.

cpulist — an array that consists of the number of elements specified by the *cpucount* parameter and contains one or more integer values indicating the processor or processors for which performance monitoring is to be started or stopped. Acceptable values and corresponding results are as follows:

0	Performance monitoring for FBS–scheduled processes executing on the processor from which the call is made is started or stopped
-1	Performance monitoring for all processes on the scheduler is started or stopped
Bit mask	If (*cpu* & (1<<*i*)) is set (where *i* is an integer ranging from zero to 15 and representing a CPU), performance monitoring for processes executing on CPU *i* is started or stopped

istat — an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A nonzero value indicates that an error of a specific type has occurred. Refer to the **`pmmonitor(3f)`** man page for a listing of the nonzero values that may be returned and the types of errors that they represent.

pmprogram – Start/Stop Performance Monitoring on a Process

This subroutine is invoked to start or stop performance monitoring for a particular process that has been scheduled on a frequency–based scheduler. You can identify the process by using one of the following methods:

- Specify the name of the process and the CPU(s) on which it is scheduled.
- Specify the process' frequency–based scheduler process identifier (slot number).
- Specify the name of the process, the CPU(s) on which it is scheduled, and its frequency–based scheduler process identifier.

Call Statement

`call pmprogram(`*schdle, name, cpu, slot, pmflag, istat*`)`
`integer` *schdle, cpu, slot, pmflag, istat*
`character* (*)` *name*

Parameters

schdle — a unique, positive integer value representing the identifier for the frequency–based scheduler on which the process is scheduled. You can obtain this value by making a call to **`fbsconfigure(3f)`** (see page 7-9 for an explanation of this subroutine) or **`fbsid(3f)`** (see page 7-14). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing its identifier, you can specify a value of **`–1`**.

name — a standard path name identifying the process for which performance monitoring is to be started or stopped. A full or relative path name of up to 1024 characters can be specified. If this variable is filled with blanks, you must provide the frequency–based scheduler process identifier in the *slot* parameter.

cpu — an integer value indicating the processor(s) to be used in conjunction with the value of the *name* parameter to identify the process for which performance monitoring is to be started or stopped. Acceptable values and corresponding results are as follows:

0	The first process named by name that is currently running on the processor from which the call is made is specified
-1	The first process named by name that is currently running on any processor is specified
Bit mask	If (cpu & (1<<*i*)) is set (where *i* is an integer ranging from zero to 15 and representing a CPU) and it is the only bit set, the first process named by name that is currently running on CPU *i* is specified
	If (cpu & (1<<*i*)) is set and it is not the only bit set, the first process named by name that is currently running on any of the selected CPUs is specified

slot	an integer value providing the unique frequency–based scheduler process identifier for the process for which performance monitoring is to be started or stopped. This value is obtained when you make a call to **schedpgmadd** (see page 7-55 for an explanation of this subroutine). This value must be **– 1** if you wish to identify the process only by specifying *name* and *cpu*.
pmflag	an integer value indicating whether performance monitoring is to be started or stopped. A nonzero value indicates that performance monitoring is to be started. A zero value indicates that performance monitoring is to be stopped.
istat	an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A nonzero value indicates that an error of a specific type has occurred. Refer to the **pmprogram(3f)** man page for a listing of the nonzero values that may be returned and the types of errors that they represent.

pmqrycpu – Query Values for Selected Processor(s)

This subroutine is invoked to obtain performance monitor values for FBS–scheduled processes on one or more specified processors on a selected scheduler.

Call Statement

`call pmqrycpu(`*schdle, cpu, bufsiz, buf, istat*`)`
`integer` *schdle, cpu, bufsiz, buf(bufsiz), istat*

Parameters

schdle — a unique, positive integer value representing the identifier for the frequency–based scheduler on which the processes are scheduled. You can obtain this value by making a call to **`fbsconfigure(3f)`** (see page 7-9 for an explanation of this subroutine) or **`fbsid(3f)`** (see page 7-14). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing its identifier, you can specify a value of **`–1`**.

cpu — an integer value indicating the processor(s) for which performance monitor values are to be obtained. Acceptable values and corresponding results are as follows:

0	Performance monitor values for FBS–scheduled processes executing on the processor from which the call is made are returned
-1	Performance monitor values for all processes on the scheduler are returned
Bit mask	If (cpu & (1<<*i*)) is set (where *i* is an integer ranging from zero to 15 and representing a CPU), performance monitor values for processes executing on CPU *i* are returned

bufsiz — an integer value indicating the size in 32–bit words of the array represented by *buf*. Because 16 words of information are returned for each process, it is recommended that this value be a multiple of 16.

buf — an array containing a series of 16 integer values for each FBS–scheduled process on the processor(s) specified with the *cpu* parameter. The number of processes for which these values are returned is bound by the value of the *bufsiz* parameter. If, for example, the value of *bufsiz* is 165, values for 10 processes will be returned. These values represent the performance monitoring information for the process(es). The type of information returned in each array element for a single process is presented in Table 7-8.

Table 7-8. Contents of Array Elements: pmqrycpu

Element	Contents
1	The process' frequency–based scheduler process identifier (slot number)
2	The amount of time that the process has spent running from the last time that it has been wakened by the scheduler until it has called **fbswait** (last time)
3	The number of times that the process has been wakened by the scheduler (total iterations, or cycles)
4	The number of seconds that the process has spent running in all cycles (total seconds). The total amount of time that the process has spent running is equal to the value of Element 4 plus the value of Element 5.
5	The additional number of microseconds that the process has spent running in all cycles (total microseconds). The total amount of time that the process has spent running is equal to the value of Element 4 plus the value of Element 5.
6	The number of hard frame overruns caused by the process
7	The least amount of time that the process has spent running in a cycle (minimum cycle time)
8	The number of the minor cycle in which the minimum cycle time has occurred (minimum cycle cycle)
9	The number of the major frame in which the minimum cycle time has occurred (minimum cycle frame)
10	The greatest amount of time that the process has spent running in a cycle (maximum cycle time)
11	The number of the minor cycle in which the maximum cycle time has occurred (maximum cycle cycle)
12	The number of the major frame in which the maximum cycle time has occurred (maximum cycle frame)
13	The least amount of time that the process has spent running during a major frame (minimum frame time)
14	The number of the major frame in which the minimum frame time has occurred (minimum frame frame)
15	The greatest amount of time that the process has spent running during a major frame (maximum frame time)
16	The number of the major frame in which the maximum frame time has occurred (maximum frame frame)

istat an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A nonzero value indicates that an error of a specific type has occurred. Refer to the **pmqrycpu(3f)** man page for a listing of the nonzero values that may be returned and the types of errors that they represent.

pmqrylist – Query Values for a List of Processes

This subroutine is invoked to obtain performance monitor values for a list of processes scheduled on a frequency–based scheduler.

Call Statement

`call pmqrylist(`*schdle, slotcount, slotlist, bufsiz, buf, istat*`)`
`integer` *schdle, slotcount, slotlist(slotcount), bufsiz, buf(bufsiz), istat*

Parameters

schdle — a unique, positive integer value representing the identifier for the frequency–based scheduler for which performance monitor values are requested. You can obtain this value by making a call to **`fbsconfigure(3f)`** (see page 7-9 for an explanation of this subroutine) or **`fbsid(3f)`** (see page 7-14). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing its identifier, you can specify a value of **`–1`**.

slotcount — an integer value indicating the number of frequency–based scheduler process identifiers contained in the array represented by *slotlist*.

slotlist — an array that consists of the number of elements specified by the *slotcount* parameter and contains one or more integer values indicating the frequency–based scheduler process identifiers for which performance monitor values are to be returned.

bufsiz — an integer value indicating the size in 32–bit words of the array represented by *buf*. Because 15 words of information are returned for each process, it is recommended that this value be a multiple of 15.

buf — an array containing a series of 15 integer values for each FBS–scheduled process. The number of processes for which these values are returned is bound by the value of the *bufsiz* parameter. If, for example, the value of *bufsiz* is 155, values for 10 processes will be returned. These values represent the performance monitoring information for the processes. The type of information returned in each array element for a single process is presented in Table 7-9.

istat — an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A nonzero value indicates that an error of a specific type has occurred. Refer to the **`pmqrylist(3f)`** man page for a listing of the nonzero values that may be returned and the types of errors that they represent.

Table 7-9. Contents of Array Elements: pmqrylist

Element	Contents
1	The amount of time that the process has spent running from the last time that it has been wakened by the scheduler until it has called **`fbswait`** (last time)
2	The number of times that the process has been wakened by the scheduler (total iterations, or cycles)
3	The number of seconds that the process has spent running in all cycles (total seconds). The total amount of time that the process has spent running is equal to the value of Element 3 plus the value of Element 4.
4	The additional number of microseconds that the process has spent running in all cycles (total microseconds). The total amount of time that the process has spent running is equal to the value of Element 3 plus the value of Element 4.
5	The number of hard frame overruns caused by the process
6	The least amount of time that the process has spent running in a cycle (minimum cycle time)
7	The number of the minor cycle in which the minimum cycle time has occurred (minimum cycle cycle)
8	The number of the major frame in which the minimum cycle time has occurred (minimum cycle frame)
9	The greatest amount of time that the process has spent running in a cycle (maximum cycle time)
10	The number of the minor cycle in which the maximum cycle time has occurred (maximum cycle cycle)
11	The number of the major frame in which the maximum cycle time has occurred (maximum cycle frame)
12	The least amount of time that the process has spent running during a major frame (minimum frame time)
13	The number of the major frame in which the minimum frame time has occurred (minimum frame frame)
14	The greatest amount of time that the process has spent running during a major frame (maximum frame time)
15	The number of the major frame in which the maximum frame time has occurred (maximum frame frame)

pmqrypgm – Query Values for a Selected Process

This subroutine is invoked to obtain performance monitor values for a particular process scheduled on a frequency–based scheduler.

Call Statement

```
call pmqrypgm(schdle, name, cpu, slot, last, iter, totsec, totusec, over, minc,
          minctc, minctf, maxc, maxctc, maxctf, minf, minftf, maxf, maxftf, istat)
integer  schdle, cpu, slot, last, iter, totsec, totusec, over, minc, minctc, minctf, maxc,
           maxctc, maxctf, minf, minftf, maxf, maxftf, istat
character* (*)  name
```

Parameters

schdle — a unique, positive integer value representing the identifier for the frequency–based scheduler for which performance monitor values are requested. You can obtain this value by making a call to **fbsconfigure(3f)** (see page 7-9 for an explanation of this subroutine) or **fbsid(3f)** (see page 7-14). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing its identifier, you can specify a value of **–1**.

name — a standard path name identifying the process for which performance monitoring values are to be returned. A full or relative path name of up to 1024 characters can be specified. If this variable is filled with blanks, you must provide the frequency–based scheduler process identifier in the *slot* parameter.

cpu — an integer value indicating the processor(s) to be used in conjunction with the value of the *name* parameter to identify the process for which performance monitoring values are to be returned. Acceptable values and corresponding results are as follows:

0	The first process named by *name* that is currently running on the processor from which the call is made is specified
-1	The first process named by *name* that is currently running on any processor is specified
Bit mask	If (*cpu* & (1<<*i*)) is set (where *i* is an integer ranging from zero to 15 and representing a CPU) and it is the only bit set, the first process named by *name* that is currently running on CPU *i* is specified
	If (*cpu* & (1<<*i*)) is set and it is not the only bit set, the first process named by *name* that is currently running on any of the selected CPUs is specified

slot — an integer value providing the unique frequency–based scheduler process identifier for the process for which performance monitoring values are to be returned. This value is obtained when you make a call to **schedpgmadd** (see page 7-55 for an explanation of this subroutine). This value must be

– **1** if you wish to identify the process only by specifying *name* and *cpu*.

last — an integer value indicating the amount of time that the process has spent running from the last time that it has been wakened by the scheduler until it has called **fbswait** (last time).

iter — an integer value indicating the number of times that the process has been wakened by the frequency–based scheduler since the last time that performance monitor values have been cleared and performance monitoring has been enabled (total iterations, or cycles).

totsec — an integer value indicating the number of seconds that the process has spent running in all cycles (total time in seconds). The total amount of time that the process has spent running is equal to the value of *totsec* plus *totusec*.

totusec — an integer value indicating the additional number of microseconds that the process has spent running in all cycles (total time in microseconds). The total amount of time that the process has spent running is equal to the value of *totsec* plus *totusec*.

over — an integer value indicating the number of times that the process has caused a hard frame overrun.

minc — an integer value indicating the least amount of time that the process has spent running in a cycle (minimum cycle time).

minctc — an integer value indicating the number of the minor cycle in which the minimum cycle time has occurred (minimum cycle cycle).

minctf — an integer value indicating the number of the major frame in which the minimum cycle time has occurred (minimum cycle frame).

maxc — an integer value indicating the greatest amount of time that the process has spent running in a cycle (maximum cycle time).

maxctc — an integer value indicating the number of the minor cycle in which the maximum cycle time has occurred (maximum cycle cycle).

maxctf — an integer value indicating the number of the major frame in which the maximum cycle time has occurred (maximum cycle frame).

minf — an integer value indicating least amount of time that the process has spent running in a major frame (minimum frame time).

minftf — an integer value indicating the number of the major frame in which the minimum frame time has occurred (minimum frame frame).

maxf	an integer value indicating the greatest amount of time that the process has spent running in a major frame (maximum frame time).
maxftf	an integer value indicating the number of the major frame in which the maximum frame time has occurred (maximum frame frame).
istat	an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A nonzero value indicates that an error of a specific type has occurred. Refer to the **pmqrypgm(3f)** man page for a listing of the nonzero values that may be returned and the types of errors that they represent.

pmquerytimer – Query Performance Monitor Mode

This subroutine is invoked to determine whether performance monitor timing values include or exclude time spent servicing interrupts.

Call Statement

```
call pmquerytimer(mode)
integer mode
```

Parameter

mode	a value indicating whether performance monitor timing values include or exclude time spent servicing interrupts. A value of one indicates that interrupt time is included. A value of zero indicates that interrupt time is excluded. A value of one or zero is returned if the call is successful; a negative value is returned to indicate that an error of a specific type has occurred. Refer to the **pmquerytimer(3f)** man page for a listing of the values that may be returned and the types of errors that they represent.

pmselect – Select Performance Monitor Mode

This subroutine is invoked to select the timing mode under which the performance monitor is to run. The timing mode can be set to include or exclude time spent servicing interrupts.

When interrupt time is included, a process' user and system times will total the elapsed time which accrues when the process is the currently running process, including all time spent servicing interrupts. The time spent servicing interrupts is added to the process' system time.

When excluding interrupt time, a process' user and system times will total the time which accrues when the process is the currently running process, excluding all time spent servicing interrupts.

Note that to set the timing mode, the calling process must have the capabilities associated with the *fbscheduser* role (see Chapter 1).

Call Statement

`call pmselect(`*mode, istat*`)`
`integer` *mode, istat*

Parameters

mode	an integer value indicating whether time spent servicing interrupts is to be included in or excluded from performance monitor timing values. A nonzero value indicates that interrupt time is to be included. A value of zero indicates that interrupt time is to be excluded.
istat	an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A nonzero value indicates that an error of a specific type has occurred. Refer to the **`pmselect(3f)`** man page for a listing of the nonzero values that may be returned and the types of errors that they represent.

A
Example rtcp Script

This appendix contains an example of a script that can be invoked at the system command prompt to execute **rtcp** commands.

This script illustrates using commands to:

- configure a scheduler and schedule programs on it (**cs**, **sp**)
- view information about the frequency-based scheduled processes (**vp**)
- view the scheduler configuration (**vs**)
- attach, set, and start a real-time clock (**ats**, **stc**, **rc**)
- clear performance monitor values (**cpm**)
- start performance monitoring and frequency-based scheduling (**pm**, **start**)
- view the minor cycle and major frame count (**vc**)
- stop performance monitoring; stop the real-time clock and the frequency-based scheduler (**pm**, **sc**, **stop**)
- detach the real-time clock (**dts**)
- remove the scheduler and all scheduled processes (**rms**)

```
rtcp cs -s37 -C10 -I664 -M5 -N10
rtcp sp -s37 -n ./program1 -c0 -bF -p19 -f4 -m0 -ohalt
rtcp sp -s37 -n ./program2 -c1 -bF -p19 -f2 -m1 -ohalt
rtcp vp -s37 -c*
rtcp vs -s37
rtcp ats -s37 -d/dev/rcim/rtc0
rtcp stc -s37 -O10000 -D1
rtcp rc -s37
rtcp cpm -s37 -c*
rtcp pm -s37 -c* -PON
rtcp start -s37
rtcp vc -s37
rtcp pm -s37 -c* -POFF
rtcp sc -s37
rtcp stop -s37
rtcp dts -s37
rtcp rms -s37 -a
```

B
rtcp Error Messages

This appendix contains descriptions of the errors that may be reported by the real-time command processor, **`rtcp`**.

Table B-1. rtcp Errors

Error	Description
-2	Interrupt device not specified
-3	Both EOC and interrupt device specified
-4	Process not specified
-5	Invalid rtcp command specified
-6	Invalid help command specified
-7	Invalid cpu (–c) parameter
-8	Invalid frequency (–f) parameter
-9	Invalid halt flag (–h|-o) parameter
-10	Invalid start cycle (–m) parameter
-11	Invalid priority (–p) parameter
-12	Invalid cycle count (–C) parameter
-13	Invalid clock tick duration (–D) parameter
-14	Invalid process per cycle (–M) parameter
-15	Invalid process per frequency-based scheduler (–N) parameter
-16	Invalid clock ticks per cycle (–O) parameter
-17	Invalid PM flag (–P) parameter
-18	Invalid parameter specified
-22	rj file not specified
-23	Invalid rj file specified
-24	Invalid pm viewing mode specified
-25	Invalid pm timing mode specified
-26	Unable to change timing mode to exclude interrupt time
-27	Invalid scheduling policy specified
-28	Exit rtcp
-29	Invalid soft overrun limit (-L) parameter
-30	Number of hosts (-H) exceeds limit
-31	Hostname(s) (-H) not specified
-32	Unable to obtain list of schedulers
-33	Invalid deadline (-T) parameter
-34	Invalid deadline origin (-r) parameter: cycle or task

C
Example C Interface

This appendix contains an example program that illustrates use of the C library interface to the Frequency-Based Scheduler and the Performance Monitor.

The program performs the following tasks:

- configures a scheduler
- schedules programs on the scheduler
- attaches, sets, and starts a real-time clock
- starts performance monitoring for each frequency-based scheduled process
- starts frequency-based scheduling
- obtains performance monitor values for the frequency-based scheduled processes
- monitors a processor's idle time

The example program begins on the next page.

schedule.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sched.h>
#include <sys/stat.h>
#include <fbsched.h>
#include <errno.h>
#define NUM_PROCS 4
#define START 1
#define STOP 0
#define PROGRAM1 "./prog1"
#define PROGRAM2 "./prog2"
#define IDLE0 "idle0"
#define IDLE1 "idle1"
#define CLOCK1 "/dev/rcim/rtc0"
void
cleanup(int fbs_id)
{
fbsrunrtc(fbs_id, 0); /* stop timing device */
fbsdetach(fbs_id); /* detach timing device */
fbsremove(fbs_id, -1); /* remove frequency-based scheduler */
}
int main()
{
struct fbsconfig_ds fbs_buf;
struct pgm2_ds sched_buf;
struct pmqry_ds pm_buf[NUM_PROCS];
struct fbsinfo_ds info_buf;
struct fbscycle_ds cycle_buf;
int idle0_fpid; /* fpid for idle on cpu 0 */
int idle1_fpid; /* fpid for idle on cpu 1 */
int pgm1_fpid; /* fpid for testprogram 1 */
int pgm2_fpid; /* fpid for testprogram 2 */
int cpu;
int istat;
int pmflg;
int intrflg;
int i;
int count;
int resolution;
char name[1024]; /* programâ€™s full or orelative path name */
char *pgmname[4]; /* programâ€™s name */
FILE *fp;
/* Open file to store performance information */
fp= fopen("pmresults", "w+");
if (fp == NULL)
printf("open failed errno = %d\n", errno);
/*
* CONFIGURE SCHEDULER
*/
fbs_buf.key = 37; /* scheduler key */
fbs_buf.cycles = 10; /* number of cycles per frame */
```

```
fbs_buf.progs = 5; /* max. number of programs per cycle */
fbs_buf.max = 10; /* max. number of programs allowed on the fbs */
fbs_buf.reset = -1; /* kill/remove processes currently scheduled */
/* owner/group read/write */
fbs_buf.configflg = 0664;
istat = fbsconfigure(&fbs_buf);
if (istat != 0) {
printf("could not configure scheduler: errno = %d\n" ,errno);
exit(1);
}
/*
* SCHEDULE test program1 PROGRAM1
*/
sched_buf.name_ptr = PROGRAM1;
sched_buf.cid = SCHED_FIFO; /* first-in-first out (FIFO) policy */
sched_buf.prior = 19;
sched_buf.param = 0; /* optional initiation parameter */
sched_buf.period = 2; /* time between wakeups */
sched_buf.cycle = 0; /* starting base cycle */
sched_buf.halt = 0; /* halt on overrun */
/* Set cpu mask to schedule testprogram1 on cpu 0 */
sched_buf.cpu = 1;
istat = sched_pgmadd(fbs_buf.fbs_id, &sched_buf);
printf("fbsid = %d\n", fbs_buf.fbs_id);
if (istat != 0) {
printf("could not schedule %s on cpu %d : errno = %d\n",
sched_buf.name_ptr, sched_buf.cpu>>1, errno);
cleanup(fbs_buf.fbs_id);
return(1);
}
pgm1_fpid = sched_buf.fpid;
printf("pgm1 fpid = %d\n", pgm1_fpid);
/*
* SCHEDULE test program2 PROGRAM2
*/
sched_buf.name_ptr = PROGRAM2;
sched_buf.prior = 19;
sched_buf.cid = SCHED_FIFO; /* first-in-first out (FIFO) policy */
sched_buf.param = 0; /* optional initiation parameter */
sched_buf.period = 2; /* time between wakeups */
sched_buf.cycle = 1; /* starting base cycle */
sched_buf.halt = 0; /* halt on overrun */
/* Set cpu mask to schedule testprogram2 on cpu 1 */
sched_buf.cpu = 2;
istat = sched_pgmadd(fbs_buf.fbs_id, &sched_buf);
printf("fbsid = %d\n", fbs_buf.fbs_id);
if (istat != 0) {
printf("could not schedule %s on cpu %d : errno = %d\n",
sched_buf.name_ptr, sched_buf.cpu>>1, errno);
cleanup(fbs_buf.fbs_id);
return(1);
}
pgm2_fpid = sched_buf.fpid;
printf("pgm2 fpid = %d\n", pgm2_fpid);
/*
* SCHEDULE IDLE ON CPU 0
*
```

```
* The only parameter required for /idle is the CPU.
*/
sched_buf.name_ptr = "/idle";
sched_buf.cid = SCHED_OTHER; /* first-in-first out (FIFO) policy */
sched_buf.prior = 0;
sched_buf.param = 0; /* optional initiation parameter */
sched_buf.period = 1; /* time between wakeups */
sched_buf.cycle = 0; /* starting base cycle */
sched_buf.halt = 0; /* halt on overrun */
/* Set cpu mask to schedule idle on cpu 0 */
sched_buf.cpu = 1; /* mask = 1<<cpu */
istat = sched_pgmadd(fbs_buf.fbs_id, &sched_buf);
printf("fbsid = %d\n", fbs_buf.fbs_id);
if (istat != 0) {
printf("could not schedule %s on cpu %d : errno = %d\n",
sched_buf.name_ptr, sched_buf.cpu, errno);
cleanup(fbs_buf.fbs_id);
return(1);
}
idle0_fpid = sched_buf.fpid;
printf("idle0 fpid = %d\n", idle0_fpid);
/*
* SCHEDULE IDLE ON CPU 1
*
* The only parameter required for /idle is the CPU.
*/
sched_buf.name_ptr = "/idle";
sched_buf.prior = 0;
sched_buf.cid = SCHED_OTHER; /* first-in-first out (FIFO) policy */
sched_buf.param = 0; /* optional initiation parameter */
sched_buf.period = 1; /* time between wakeups */
sched_buf.cycle = 0; /* starting base cycle */
sched_buf.halt = 0; /* halt on overrun */
/* Set cpu mask to schedule idle on cpu 1 */
sched_buf.cpu = 2; /* mask = 1<<cpu */
istat = sched_pgmadd(fbs_buf.fbs_id, &sched_buf);
if (istat != 0) {
printf("could not schedule %s on cpu %d : errno = %d\n",
sched_buf.name_ptr, sched_buf.cpu, errno);
cleanup(fbs_buf.fbs_id);
return(1);
}
idle1_fpid = sched_buf.fpid;
printf("idle1 fpid = %d\n", idle1_fpid);
/*
* ATTACH/SET REAL-TIME CLOCK
* Set the clock to interrupt every 10 msecs.
*/
count = 10000;
resolution = 1;
istat = fbsattach(fbs_buf.fbs_id, CLOCK1);
if (istat != 0) {
printf("could not attach timing source: errno = %d\n", errno);
cleanup(fbs_buf.fbs_id);
return(1);
}
```

```
istat = fbssetrtc(fbs_buf.fbs_id, count, resolution);
if (istat != 0) {
printf("could not set rtc: errno = %d\n", errno);
cleanup(fbs_buf.fbs_id);
return(1);
}
istat = fbsrunrtc(fbs_buf.fbs_id, START);
if (istat != 0) {
printf("could not start rtc: errno = %d\n", errno);
cleanup(fbs_buf.fbs_id);
return(1);
}
/*
* START PERFORMANCE MONITORING
*/
pmflg = 1;
/* zero out the "name" variable (fpid must be specified).
* Ulimately, the "name" variable can be used to store
* the full or relative path name of a test program.
*/
bzero(name, sizeof(name));
cpu = 0; /* not used if fpid is being used */
/* start performance monitoring for testprogram1 */
istat = pmprogram(fbs_buf.fbs_id, name, cpu, pgm1_fpid, pmflg);
if (istat != 0) {
printf("could not start pm for testprogram1 : errno = %d\n", errno);
cleanup(fbs_buf.fbs_id);
return(1);
}
/* start performance monitoring for testprogram2 */
istat = pmprogram(fbs_buf.fbs_id, name, cpu, pgm2_fpid, pmflg);
if (istat != 0) {
printf("coul dnot start pm for testprogram2 : errno = %d\n", errno);
cleanup(fbs_buf.fbs_id);
return(1);
}
/* start performance monitoring for idle on cpu 0 */
istat = pmprogram(fbs_buf.fbs_id, name, cpu, idle0_fpid, pmflg);
if (istat != 0) {
printf("could not start pm for idle0 : errno = %d\n", errno);
cleanup(fbs_buf.fbs_id);
return(1);
}
/* start performance monitoring for idle on cpu 1 */
istat = pmprogram(fbs_buf.fbs_id, name, cpu, idle1_fpid, pmflg);
if (istat != 0) {
printf("could not start pm for idle1 : errno = %d\n", errno);
cleanup(fbs_buf.fbs_id);
return(1);
}
/*
* START SCHEDULING
*/
intrflg = 1;
istat = fbsintrpt(fbs_buf.fbs_id, intrflg);
if (istat != 0) {
printf("could not start scheduler : errno = %d\n", errno);
```

```
cleanup(fbs_buf.fbs_id);
return(1);
}
/*
* QUERY PERFORMANCE MONITOR VALUES
* 1 second = 100 cycles 1 minute = 600 frames
* Query once per second for 1 minute
*/
pm_buf[0].fpid = pgm1_fpid; pgmname[0] = PROGRAM1;
pm_buf[1].fpid = pgm2_fpid; pgmname[1] = PROGRAM2;
pm_buf[2].fpid = idle0_fpid; pgmname[2] = IDLE0;
pm_buf[3].fpid = idle1_fpid; pgmname[3] = IDLE1;
sleep(1); /* sleep for a while */
printf("Please wait, performance information being gathered. \n");
cycle_buf.cframe = 0;
while ((cycle_buf.cframe < 600)) {
istat = fbscycle(fbs_buf.fbs_id, &cycle_buf);
if (istat != 0) {
printf("could not query fbscycle: errno = %d\n", errno);
cleanup(fbs_buf.fbs_id);
return(1);
}
istat = pmqrylist(fbs_buf.fbs_id, pm_buf, NUM_PROCS);
if (istat != 0) {
printf("could not query process: errno = %d\n", errno);
cleanup(fbs_buf.fbs_id);
return(1);
}
fprintf(fp,"****************************************\n");
fprintf(fp,"\tFRAME=%d, CYCLE=%d\n", cycle_buf.cframe,
cycle_buf.ccycle);
fprintf(fp,"****************************************\n\n");
/* Write performance information for each
* process into a file
*/
for (i = 0; i < NUM_PROCS; i++) {
fprintf(fp,"pgm=%s, fpid=%d\n", pgmname[i],
pm_buf[i].fpid);
fprintf(fp,"--------------------------------------\n");
fprintf(fp,"last cycle secs:nsecs = %d:%d\n",
pm_buf[i].lastcyc_tm.tv_sec,
pm_buf[i].lastcyc_tm.tv_nsec);
fprintf(fp,"total cycles = %d\n",pm_buf[i].tot_cycles);
fprintf(fp, "total cycles secs:nsecs = %d:%d\n",
pm_buf[i].tot_cycles_tm.tv_sec,
pm_buf[i].tot_cycles_tm.tv_nsec);
fprintf(fp, "overruns = %d\n",pm_buf[i].overruns);
fprintf(fp, "mincyc secs:nsecs = %d:%d\n",
pm_buf[i].mincyc_tm.tv_sec,
pm_buf[i].mincyc_tm.tv_nsec);
fprintf(fp, "mincyc_cycle = %d\n",
pm_buf[i].mincyc_cycle);
fprintf(fp, "mincyc_frame = %d\n",
pm_buf[i].mincyc_frame);
fprintf(fp, "maxcyc secs:nsecs = %d:%d\n",
pm_buf[i].maxcyc_tm.tv_sec,
```

```
pm_buf[i].maxcyc_tm.tv_nsec);
fprintf(fp, "maxcyc_cycle = %d\n",
pm_buf[i].maxcyc_cycle);
fprintf(fp, "maxcyc_frame = %d\n",
pm_buf[i].maxcyc_frame);
fprintf(fp, "minframe secs:nsecs = %d:%d\n",
pm_buf[i].minframe_tm.tv_sec,
pm_buf[i].minframe_tm.tv_nsec);
fprintf(fp, "minframe = %d\n", pm_buf[i].minframe);
fprintf(fp, "maxframe secs:nsecs = %d:%d\n",
pm_buf[i].maxframe_tm.tv_sec,
pm_buf[i].maxframe_tm.tv_nsec);
fprintf(fp, "maxframe = %d\n", pm_buf[i].maxframe);
fprintf(fp, "\n");
}
sleep(1); /* sleep for a while */
}
printf("Performance data has been gathered. \n");
if (istat != 0)
printf("istat = %d\n", istat);
/* Stop PM on CPUs 0 and 1 */
pmmonitor(fbs_buf.fbs_id, 0, 3);
/* Stop the clock */
istat = fbsrunrtc(fbs_buf.fbs_id, STOP);
if (istat != 0) {
printf("could not stop rtc: errno = %d\n", errno);
cleanup(fbs_buf.fbs_id);
return(1);
}
/* Detach from the fbs */
istat = fbsdetach(fbs_buf.fbs_id);
if (istat != 0) {
printf("could not dettach timing source: errno = %d\n", errno);
cleanup(fbs_buf.fbs_id);
return(1);
}
/* Remove the fbs */
istat = fbsremove(fbs_buf.fbs_id, -1);
if (istat != 0) {
printf("could not remove timing source: errno = %d\n", errno);
cleanup(fbs_buf.fbs_id);
return(1);
}
cleanup(fbs_buf.fbs_id);
return(0);
}
```

prog.c

```
#include <fbsched.h>
#include <time.h>

double fact(double x) {
    if (x == 1.0) return x;
    else return x * fact(x-1.0);
}

int main() {
    double factVal;
    struct timespec currTime;
// Get/Create a seed for random number creation
    clock_gettime(CLOCK_REALTIME, &currTime);
    srandom( (unsigned) currTime.tv_nsec);

    while(1){
        fbswait( );
        factVal = fact ( (double) ((random( ) % 10 ) + 1) );
    }
}
```

Makefile

```
cc = gcc
FLAGS = -g -lccur_rt -lccur_fbsched

all: schedule prog1 prog2

schedule: schedule.c
    $(CC) $(FLAGS) -o schedule schedule.c

prog1: prog.c
    $(CC) $(FLAGS) -o prog1 prog.c
    cp prog1 prog2

clean:
    rm -f prog1 prog2 schedule pmresults *.*~
```

Glossary

deadline violation

The condition that occurs when a frequency-based scheduled process does not finish its processing before the end of its deadline time. Note that detection of a deadline violation may cause the scheduler to be stopped.

end-of-cycle scheduling

A form of frequency-based scheduling in which scheduling is triggered when the last process scheduled to execute in the current minor cycle of the current major frame completes its processing.

frame overrun

The condition that occurs when a frequency-based scheduled process does not finish its processing before it is scheduled to run again. This applies to interrupt scheduling only, not end-of-cycle scheduling.

frequency

When applied to the frequency-based scheduler, indicates a time interval. This interval can be based on high-resolution clocks, an external interrupt source or completion of a cycle.

frequency-based scheduler

A high resolution task synchronization mechanism that enables processes to run at user-specified frequencies.

hard overrun

A type of frame overrun that is a catastrophic failure of the scheduled process.

idle time

Time during which the CPU is not busy.

iteration

One instance of a process being awakened by a frequency-based scheduler.

last time

A value returned by the performance monitor indicating the amount of time that a frequency-based scheduled process has spent running from the last time that it has been awakened by the scheduler until it has called **`fbswait(3)`**.

major frame

One pass through all of the minor cycles with which a frequency-based scheduler is configured. A major frame has associated with it a duration, which is obtained by multiplying the duration of a minor cycle by the number of minor cycles per major frame.

maximum cycle cycle

A value returned by the performance monitor indicating the number of the minor cycle in which the maximum cycle time has occurred.

maximum cycle frame

A value returned by the performance monitor indicating the number of the major frame in which the maximum cycle time has occurred.

maximum cycle time

A value returned by the performance monitor indicating the greatest amount of time in microseconds that a frequency-based scheduled process has spent running in a cycle.

maximum frame frame

A value returned by the performance monitor indicating the number of the major frame in which the maximum frame time has occurred.

maximum frame time

A value returned by the performance monitor indicating the greatest amount of time in microseconds that a frequency-based scheduled process has spent running during a major frame.

minimum cycle cycle

A value returned by the performance monitor indicating the number of the minor cycle in which the minimum cycle time has occurred.

minimum cycle frame

A value returned by the performance monitor indicating the number of the major frame in which the minimum cycle time has occurred.

minimum cycle time

A value returned by the performance monitor indicating the least amount of time in microseconds that a frequency-based scheduled process has spent running in a cycle.

minimum frame frame

A value returned by the performance monitor indicating the number of the major frame in which the minimum frame time has occurred.

minimum frame time

A value returned by the performance monitor indicating the least amount of time in microseconds that a frequency-based scheduled process has spent running during a major frame.

minor cycle

The smallest unit of frequency maintained by a frequency-based scheduler. A minor cycle has associated with it a duration, which is the time that elapses between interrupts generated by the timing source attached to the scheduler. If the timing source is a real-time clock, the minor cycle duration is defined by specifying the number of clock counts per minor cycle and the number of microseconds per clock count.

number of overruns

A value returned by the performance monitor indicating the number of times that a frequency-based scheduled process has caused a frame overrun.

performance monitor

A mechanism that makes it possible to monitor use of the CPU by processes that are scheduled on a frequency-based scheduler.

period

A frequency-based scheduler parameter that specifies how often a specified program is to be awakened in each major frame. A period of one indicates that the program is to be awakened every minor cycle; a period of two indicates that it is to be awakened once every two minor cycles; and so on.

privilege

A mechanism through which processes are allowed to perform sensitive operations or override system restrictions.

process dispatch latency

The time that elapses from the occurrence of an external event, which is signified by an interrupt, until the process waiting for that external event executes its first instruction in user mode.

scheduler key

A user-supplied numeric identifier for a frequency-based scheduler.

shielded processor

A CPU that is responsible for running high-priority tasks that are protected from the unpredictable processing associated with interrupts and system daemons.

soft overrun

A type of frame overrun that is a catastrophic failure only if the process has reached its limit on the number of soft overruns tolerated. Each scheduled process has a soft overrun limit, defaulting to 0.

Some soft overruns result from random, unpredictable, or external events unlikely to recur. Other soft overruns result from only minor frame overruns. Soft overruns give the scheduled process a chance to recover from a frame overrun and return to synchronization.

Soft overruns apply to interrupt scheduling only, not end-of-cycle scheduling.

spare time

Processor time that is composed of the following: (1) idle time, (2) CPU time of processes that are not scheduled on a frequency-based scheduler, and (3) CPU time of frequency-based scheduled processes for which performance monitoring has not been enabled.

starting base cycle

A frequency-based scheduler scheduling parameter that specifies the first minor cycle in which a frequency-based scheduled process is to be awakened in each major frame.

timing mode

The mode under which the performance monitor runs. It specifies whether time spent servicing interrupts is to be included in or excluded from performance monitor timing values.

total iterations

A value returned by the performance monitor indicating the number of times that a frquency-based scheduled process has been awakened by the scheduler.

total time

A value returned by the performance monitor indicating the total amount of time that a frequency-based scheduled process has spent running in all cycles.

unscheduled process

A process that is not awakened by the frequency-based scheduler and does not call **fbswait**; it is not scheduled to run at a certain frequency.

Index

D

E

F

P

Q

R

S

T

U

V

W