TimeSys Linux/RT™

Version 1.0

User's Manual

The Power of Real-Time in Pure Linux™

TimeSys Corporation
4516 Henry Street, Pittsburgh, Pennsylvania 15213 USA

Printed in the United States of America.

Table of Contents

Table of Figures

Acknowledgements

TimeSys would like to acknowledge the contributions of the following:

Members of the Real-Time and Multimedia Systems Laboratory at Carnegie Mellon University including Prof. Raj Rajkumar, Dr. Shui Oikawa, Akihiko Miyoshi, and Dionisio de Niz.

Members of the RED Linux project at the University of California at Irvine, including Prof. Kwei-Jay Lin and Yu-Chung Wang.

Members of the RTAI project at DIAPM in Italy. RTAI contributors include Paolo Mantegazza and his research group at DIAPM, Steve Papacharalambous of Zentropix, and Pierre Cloutier of Poseidon Controls, and

Of course, the Linux open-source community.

Chapter 1:

TimeSys Linux/RT: An Introduction

Welcome to TimeSys Linux/RT!

In this book, you'll find everything you need to get started with TimeSys Linux/RT, from instructions in how to set your system up and navigate it to information about the wide variety of applications available to you under Linux. But first this chapter seeks to answer two very important questions: What exactly is Linux, and what's so special about TimeSys Linux/RT?

1.1 Linux Explained

Linux is a free operating system modeled after UNIX — one of the oldest, most reliable, and most widely used operating systems in the world.

Linux offers you the benefits of full multitasking, TCP/IP networking, the X-Windows system, and almost any other capability you'd expect from a computer system. With Linux, the oldest personal computer can become a powerful workstation. Programmers everywhere, from the corporate world to academia, have found Linux to be the right operating system for their needs.

Yet the most distinctive aspect of Linux is the fact that it's free. This means, of course, that you can download Linux

installation software and continue to get upgrades without laying out any money whatsoever.

But it also means that the source code behind the system is available to everyone. Linux is an exemplar of the open source philosophy, which gives every user the potential to become a developer — to play around with the source code, to fix bugs, to extend what the system can do. Anyone with a knowledge of programming basics and a desire to get "down and dirty" can join the community of Linux programmers.

The ease of adding to Linux helped TimeSys in designing TimeSys Linux/RT. Instead of having to layer real-time software above Linux, TimeSys Linux/RT was able to give the kernel itself — the core of any Linux system — the capacity to handle real-time applications. Thus, TimeSys Linux/RT combines the functionality of a real-time operating system with the stability and reliability of Linux.

1.1.1 A Brief History of Linux

Once upon a time, a young University of Helsinki student named Linus Torvalds got it into his head to create a UNIX-like operating system. At this point (1991), UNIX — which made its debut at AT&T in 1969 and became popular through its support of multitasking and file sharing — had spawned a lookalike called Minix.

Torvalds had played around with Minix, but decided to take on the challenge of developing a completely new operating system — one that would hopefully be "a better Minix than Minix."

By October 1991, Torvalds had gotten a rudimentary OS working. While it was far from fully functional, he could run bash (the GNU Bourne Again Shell), gcc (the GNU C compiler), and a few other assorted programs.

On October 5, he announced the availability of Linux version 0.02 (0.01 had never been officially released) on `comp.os.minix`. In that post, he wrote:

> ```
> Do you pine for the nice days of Minix 1.1,
> when men were men and wrote their own device
> drivers? Are you without a nice project and
> just dying to cut your teeth on a OS you can
> try to modify for your needs? Are you finding
> it frustrating when everything works on
> Minix? No more all-nighters to get a nifty
> program working? Then this post might be just
> for you.
> As I mentioned a month ago, I'm working on a
> free version of a Minix-lookalike for AT-386
> computers. It has finally reached the stages
> where it's even usable (though may not be de-
> pending on what you want), and I am willing
> to put out the sources for wider distribu-
> tion. It's just version 0.02 \ldots{} but
> I've successfully run bash, gcc, GNU make,
> GNU sed, compress, etc. under it.
> ```

By March 1992, Linux was up to version 0.95 (i.e., very, very close to an official release). After another two years of revision, the Linux kernel was finally at version 1.0, meaning that it was theoretically devoid of all bugs. As of this writing, Linux has reached version 2.2.

The increasing popularity of Linux has given rise to software packages that cut down on the time and hassle associated with getting started. These packages, called distributions, gather into one place everything a user needs to install Linux on his or her computer. The current TimeSys Linux/RT distribution is based on Debian, one of the most common — not to mention well-respected — distributions around. Debian has been making itself available since 1993 and is currently on version 2.1. However, as mentioned earlier, TimeSys Linux/RT can be used with any other popular Linux distribution as well by appropriately replacing the Linux kernel in the distribution with the TimeSys Linux/RT distribution.

To refresh your Linux memory, we've included some of the most commonly used commands in an appendix to this manual.

1.2 Introduction to TimeSys Linux/RT

TimeSys Linux/RT, as mentioned above, differs from other real-time Linux systems in that the kernel itself is modified to handle real-time applications. This means that, among other things, if a single real-time process crashes, the other processes and the kernel will continue safely along as if nothing happened. This stability represents a big improvement over other real-time variants, which were likely to bring the whole system down whenever one process crashed.

TimeSys Linux/RT has its base in the Linux resource kernel, or Linux/RK, developed at Carnegie Mellon University. Linux/RK is included in Linux/RT as a module that's called up whenever necessary. TimeSys Linux/RT has inserted code in the Linux kernel proper so that, every time something having to do with real time needs to be done, the kernel goes to Linux/RK for instructions on how to handle it. Thus, the real-time capability is there when it's needed but doesn't interfere with the rest of Linux when the user is doing something else.

To make TimeSys Linux/RT as convenient as possible to people who may need to use other pre-existing systems, the distribution includes, along with Linux/RK, RTAI (Real-Time Applications Interface) from DIAPM in Italy and RED Linux from the University of California at Irvine.

1.3 What's in This Book

Here in the User's Guide, you can learn pretty much everything you need to know to get TimeSys Linux/RT installed and working on your computer. What there is to know about Linux as a whole could fill a book a thousand times

this size, but this particular manual sticks to Linux basics and features specific to TimeSys Linux/RT. (If you're looking for more detailed information about particular aspects of Linux, there should be something for you somewhere in the vast world of Linux guidebooks.) Subjects covered include:

- Installing TimeSys Linux/RT (Chapter 2)
- Overview of the System (Chapter 3)
- Running Real-Time Programs (Chapter 4)
- Programming with TimeSys Linux/RT (Chapter 5)

More in-depth information about programming with TimeSys Linux/RT can be found in the Programmer's Manual.

1.3.1 Conventions Used in This Guide

- **bold** highlights important words (i.e. do **not** try to install Linux unless you have hours of time to spare)
- *italic* is used for unfamiliar words the first time they're encountered in the text (i.e. *module*) or when discussing a generic class of data (i.e. *variable1*)
- `courier` is used for input to or output from programs, including names of programs, program code, commands, and files (i.e. type `ls -l foo` at the prompt)
- **`bold courier`** represents input the user has typed in verbatim
- *`italic courier`* is used to name a generic class of data that appears in a particular input or output (i.e. `ifdef` *`variable-name`*, with *`variable-name`* of course being replaced by the actual name of the variable whenever it's entered by a user).

Chapter 2:

Installing TimeSys Linux/RT

Installing TimeSys Linux/RT is only as hard as performing a Debian installation and inserting some modules. TimeSys Linux/RT can *also* be used with other distributions including RedHat, Suse, and Mandrake. This is because TimeSys Linux/RT consists of changes and extensions to the Linux core kernel, which is the same in most Linux distributions.

2.1 Installing with Debian

Here is what you need to do to install TimeSys Linux/RT.

First you need to perform a Debian Linux installation, because that is the distribution that TimeSys Linux/RT is currently using. If you have no existing Linux installation on the target system, then make sure your BIOS is configured to boot from the CD-ROM, and reboot with this CD-ROM in the drive. If you cannot boot from a CD-ROM, then you will need to create a boot floppy using the `/debian/dists/potato/main/disks-i386/current/images-1.44/rescue.bin` image. You can do this from DOS using the `rawrite` program in the `/debian/tools` directory, or from another Linux system by copying the image to `/dev/fd0`.

If you need more help installing Debian's Linux distribution, please refer to their website, *www.debian.org*. Here you can find online manuals that will walk you through the installation.

If the target system already has Debian installed, you can install the packages in the `/debian/dists/potato/local/binary-i386` directory using `dpkg`.

2.2 Installing With Distributions other than Debian Linux

As mentioned earlier, Linux/RT can *also* be used with other distributions including RedHat, Suse, and Mandrake.

If the target system has a Linux distribution installed which is *not* based on Debian, run the `/timesys/install` script from within the `/timesys` directory.

2.3 Post-Installation Process

Now that you successfully installed the kernel, you are ready to use the real-time implementations. There are two different implementations, but only one can be used at a time. These two implementations are RK ("Resource Kernel") and RTAI ("Real-Time Applications Interface"). They are implemented as kernel modules so they must be inserted when you want to use them. For information about kernel modules, please read the modules chapter. By default the RK module is loaded at boot time, so if you want to use the RK implementation, you can use it without doing anything. The following explains what you need to do to run TimeSys Linux/RT without real-time capabilities, how to load the real-time capabilities, and how to switch between RK and RTAI.

Here are the various ways you can run TimeSys Linux/RT and how to do it:

NOTE: *You must have superuser privileges to do these operations.*

If you want to run TimeSys Linux/RT without either real-time implementation:

All you have to do is remove the RK kernel module.

```
# rmmod rk
```

If you want to switch from one real-time implementation to the other:

You will call the rttype script which has one argument, the module you would like to switch to.

```
# rttype rk
// if you are using rtai, but want to switch
to rk\
# rttype rtai
// if you are using rk, but want to switch to
rtai
```

Of course, you can manually insert and remove the modules using the `insmod` and `rmmod` commands. If you want to list all of the modules that are loaded on the system, use the `lsmod` command.

If you want to use the TimeTrace tool:

You must insert the module named `measure-rk` or measure-rtai

```
# insmod measure-rk
```

or

```
# insmod measure-rtai
```

NOTE: *RK and RTAI modules cannot be inserted at the same time.*

Chapter 3:

Overview of the System

3.1 Introduction to Modules

As mentioned in the last chapter, the guts of TimeSys Linux/RT reside in a module — a chunk of code that can be inserted into the kernel for added functionality. This section explores the concept of modules in a little more detail and explains how to actually use them.

Modules often take the form of device drivers — code which lets the kernel communicate with a specific peripheral device such as a keyboard or CD-ROM drive. But, like TimeSys Linux/RT, they can also be used to extend the system in ways that don't have anything to do with any specific device.

To get a handle on what a module is, it helps to understand what a module is not — namely, an application. There are several important differences between a module and an application.

The first important difference is that, while an application is designed to perform and complete a particular task whenever it's called up, a module will register itself with the kernel on first run so that the kernel knows which functions it can evoke later on. So, since the ``main'' function of a mod-

ule serves to initialize it rather than do any actual task, the terminology is a little different from that of your average, everyday program. The traditional `main()` function becomes `init_module()`. Also, every module must end with a `cleanup_module` function, which tells the kernel when the module is finished.

Second, a module has less flexibility than an application in calling functions. As you know, an application can access the functions found in any library provided it includes that library in its header files.

A module, on the other hand, can only be linked to the kernel and so can only use functions defined in the kernel or found in the module itself. One result of this is that a module has no need of standard header files.

To load a module, use the command:

```
insmod <object_file>
```

`insmod` comes with a number of flags and options:

`-f`	Attempts to load the module even if the version of the kernel currently running and the version for which the module was compiled do not match.
`-k.`	Auto-clean; removes modules that haven't been used in some period of time, usually one minute
`-m`	Outputs a load map, making it easier to debug the module in the event of a kernel panic.
`-o module_name`	Explicitly name the module, rather than deriving the name from the base name of the source object file.
`-p`	Probe the module to see if it could

	be successfully loaded. This includes locating the object file in the module path, checking version numbers, and resolving symbols.
`-s`	Output everything to syslog instead of the terminal.
`-v`	Be verbose.

To unload a module, type:

```
rmmod module_name
```

`rmmod` also supports a few flags:

`-a`	Remove all unused modules.
`-s`	Output everything to `syslog` instead of the terminal.

3.3 Overview of TimeSys Linux/RT

***Figure 1*: The TimeSys Linux/RT mascot "taming" time.**

TimeSys Linux/RT's mascot, shown in Figure 1, is *Tux,* the Linux penguin riding a clock bullfighter-style. This image neatly symbolizes our goal — to give you the tools to "tame" time as effortlessly as a good bullfighter can tame a bull.

3.3.1 Predictable Real-Time Performance Within Pure Linux

TimeSys Linux/RT extends the Linux kernel, rather than adding a proprietary non-Linux RTOS as an abstraction layer between Linux and the system hardware. By enhancing the actual operating system, engineers can build hard real-time systems with TimeSys Linux/RT, while also enjoying the reliability and stability that have become the hallmark characteristics of the Linux OS.

The TimeSys Linux/RT architecture ensures that if a single real-time process crashes, the rest of the processes, as well as the kernel, will still run. This solves a major problem associated with the alternative, in which the entire Linux operating system could fail if a single process crashes. Nonetheless, TimeSys Linux/RT also offers engineers the ability to incorporate a layer called RTAI (Real-Time Applications Interface), which furnishes high performance and small system footprint characteristics.

TimeSys Linux/RT offers the best of both worlds. With TimeSys Linux/RT, developers can build real-time systems based upon the RTAI layer, emphasizing speed and small footprint for fully tested applications. And of course, you can also use TimeSys Linux/RT as a "true-blue" and pure real-time Linux OS, with all of Linux' robustness and reliability.

TimeSys supports TimeSys Linux/RT by offering a complete spectrum of fully interoperable software products, training, customization, consulting, and application engineering services that support all phases of software development for real-time systems. These include:

- **SuiteTime**® – A complete set of support tools from TimeSys, including TimeWiz®, TimeTrace™, and others currently under development.
  - **TimeWiz**® is a tool for timing analysis, simulation and modeling of single processor and distributed real-time systems including custom network protocols. It uses the popular rate-monotonic analysis (RMA) methodology for building predictable real-time systems, and can be readily customized and extended for other schemes as well.
  - **TimeTrace**™ measures and displays the exact execution sequence within a real-time target and can also provide the execution time data necessary for timing analysis by TimeWiz®.
- **a real-time Java Virtual Machine** – TimeSys will release a real-time Java Virtual Machine that will run on top of TimeSys Linux/RT, and will comply with Sun Microsystems' Real-Time Java Specification. TimeSys is a member of Sun's Real-Time Java Experts Group (RTJEG), the body that is creating the specification.

3.3.2 The TimeSys Linux/RT Architecture

TimeSys Corporation has incorporated a **critical set** of components into its Linux/RT offering that, together, offer a highly innovative approach to meeting time constraints. These components can be combined in some critical ways to handle a wide variety of application requirements. The basic components of **TimeSys Linux/RT** are:

- Current Linux Components
- Resource Kernel (RK)
- RED Linux (RED)
- Real-Time Applications Interface (RTAI)

Next, let us look at each component individually and what combinations can be created to meet response time requirements.

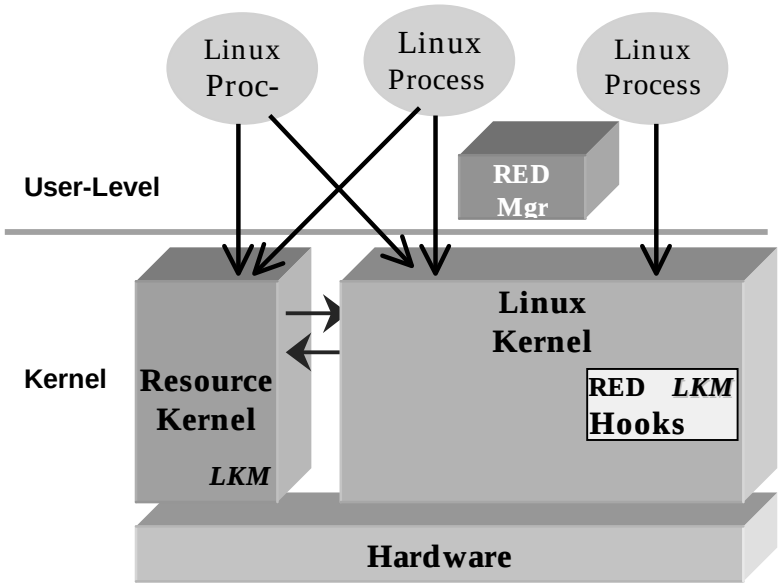


***Figure 2*: Resource Kernel (RK) configuration of TimeSys Linux/RT**

RESOURCE KERNEL

The Resource Kernel (RK) shown in Figure 2 is a Linux Loadable Kernel Module (LKM) that takes over some of the most critical real-time functions of the operating system to provide critical support for meeting bounded time constraints. For example, it supports **Fixed-Priority Scheduling with 256 priority levels**. These can be used to directly support the highly predictable **Rate Monotonic Analysis** techniques.

A unique capability of the RK is its support for a **Temporal Firewall**. This capability allows an application to reserve CPU capacity (soon disk and network capacity) that can be

guaranteed to remain available to the application, even if it or another high-priority application component goes out of control and attempts to monopolize the CPU.

In addition, RK supports core OS features required to build real-time systems:

- **priority inheritance** to minimize problems of priority inversion,
- **high-resolution clock and timers**,
- first-class support for **periodic threads**, and
- **fine-grained control** over which processes run after their reservations expire.

With available support for pinning down of memory pages of real-time processes, predictable and deterministic execution of Linux processes can be accomplished.

RED LINUX

TimeSys Linux/RT includes RED Linux support for system event logging and user event logging will be included. Associated with this will be special “hooks” needed to fully support TimeSys’ innovative TimeTrace™ tool that supports fine-grained visualization of system-level and user-level events. Task execution times and statistics get displayed automatically.

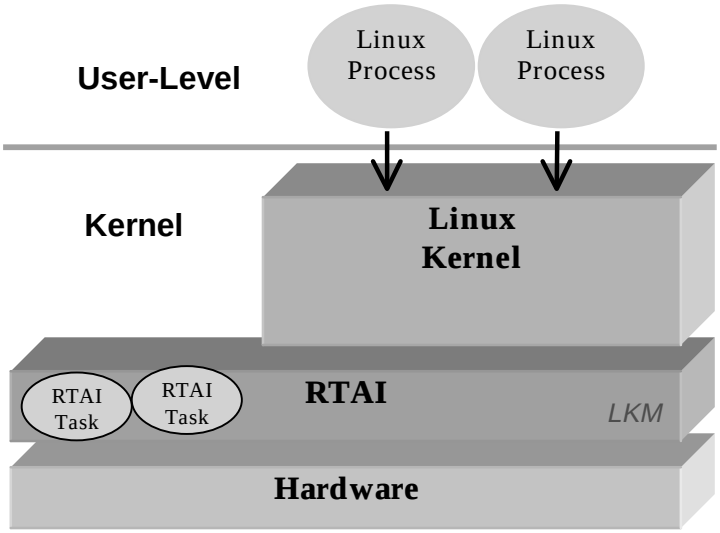


***Figure 3*: The RTAI configuration of TimeSys Linux/RT**

RTAI (Real-Time Applications Interface)

The RTAI (Real-Time Applications Interface) kernel configuration shown in Figure 3 provides a compact, extremely high-performance **open-source executive** that operates directly on the underlying hardware, allowing the remainder of the Linux/RT components to get processor control when the RTAI kernel or its high performance applications do not need it.

RTAI provides fixed-priority scheduling, message queues, and synchronization. RTAI should be used for well-tested applications, because "crashes" of RTAI applications will cause the entire system including the Linux kernel to "crash" also. When RTAI applications interface with system devices, specialized device drivers, separate from the Linux device drivers, will be necessary.

RTAI will soon be able to run **stand-alone** (i.e., without Linux) as an open source kernel for applications requiring high performance and a very small memory footprint.

COMBINATIONS OF TIMESYS LINUX/RT COMPONENTS

Although RTAI is designed to operate with Linux, its applications remain separate from the Linux environment. This is good from the performance perspective, but it means that its support for meeting bounded time constraints is not extended to Linux applications. Thus, RTAI cannot be used in combination with the RK or the RED Linux components.

LOADABLE KERNEL MODULES

TimeSys Linux/RT makes important advantage of Linux loadable kernel modules (LKMs) for all of its components. This allows the application to position itself among a **wide range of memory** footprints. LKMs are object modules that can be loaded (inserted) into or removed from the kernel at run-time. From the application's point of view, the system calls made available by these modules are **indistinguishable** from the system calls in the kernel itself, since the modules are run in system (as opposed to user) mode.

3.3.3 Target Application Domains, Components, and Tools

TimeSys Linux/RT is aimed at a host of embedded and real-time applications in domains including telecommunications, process control, industrial automation, Internet appliances, Web servers, multimedia servers, set-top boxes, high-definition TVs, medical electronics, avionics, and defense systems.

The TimeSys Linux/RT distribution supports and extends several innovations from the real-time Linux research community:

- Linux/RK (resource kernel) from Carnegie Mellon University.

- RTAI (Real-Time Applications Interface) from DIAPM in Italy.
- RED Linux from the University of California at Irvine.

TimeSys Linux/RT comes with effective tools and expert support to leverage the benefits of the open-source Linux operating system in the context of building real-time systems.

3.4 Overview of Resource Kernel (RK)

We now provide an overview of the most important subsystem inside TimeSys Linux/RT, called Linux/RK where RK stands for "*Resource Kernel*".

3.4.1 Linux/RK Capabilities

The Linux/RK subsystem of TimeSys Linux/RT provides the following capabilities:

- **Fixed-priority scheduling** with 256 priority levels: You can use the standard POSIX-compliant calls to assign a priority to any Linux process.
- **Priority inheritance** to avoid unbounded priority inversion: Timing problems from potentially unbounded priority inversion can be eliminated by the use of priority inheritance protocols using the Real-Time POSIX threads library and kernel support provided by TimeSys Linux/RT. The APIs used by TimeSys are the same as POSIX in this regard.
- **Quality of Service** (QoS) support for **Resource Reservation**: TimeSys Linux/RT, through the Linux/RK module, provides direct support to deliver guaranteed Quality of Service (QoS) to your real-time applications. An application can explicitly request and obtain CPU and timing guarantees.

- **High-Resolution Clocks and Timers**: Linux/RK supports high-resolution clocks and timers. Resolutions of a few microseconds or better are available.
- **Periodic Real-Time Tasks**: Periodic execution of tasks is a common requirement in real-time systems. TimeSys Linux/RT allows Linux processes to be marked as periodic processes, in which case they will be executed in periodic fashion.
- **Memory Wiring**: The physical memory pages of a real-time process can be "locked" by Linux so that they are not swapped out by the paging system. Otherwise, the predictability of real-time processes can suffer significantly.

3.4.2 Introduction to Resource Kernels

TimeSys Linux/RT has its roots in the resource kernel (RK) work done by Prof. Raj Rajkumar and his research group at the Real-time and Multimedia Systems Laboratory at Carnegie Mellon University in Pittsburgh. Behind the module that enables TimeSys Linux/RT to handle real-time requests lie the concepts of the "resource kernel". A *resource kernel* is one which provides timely, guaranteed, and enforced access to system resources to applications. It allows applications to specify only their resource demands, leaving the kernel to satisfy those demands using hidden resource management schemes. This separation of resource specification from resource management allows OS-subsystem-specific customization by extending, optimizing, or even replacing resource management schemes. As a result, this resource-centric approach can be implemented with any of several different resource management schemes.

The *resource kernel* gets its name from its resource-centricity and its ability to:

- apply a uniform resource model for dynamic sharing of different resource types

- take resource usage specifications from applications
- guarantee resource allocations at admission time
- schedule contending activities on a resource based on a well-defined scheme
- ensure timeliness by dynamically monitoring and enforcing actual resource usage

A main function of an operating system kernel is to multiplex available system resources across multiple requests from several applications. The traditional non-real-time kernel allocates a time-multiplexed resource to an application based on fairness metrics during a certain period. With a resource kernel, an application can request the reservation of a certain amount of a resource, and the kernel can guarantee that the requested amount is available to the application. Such a guarantee of resource allocation gives an application specific knowledge of the amount of its currently available resources. A QoS manager or an application itself can then optimize the system behavior by computing the best QoS obtained from the available resources.

The original resource kernel project aimed to create a ''Portable Resource Kernel," one which could be adapted relatively painlessly to work with a variety of operating systems. The Real-time and Multimedia Systems Lab chose Linux as a basis for the first implementation of a resource kernel in part because of its popularity on a wide variety of platforms and the easy availability of its source code.

In order to ensure maximum portability, the system was designed to explicitly avoid making extensive modifications to the original Linux kernel, and TimeSys has followed this precedent. Thus, the real-time code was developed as an independent module, modifying the Linux kernel only to introduce several callback hooks that catch relevant scheduling points in the Linux kernel and send these events to the RK. The resource kernel module uses the well-defined functions in the Linux kernel to control kernel entities, such as processes and device drivers.

TimeSysLinux/RT has inserted the following types of callback hooks in the Linux kernel:

- **schedule callback hook:** within `schedule()`, the scheduling function of the Linux kernel
- **interrupt callback hook:** intercepts Linux interrupt handling
- **interrupt out callback hook:** used to notify the resource kernel when an interrupt has finished being processed
- **kernel out callback hook:** used to notify the resource kernel when the execution is returning from kernel mode to the user level

3.4.3 Reserves and Resource Sets

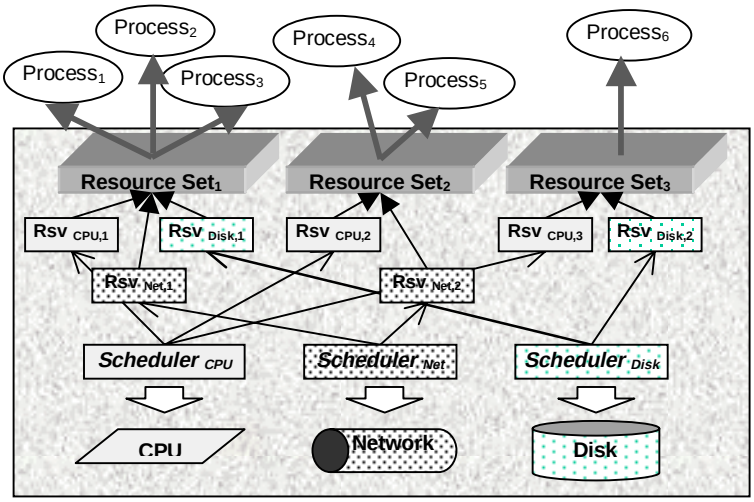


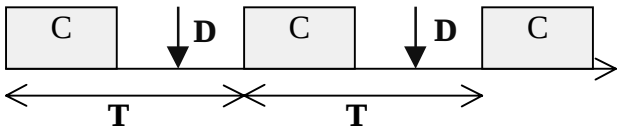
***Figure 4*: Resource Kernel (RK) Architecture**

The primary abstractions behind the resource kernel are the *resource capacity reservations* (*reserves* in short) and the *resource set*. The relationship between a resource set and re-

serves is illustrated in the Figure 4. While the Linux kernel provides normal operation system functions, the resource kernel uses reserves and resource sets — along with the necessary mechanisms for admission control, resource scheduling, resource usage accounting, and enforcement — to augment these functions.

A reserve represents a share of a single computing resource. Such a resource can be CPU time, physical memory pages, a network bandwidth, or a disk bandwidth. A certain amount of a resource is reserved for use by the programs. A reserve is implemented as a kernel entity; thus, it cannot be counterfeited. The kernel keeps track of the use of a reserve and will enforce its utilization when necessary. Appropriate scheduling and enforcement of a reserve by the resource kernel guarantees that the reserved amount is always allocated for it.

A reserve can by time-multiplexed or dedicated. Temporal reserves like CPU cycles, network bandwidth, and disk bandwidth are time-multiplexed, and spatial resources like memory pages are dedicated. A time-multiplexed resource has the following primary reserve parameters: *T*, *C*, and *D*, where *T* represents a recurrence period, *C* represents the processing time required within *T*, and *D* is the deadline within which the C units of processing time must be available within *T*. These parameters[1] are illustrated in Figure 5.[2]



[1] TimeSys Linux/RT also uses a "*B*" parameter in the specification of CPU reserves. It is used to represent priority inversion conditions, if any. This parameter can be normally set to zero.

[2] As of Release 1.0, TimeSys Linux/RT only supports the primary resource, CPU reserves. Support for network bandwidth reservation and disk bandwidth reservation will be added in the future.

***Figure 5*: The "Reservation" Parameters.**

A resource set represents a set of reserves. A resource set is bound to one or more programs, and provides the exclusive use of its reserved amount of resources with those programs. A resource set groups necessary resources for the job of user applications; thus, it is easy to examine and compare the utilization of each resource in it. If the kernel or a QoS manager finds an imbalance in resource utilization, an application will be notified and will be able to change its QoS parameters in order to balance the utilization.

When a reserve uses up its allocated time units C within an interval T, it is said to be depleted. A reserve which is not depleted is said to be an undepleted reserve. At the end of the current interval T, the reserve will obtain a new quota and is said to be replenished. In our resource management model, the behavior of a reserve between depletion and replenishment can take one of three forms:

- **Hard reserves:** will not be scheduled on depletion until they are replenished.
- **Firm reserves:** scheduled for execution on depletion only if no other un-depleted reserve or unreserved resource uses can be scheduled.
- **Soft reserves:** can be scheduled for execution on depletion along with other unreserved resource use and depleted reservations.

Reserves contain certain amounts of resources and control their utilization. A reserve may represent one of many different types of resources such as CPU cycles and network bandwidth. Different types of resources have their own accounting information and their own ways to deal with resource management. At the same time, reserves need to provide a uniform interface; otherwise, modifications are required each time a new resource type is added. Therefore, a reserve is de-coupled into abstract and real reserves. An abstract reserve implements the functionality common across all reserves and provides a uniform interface. A real

reserve implements resource-type-specific portions and exports functions that adhere to the uniform resource management interface. Abstract and real reserves are always paired. When a reserve is created, each of them is created and is coupled with each other. The distinction is useful because it requires only that real reserves be implemented for a new resource type.

Real reserves implement the following mechanisms which guarantee resource utilization based on reservation.

- **Admission control:** TimeSys Linux/RT performs an admission control test on a new request to determine if it can be accepted or not. If the request can be admitted, a reserve based on the requested parameters is created.
- **Scheduling policy:** A scheduling policy controls dynamic resource allocation, so that an application can receive its reserved amount of a resource.
- **Enforcement:** TimeSys Linux/RT enforces the use of a resource by an application based on its allocated reserves. An enforcement mechanism prevents a resource from being used more than its reserved amount.
- **Accounting:** TimeSys Linux/RT tracks how much of a resource an application has already used. This information is used by the scheduling policy and the enforcement mechanism. An application, a QoS manager, or a real-time visualization tool can also query this information for observation and/or dynamic resource allocation control purposes.

Reserves are gathered into a ''container" called a *resource set*, which provides a well-defined resource environment for applications. An execution object, a "process" in the Linux kernel, can be bound to at most a single resource set. Even when an execution object uses only a single reserve, it has to create a resource set and attach its only reserve to the set.

A resource set greatly simplifies the mechanism by which the current active reserve can be determined at run time. A

process also contains a reference to its resource set, and the resource set holds the references to its attached reserves.

3.4.3 Implementation Features

One important characteristic of the TimeSys Linux/RT implementation is its accurate time management. The combination of a timestamp counter with a high-resolution timer contributes to improving the precision of resource management.

A timestamp counter, built into most modern CPUs, provides the standard time for use by the resource kernel. The representation of time that the RK uses in accounting and scheduling is based on the values from this timestamp counter.

A high-resolution timer is supported to make the enforcement of reserves more precise. It is implemented by using the one-shot mode of the ISA clock timer chip in PC-compatible systems. The RK sets the latch for the next interrupt every time after a timer interrupt occurs.

In TimeSys Linux/RT, the interrupt callback hook calls the ISR (Interrupt Service Routine) in the resource kernel and processes timer interrupts. Interrupts are propagated to the Linux kernel as needed to ensure Linux compatibility.

The Linux kernel supports the proc filesystem, which provides, in a portable way, information on the current status of Linux kernel and running processes.

TimeSys Linux/RT uses the proc filesystem for providing information on the hardware platform, the reservation status, and the status of resource sets and reserves. Information on reserves includes the current, minimum, and maximum utilization of their underlying resources.

3.5 Overview of RTAI

One of the first important attempts to introduce real-time capabilities to Linux via modules came out of the department of aerospace engineering at Milan Polytechnic. Programmers there needed an operating system with a periodic scheduler in order to enhance efficiency in control applications that could work with a basic period and integer multiples of it.

They started out trying to develop a Real-Time Hardware Abstraction Layer (RT-HAL) onto which a Real-Time Application Interface (RTAI) could be mounted to make Linux usable for hard real-time applications. Unfortunately, kernel 2.0.25, the original basis for this project, was too unclean in design for the idea they had in mind. Thus, they switched gears and began to modify a kernel that NMT had introduced called RTLinux.

The initial release of RTLinux had had no real-time support functions (i.e. semaphore, timing functions, messages, etc.), which had made it impractical to implement relatively complex control applications requiring a few cooperating tasks and which meant that the Milan programmers had their work cut out for them.

They found that future versions of real-time Linux would require one-shot precise heterogeneous timers to implement PWM control systems at medium frequencies that, if done within the RTLinux tasks, would have allowed much more flexibility than a hard-wired implementation.

Again, however, the original release maintained its architecture and its overhead remained excessive. So the Milan programmers implemented the oneshot timers in a different way by using the CPU TSC (Time Stamp Clock), which was much more efficient but, with earlier-than-Pentium machines and compatibles, no more usable.

Another important reason for a variant was the fact it met the demand for a bug-free FPU support — a feature that, for

some time, had been lacking in the official release. Eventually, the official RTLinux solved most of these problems and added a periodic timer, along with semaphores and mailboxes. However, the programmers in Milan still thought the oneshot timing lacked efficiency.

When Linux kernel 2.2.xx made its debut, appearing to have a cleaner interface to the hardware, the Milan group went back to its original idea of developing an abstraction layer and an interface — but with a somewhat deeper understanding of what was behind it. The result was a comprehensive Real-Time Application Interface, usable both for uniprocessors (UP) and for symmetric multi processors (SMP), that allows the use of Linux kernel 2.2.xx for many hard real-time applications.

SMP tasks are defaulted to work on any cpu, but you can assign them to any subset, or even to a single cpu, by using the function `rt_set_runnable_on_cpus`. It is also possible to assign any real-time interrupt service to a specific cpu by using `rt_assign_irq_to_cpu` and `rt_reset_irq_to_sym_mode`.

Thus, a user can statically optimize his/her application if he/she believes that it can be better done than by using a symmetric load distribution. The possibility of forcing any interrupts to a specific cpu is clearly not related to the smpscheduler and can be used also with interrupt handlers alone.

Note that only the real-time interrupt handling is forced to a specific CPU. That means that if you check this feature by using cat /proc/interrupts for a real-time interrupt that is chained to Linux (e.g. the timer when `rtl_sched` is installed), you can still see some interrupts distributed to all the CPUs, even if they are mostly on the assigned one. That is because Linux interrupts are kept symmetric by the RTAI dispatcher of Linux irqs.

The schedulers allow you to choose between a periodic and a one-shot timer, not to be used together. The periodic ticking is less flexible but, with the usual PC hardware,

much more efficient. So it is up to you to choose the appropriate one for the applications at hand.

It should be noted that in the one-shot mode the time is measured on the base of the CPU time stamp clock (TSC) and not on the 8254 chip, which is used only to generate one-shot interrupts. The periodic mode is instead timed by the 8254 chip only. In this way, slow I/Os to the ISA bus are limited as much as possible with a sizeable gain in efficiency. The oneshot mode has just about 15-20% more overhead than the periodic one. It is likely that local APIC timers could lead to a further improvement.

Right now, local APIC timers are hard-disabled on UPs and a preliminary experience with a single SMP local APIC timer, to be released soon for SMP, shows that there is no performance improvement for a periodic scheduling when the one-shot case gain is sizeable, but not so large with respect to the already-available solution. In fact, by using the TSC, just two outb (approximately 2.5 us) are required to reprogram the 8254, as compared to almost nothing for the APIC timer. However, you have to broadcast a message to all the CPUs in any case, and that is about 2 us. The APIC bus is an open drain 2 wired one, and is not lightning-quick. Note that the performance loss of the 8254 is just a fraction of the overall task switching procedure, which is always substantially heavier in the one-shot case than in periodic mode.

Since the TSC is not available on 486 machines, these systems use a form of emulation of the ‘‘read time stamp clock” (rdtsc) assembler instruction based on counter2 of the 8254. So you can use RTAI also on such machines. Be warned that the one-shot timer on a 486 is a performance overkill because of the need to read the tsc, i.e. 8254 counter2 in this case, twice. That can take 6-8 us, i.e. more than it takes for a full switch among many tasks while using a periodic timer. Thus, only a very short period of a few Khz is viable for real-time tasks if you want to keep Linux alive.

No similar problems exist for the periodic timer that need not use any TSC at all. So, compared to the 20% cited above, the real-time performance ratio of the one-shot/periodic timer efficiency ratio can be very low on 486 machines. Moreover, it will produce far worse jitters than those caused on Pentiums and upward machines. If you really need a one-shot timer, buy a Pentium, at least. But if you care mainly about periodic timing, 486s can be still more than adequate for many applications.

A feature of the RTAI implementation is that interrupt handlers preambles take care natively of the task switched (TS) flag. Thus, you can freely use floating-point operations in your interrupt handlers, without causing a trap-fault whatever thing Linux is doing. RTAI is thus very suitable for trapping interrupts without taking Linux into account so that you can effectively interact with the bare PC hardware.

With RTAI you have the added advantage that Linux maintains all of its features untouched so that you can pass to it whatever you get from your handler for logging, displaying, and post-processing, by using FIFOs and/or shared memory. Imagine a remote controller at 10 Khz, +-5 us average interrupt uncertainty, connected through the internet, with all the bells and whistles of X and its applications. It is an application that was simulated easily.

Note that RTAI also has some very useful system services, including: timings, semaphores, messages, and remote procedure calls (RPC). These features make it easier to develop complex real-time applications.

RPCs are a limited form of QNX messages that pass either just an unsigned integer or a pointer to an unsigned integer for reason of efficiency. They can be easily changed to be fully compatible with QNX if you'd like.

3.6 TimeSys Tools and Services

TimeSys provides a complete set of solutions for users of TimeSys Linux/RT including a set of friendly tools for enhanced productivity, a (future) offering of a Real-Time Java virtual machine and a complete set of services including training, customization, consulting and turn-key project development services.

3.6.1 TimeWiz®: Modeling, Analysis, & Simulation

TimeWiz® is a product from TimeSys Corporation specifically designed for the construction of simple or complex real-time systems with predictable timing behavior. A screen-shot of TimeWiz is presented in Figure 6. TimeWiz lets you

- represent your hardware and software configurations,
- analyze the worst-case timing behavior of your system,
- simulate its average-case timing behavior,

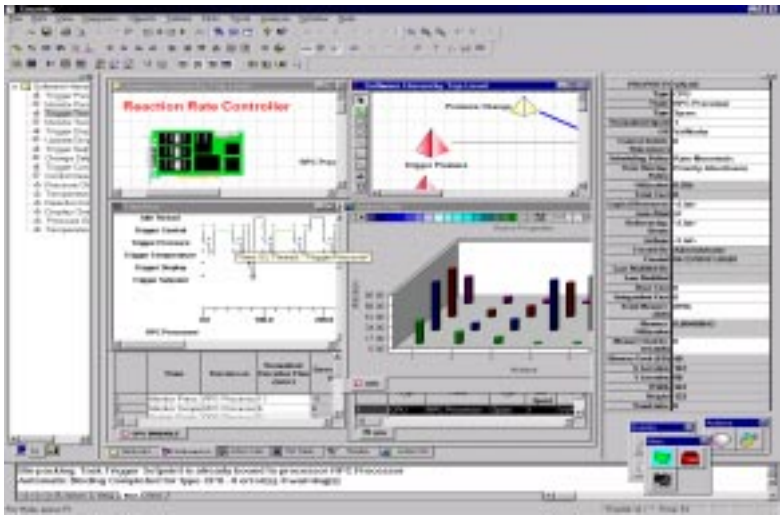


***Figure 6*: A screen-shot of TimeWiz®, a modeling analysis and simulation tool to predict and prove the timing behavior of your real-time system.**

- model processors *and* networks for end-to-end performance,
- chart your system parameters and generate integrated system reports.

TimeWiz currently runs on Windows NT and Windows 2000 platforms, and can also be run on Windows emulation platforms such as VMWare on Linux.

For more information, please contact TimeSys (*www.timesys.com*).

3.6.2 TimeTrace™: Visualization and Profiling

TimeTrace™ is a product from TimeSys Corporation that allows the monitoring and visualization of your real-time applications running on TimeSys Linux/RT. You can view scheduling, context-switching, system calls and user events as and when they happen. You can also obtain worst-case execution times, average execution times, and period information of your tasks. These can be used readily to analyze your system's timing behavior using tools such as TimeWiz.

- Profile your TimeSys Linux/RT target in real-time.
- Capture execution sequence on targets efficiently.

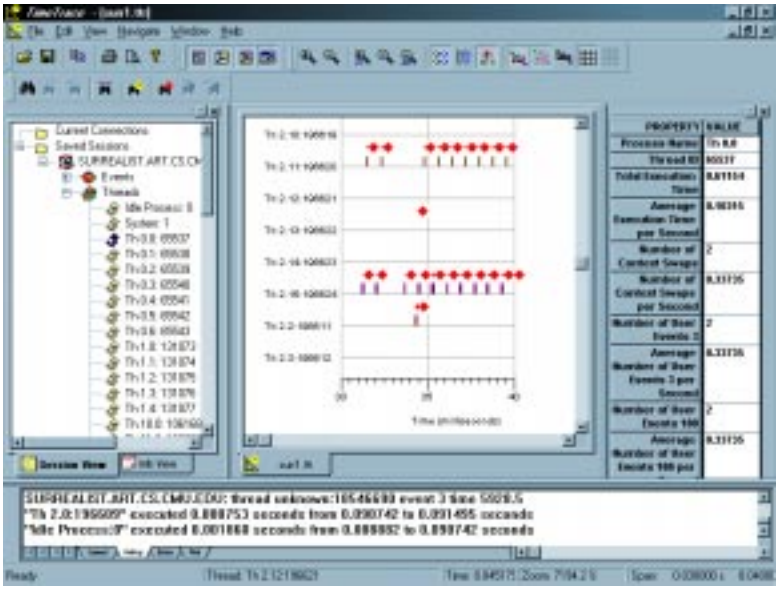


***Figure 7*: A screen-shot of TimeTrace™ to monitor, visualize, and profile the execution of TimeSys Linux/RT threads.**

- Display target execution sequences visually to create a "*software oscilloscope*"

- Monitor multiple TimeSys Linux/RT targets simultaneously from a single workstation (needs TimeTrace for TimeSys Linux/RT, Professional Edition).
- Feed TimeTrace data into TimeWiz as execution time and period parameters for worst-case analysis and/or average-case simulation.

TimeTrace for TimeSys Linux/RT is available in two editions:

- The Standard Edition of TimeTrace is part of the TimeSys Linux/RT Professional Edition.
- The Professional Edition of TimeTrace allows you to monitor multiple TimeSys Linux/RT targets from a single host.

TimeTrace runs on Windows NT and Windows 2000 platforms, and on Windows emulation software such as VMWare.

3.7 For More Information

For more information about TimeWiz, TimeTrace, Real-Time Java and other services including consulting, training, customization and turn-key project development, please contact TimeSys Corporation. Contact information can be found at *www.timesys.com*.

Chapter 4:

Running Real-Time Programs

4.1 Using Linux/RK User Programs

4.1.1 Benefits of Linux/RK

Linux/RK is an extension to the core Linux kernel. As a result, the RK subsystem of TimeSys Linux/RT supports a powerful array of capabilities:

1. ***Real* Real-Time Linux applications**: Any Linux process can now become a real-time process. You are no longer constrained to choose between a real-time OS and Linux; you do not have to embed a thin real-time OS layer below the Linux kernel; you just use Linux processes as is and imbibe them with real-time capabilities as you wish. It's that simple.
2. **POSIX support for your real-time needs**: TimeSys Linux/RT provides complete support for the traditional real-time systems paradigm of using a fixed-priority preemptive scheduling policy. In fact, it supports 256 priority levels. It also supports priority inheritance on mutexes to avoid the unbounded priority inversion problem. You can use standard Real-Time POSIX interfaces to access these functions.
3. **QoS Delivery**: TimeSys Linux/RT provides direct and explicit support for QoS delivery to your real-time ap-

plications. A discussion of this capability was provided in Chapter 3.

4. **Real-Time Support for Legacy Applications**: A pleasant surprise is that you can take existing legacy applications running on Linux and endow them with QoS guarantees, providing a Linux process with a guaranteed 30% of the processor, for example. You can actually specify whether you want this process to receive 3ms every 30 ms of time, or 300ms every 3 seconds.

4.1.2 Linux/RK Utilities

Linux/RK provides a wide-ranging API (applications programming interface) to allow real-time applications to access its internal capabilities. This API is summarized briefly in Chapter 5 and described in detail in the TimeSys Linux/RT Programmer's Manual.

In addition, Linux/RK provides a group of utilities: in LinuxRK/bin. These utilities include the following:

- `rklist`: lists the current resource sets in the system and their parameters.
- `rkdestroy`: allows to destroy a resource set (whose id specified using the hexadecimal format).
- `RKcleanRS`: a shell script that destroys *all* resource sets and their reserves in the processor.
- `rkattach`: allows you to attach a process (specifying its pid) to an existing resource set. Remember to specify the resource set id using the hexadecimal format. You can attach *any* Linux process using this utility, even if the process was written without any knowledge of RK.
- `rkdetach`: allows you to detach a process (specified by its pid) from an existing resource set.
- `rkexec`: allows you to create a new resource set with CPU reservation parameters, and attach a new process to the resource set. Again, this allows any legacy proc-

ess (written without any knowledge of TimeSys Linux/RT) to be able to use and benefit from the QoS guarantees provided by TimeSys Linux/RT.

- `clockfreq`: allows you to retrieve the processor clock frequency at which the CPU is running.

Any utility that needs parameters can be invoked without specifying any parameters and a "help" message specifying the required syntax will be printed out.

The source code for all these utilities are provided in the TimeSys Linux/RT distribution. The code also serves to provide very good examples of how to use and benefit from RK abstractions and primitives.

4.1.3 The "rolling" Demo Utility

PURPOSE

This program was designed to demonstrate how to create cpu resource sets and attach them to a process. All six balls are controlled by identical threads, with the exception that cpu resource sets are attached to processes controlling the blue balls.

***Figure 8*: Screen-shot of the "rolling" demo utility**

MAKING AND MODIFYING

This application is based on the GTK API. This directory should include `gtkpiechart.c`, `gtkpiechart.h`, `process.c`, `process.h`, and `rolling.c`. `gtkpiechart` is a custom widget for drawing piecharts. Running `make` should be all that is needed to create rolling.

RUNNING THE ROLLING DEMO

"`rolling`" needs two arguments.

The first argument is the number of milliseconds each real time ball will be given in computation time. The next number is the reservation period in nanoseconds.

RUNNING THE PROGRAM

Clicking the START button starts the ball rolling. The first set of balls both have resource sets attached to their process. The next line has one with a resource set. The last line consists of two regular processes.

The process is a simple loop that increments the x value of the ball, and makes checks to see if it has hit a side or the other ball.

The routine of creating the resource set, attaching it to a process, and creating the cpu reserve is detailed in the terminal in which the program was run.

The number of taps (hits) between paired balls is noted by the counter above the pair on the line.

The period and amount of cpu reservation can be controlled by entering values into the edit boxes and hitting the button. If the request is valid (i.e. the computation times of the three real time balls are within 70% usage of the specified period), the piechart gives a graphical depiction of the percentage of the time that the threads have exclusive use of the CPU during each period versus the rest of the system's processes (including the regular balls).

The HARD/SOFT button changes the reservation type of the real-time threads. The button's label is the current reservation type being used.

WHAT YOU SHOULD NOTICE

The top row should increase its number of taps rapidly, even at small reservation percentages. The next row should also do the same, since all three blue balls have (at even small computation times) plenty of time to update their x position.

The blue balls look like they're disappearing and reappearing elsewhere. This effect also stems from the fact that these balls have ample opportunity to update their x position.

You might notice that the middle row increases its taps the same, or more rapidly than the top row. Why? The logic for counting the taps is located in the `update_screen()` event in `rolling.c`. The top two balls are updating their x positions so rapidly, they often miss each other entirely as far as `update_screen()` is concerned. The green ball, on the other hand, does not have such a rapid update and can always tell when its blue ball is smacking against it. You'll see that at high computation times, this poor ball hardly gets to move. TimeSys Linux/RT had not anticipated this happening, but it is interesting nonetheless.

Play around with the computation times and the reservation periods. Keep the period over 20ms, however, since the x server has a tendency to freeze with shorter periods. (Maybe 1 out of a hundred times, but you've been warned. If this happens, telnet into the machine and kill the rolling process.

EXPERIMENT

Try the program with both HARD and SOFT to get a feel for how the processes react with the different scheduling priorities.

Try to reserve more time than is possible at the command line and you'll notice that not every thread gets a resource set. While running the program, increase the computation time until over 50% is allocated to the first three threads. You'll notice that everything else pauses for half of the period since they have to wait for the first three threads to be done with the cpu.

With low reservation periods and small computation times, the balls should slip across the screen rapidly, easily outpacing the other three balls. This shows the power of RT-- a small user program can easily control the usage of the CPU with just a few commands.

4.1.4 The TimeSys Resource Manager Tool

The TimeSys Resource Manager (TRM) is a tool for managing the resource sets in a system running the Resource Kernel. It allows you to create, rename, and delete resource sets, attach and detach processes from resource sets, and modify the mode, period, and cpu time of a resource set.

Figure 9: **TRM screen-shot showing creation of resource set**

Figure 9 shows a screenshot of TRM. A resource set has been created and renamed to "My Resource Set". Resource sets are created by selecting "Add Resource Set" from the menu near the top of the screen. A different resource set can be selecting by choosing it from the menu. Resource sets are listed by their identifier (a hexadecimal string) and their name. Existing resource sets can then be renamed by type a new name in the name field and hitting return. The cur-

rently selected resource set can be deleted by clicking on the "Delete" button.

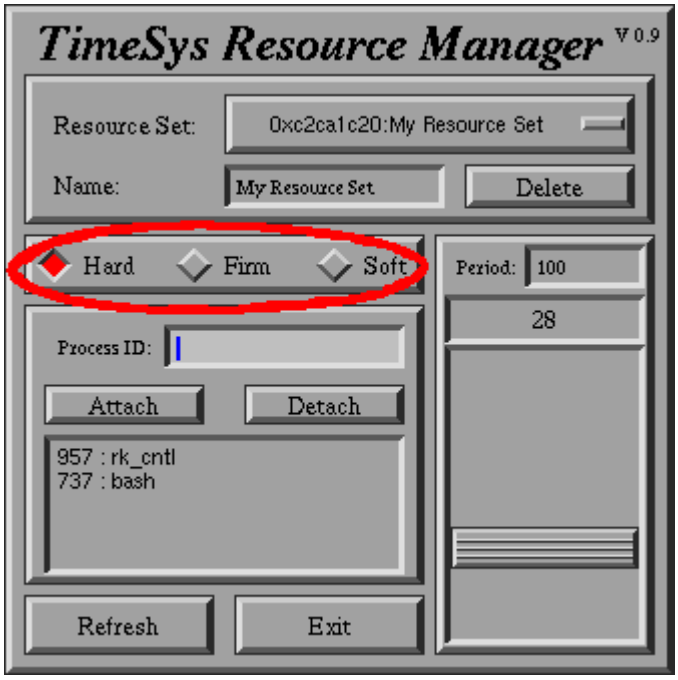


Figure 10: **Mode settings for the Resource Manager**

The mode of a cpu reservation can be set to Hard, Firm, or Soft. If the mode of a reservation is set to Hard, then the processes attached to the reservation will get exactly as much cpu time as they have been allotted. If the mode of a reservation is set to Soft, then processes will be able to get more time than they have been allotted if there is not much activity on the processor.

Figure 11: **Attaching processes to a resource set**

Processes can be attached to a resource set by typing their process ID into the "Process ID:" field, and then clicking the "Attach" button. If the process ID is valid, then the process will appear in the list, along with the name of the process. To detach a process from a resource set, simply click on the process you wish you detach, and then click the "Detach" button.

Figure 12: **Modifying the CPU time of a resource set**

The cpu time of a resource set can be modified by entering a new period into the "Period:" field, and hitting return. The cpu time can be adjusted by moving the slider in the lower right corner of the window.

There are also "Refresh" and "Exit" buttons. The refresh button refreshes the information on the screen. This is only necessary when the resource sets are being modified by other processes on the system. The exit button will exit TRM.

4.1.5 Other TimeSys Utilities and Demos

We recommend that you, the gentle reader, access the TimeSys Web site at *www.timesys.com* on a regular basis for downloading other utilities and demonstration code that are constantly being created. For example, a video-

conferencing application that runs on TimeSys Linux/RT will be available in the 2Q 2000 from TimeSys.

4.2 Using RTAI User Programs

The following is a general description of how to run user programs. For a more detailed description and a walk-through of this process, please read the chapter program in the Programmer's Guide on creating and executing an RTAI.

Before running a user program, the core RTAI modules need to be inserted. These core modules are the actual real-time implementation of Linux. Without inserting these modules, all that you are running is Linux without real-time. The files `ldmod` and `remod` can be used to `insmod` and `rmmod` all of the core modules at once: `rtai`, `fifo`, and the installed scheduler module with the default parameters. For the scheduler, the modules are: the CPU frequency, as set in the corresponding macro `CPU_FREQ` in `rtai.h`; a periodic scheduler and Linux assumed not to use the FPU. You can change any of the above parameters by either setting the corresponding macros in `"rtai.h"` or typing the following command when installing the module:

```
insmod
/usr/src/linux/modules/rtai_sched CpuFreq=<x>
LinuxFpu=<y> OneShot=<z>
```

x is the CPU frequency in Hz. If *y* == 0, Linux does not use the fpu but if *y* ! = 0 it does. If *z* == 0, it uses a periodic timer but, if *z* != 0, it uses the oneshot timer. Clearly, you can set any combination of the above parameters. The CPU frequency can be changed just by compiling the scheduler modules, or the functions: `rt_linux_use_fpu(int yes_no)`, `rt_set_periodic_mode()`, `rt_set_oneshot_mode()`, can be used during installation to set/reset the scheduler dynamically. Note that any setting of the timer mode stops any timer currently running.

After finishing this, try to run some user programs. Some test cases are available in the `/usr/src/rtai/examples` directory. All the examples have some macros that allow you to experiment with forcing tasks and timer interrupts to any CPU and a summary of CPU usage is printed at module removal.

Be careful in setting the macro `TICK_PERIOD` (nanosecs) in the various examples to a value appropriate to your machine. The defaults work for a 200Mhz PPro and can be too demanding for lower-class Pentiums and 486s. In any case, read the appropriate README file before running the corresponding examples. In all the tests, the choice of which timer, periodic or one-shot, to use is done by commenting/uncommenting the macro `ONE_SHOT`. It is important to remark that, since in all tests a one-shot timer is set specifically at module load time, any choice made at the scheduler installation is overridden.

It should also be noted that, if the timer mode is chosen at module installation, it must be done by adding a call to `rt_set_[oneshot/periodic]_mode()` before any time conversion or service is requested by the scheduler. Thus, in the case of a multi-module application set, the mode in the very first module should be loaded or put in a common header file, as is done in the digital wrist clock examples (see file `clock.h` in directories `sem_clock` and `msg_clock`). In the directory `jitter_free_sw` there is an example that shows how, by loosing computer power, you can get an almost jitter-free scheduling.

Note that the examples related to the use of the MUP scheduler are in the directory `mups_examples`. The RTAI directory includes html files that document the various function calls (in `doc_rtai`) and even more recent information is included in the various README files. You can also read the TimeSys Programmer's Guide for further information.

Chapter 5:

Programming With TimeSys Linux/RT

5.1 Programming in Linux/RK

A brief description of the APIs added to the Pure Linux kernel by TimeSys Linux/RT follows.

5.1.1 Building Linux/RK programs

To compile a program for RK, please link to the library `librk.a` located under "`LinuxRT/lib`" (or `/usr/lib/rk` depending on your installation).

The macro `__RK__` must be defined in each of your program source files and is typically defined in your `Makefile` to build your TimeSys Linux/RT programs.

5.1.2 Linux/RK Capabilities

The Linux/RK subsystem of TimeSys Linux/RT provides the following capabilities:

Fixed-priority scheduling with 256 priority levels: You can use the standard POSIX-compliant calls to assign a priority to any Linux process.

Priority inheritance to avoid unbounded priority inversion: When you use mutexes with the POSIX threads library, unbounded priority inversion can occur. To understand this concept better, the reader may want to obtain an article titled "*What Happened on Mars?*" from TimeSys, and their "**Pocket Bible**" on real-rime systems called "*A Concise Handbook on Real-Time Systems*". As the name implies, uncontrolled priority inversion can be unhealthy to your system's ability to meet its timing constraints.[3]

Quality of Service (QoS) support for **Resource Reservation**: Linux/RK provides direct support to manage QoS delivery to your real-time applications. An application can explicitly request and obtain CPU and timing guarantees. This is accomplished through the use of "Resource Sets" and "CPU Reservations ". A more detailed overview of the Resource Kernel (RK) and its support for QoS delivery is available in Chapter 3.

High-Resolution Clocks and Timers: Linux/RK supports high-resolution clocks and timers on Pentium-class processors and beyond. Resolutions of a few microseconds or better are available.

Periodic Real-Time Tasks: Periodic execution of tasks is a common requirement in real-time systems. For example, video processing and sound processing are typically done periodically in multimedia systems. Periodic sampling of sensor signals is very common in feedback control systems. Linux/RK allows Linux processes to be marked as periodic processes, in which case, they will be executed in periodic fashion.

Memory Wiring: Real-time processes can suffer adverse timing consequences if their memory pages are swapped to disk during execution. (Linux, by default, uses demand-paging of processes). The physical memory pages of a real-time process can be "locked" by Linux so that they are not swapped out by the paging system.

[3] You *must* link your programs to the TimeSys Linux/RT library "`libpthreadsrt.a`" to obtain support for priority inheritance.

We next provide additional details and a brief description of the application programming interfaces available to use each of the above capabilities.

5.1.3 Resource Sets and CPU Reservations

For more information about the Resource Kernel paradigm, please see the paper on Portable Resource Kernels and Resource Kernels.

To use CPU reservations, the following steps are necessary:

1. Create resource set.
2. Create CPU reservation for the resource set.
3. Attach a process to the resource set.
4. *<program execution>*
5. Destroy resource set.

You must include <rk/rk.h> to get these function prototypes and link with "librk.a" in your application programs.

- `rk_resource_set_t rk_resource_set_create(char *name)`

  Creates a resource set with a name.

- `rk_resource_set_destroy(rk_resource_set_t rs)`

  Destroys resource set rs.

- `rk_resource_set_attach_process(rk_resource_set_t rs, pid_t pid)`

  Attaches the specified process to a resource set which must already exist.

- `rk_resource_set_detach_process(rk_resource_set_t rs, pid_t pid)`

  Detaches the specified process from a resource set.

- `rk_resource_set_get_name(rk_resource_set_t rs, char *name);`

  Returns the name of the specified resource set.

- `rk_reserve_t rk_resource_set_get_cpu_rsv(rk_resource_set_t rs);`

  Returns the CPU reserve (if any) attached to a resource set.

- `rk_reserve_t rk_proc_get_resource_set(pid_t pid);`

  Returns the resource set (if any) attached to a process.

- `rk_resource_sets_get_num(void);`

  Returns the number of resource sets currently in the system.

- `rk_resource_sets_get_list(rk_resource_set_t *rs, int count);`

  Returns the list of resource sets in the system.

- `rk_resource_set_t rk_proc_get_rset(pid_t pid)`

  Returns the resource set to which the process specified by pid is attached to.

- `rk_resource_set_get_num_procs(rk_resource_set_t rs)`

  Returns the number of processes attached to a resource set.

- `rk_resource_set_get_proclist(rk_resource_set_t rs, pid_t *procs);`

  Returns the list of processes attached to a resource set.

- `rk_reserve_t rk_cpu_reserve_create(rk_resource_set_t rs, cpu_reserve_attr_t attr);`

  Creates a CPU reservation and attaches to resource set rs. The amount of CPU reservation is specified with struct cpu_reserve_attr (defined in <rk/rk.h>). It permits the definition of computation time (C), period (T), deadline (D), blocking time (B, typically 0), and enforcement mode (hard, or soft).

Currently, TimeSys Linux/RT supports `RSV_HARD` and `RSV_SOFT`.

- `RSV_HARD`: guaranteed to receive the specified amount on success

- `RSV_SOFT`: guaranteed to receive the specified amount on success. If resource is still available after using up guaranteed amount, it will compete against unreserved tasks for more CPU time.

- `cpu_reserve_ctl(rk_resource_set_t rs, cpu_reserve_attr_t cpu_attr)`

  Changes the properties of existing CPU reservations (computation time, period, deadline, blocking time and enforcement mode.)

- `rk_cpu_reserve_delete(rk_resource_set_t rs);`

  Deletes the CPU reserve associated with a resource set.

- `rk_cpu_reserves_get_scheduling_policy(void);`

  Returns the scheduling policy used to schedule CPU reserves. The policy can either be `RATE_MONOTONIC` or `DEADLINE_MONOTONIC`.

- `rk_cpu_reserves_set_scheduling_policy(int policy);`

  Sets the scheduling policy used to schedule CPU reserves. The policy can either be `RATE_MONOTONIC` or `DEADLINE_MONOTONIC`.

- `rk_cpu_reserves_get_num(void);`

  Returns the number of CPU reserves currently in the system.

- `rk_cpu_reserves_get_list(rk_reserve_t *rsv, int count);`

  Returns the list of CPU reserves in the system.

- `rk_cpu_reserve_get_attr(rk_reserve_t rsv, cpu_reserve_attr_t attr);`

Returns the attributes of the specified CPU reserve rsv which include the reserve's computation time (C), period (T), deadline (D), blocking time (B, typically 0), and enforcement mode (hard, or soft).

- `sys_inherit(int mode);`

  Determines whether children created by this process inherit the resource set of the parent process.

Please consult example programs in `/usr/src/TimeSys/LinuxRK/examples`, the utilities in `/usr/src/TimeSys/LinuxRK/bin` and the TimeSys Programmer's Manual for additional details.

5.1.4 Priority Inheritance Support

You can use priority inheritance on mutexes to bound problems due to priority inversion in TimeSys Linux/RT. Due to the original coding of the Posix pthreads library in Linux, the support for priority inheritance requires the use of a new library called `libpthreads_rt.a`. Your application program **must** link with this new library to obtain the use of priority inheritance. The use of the older library will not give you the capability to use priority inheritance.

5.1.5 High-Resolution Clocks and Timers

Most, if not all, functions below are compatible with their IEEE POSIX 1003.1 standard counterparts.

- `unsigned long rt_get_clock_frequency(void);`

  Returns the system processor clock frequency in Hz.

- `clock_settime (clockid_t clock_id, __const struct timespec *tp);`

  Sets the time-of-day clock specified by to specified value.

- `clock_gettime (clockid_t clock_id, struct timespec *tp);`

Get the time of day.

- `clock_getres (clockid_t clock_id, struct timespec *res);`

  Get the resolution of the specified clock.

- `timer_create (clockid_t clock_id, struct sigevent *evp, timer_t *timerid);`

  Create a timer with the appropriate `sigevent`.

- `timer_delete (timer_t timerid);`

  `Delete the specified timer.`

- `timer_settime (timer_t timerid, int flags, __const struct itimerspec *value, struct itimerspec *ovalue);`

  Set timer value (the old timer value is returned).

5.1.6 Periodic Real-Time Threads

- `rt_make_periodic(struct timespec *period, struct timespec *start);`

  This is NOT persistent across exec and fork system calls. The calling thread is made periodic with the specified period parameter and its periodicity will start at time `start`.

- `rt_wait_for_start_time(void);`

  This function is called by a periodic task and allows the task to be delayed until the point in time when its periodicity starts.

- `int rt_wait_for_next_period(void);`

  This function is called by a periodic task to wait for its next (possible) period boundary. The task is blocked until the next boundary.

- `int rt_process_get_period(pid_t pid, struct timespec *period);`

  This function can be used to obtain the period value being used by a periodic real-time process.

- `int rt_process_set_period(pid_t pid, struct timespec *period);`

  This function can be used to set the period value being used by a periodic real-time process.

5.1.7 Physical Memory Management

System calls to support locking and unlocking of memory pages are available as part of standard Linux.

- `mlock(caddr_t addr, size_t len);`

  Run "`man mlock`" for more information.

- `mlockall(int flags);`

  The `mlockall()` function locks in memory all pages mapped by an address space.

  The value of `flags` determines whether the pages to be locked are those currently mapped by the address space, those that will be mapped in the future, or both:

  - `MCL_CURRENT`: Lock current mappings.
  - `MCL_FUTURE`: Lock future mappings

  Locks established with `mlockall()` are NOT inherited by a child process after a fork() call, and are NOT nested.

- `munlock(caddr_t addr, size_t len);`

  `munlock()` removes locks established with `mlock()`.

- `munlockall(void)`

  The `munlockall()` function removes address space locks and locks on mappings in the address space.

5.2 Programming in RTAI

This section presents the RTAI program: its structure, environment, and unique needs. The most important thing to

remember is that RTAI programs work at the kernel level. All of the supporting functions, such as shared memory, task scheduling (FIFO, LXRT), and the core functionality itself (RTAI) are attached to the Linux kernel as modules. The RTAI root directory contains executable scripts (`ldmod`, `remod`) for inserting and removing the core modules. This is discussed in more depth in the Programmer's Guide. Since all execution takes place in the operating system, you will need either to be logged in as root or to initiate super user privileges. Remember, working with root privileges creates the chance for serious damage to your operating system by accidentally modifying or deleting necessary files. Also, since the code that you are creating will be attached directly to the kernel, it is possible to create tasks that will starve the rest of the system. The only other option after releasing such a monster on your processor(s) is a cold boot. To paraphrase Elmer Fudd, "be vewy, vewy, vewy careful."

For those not initiated in the ways of the module, please refer to the Programmer's Guide for a longer discussion. There are basically two things to be done with them: insert and remove. The RTAI programs that you write will be compiled into executables that are to be inserted as modules. As mentioned above, depending on the functions that you use, certain modules included in the RTAI package will need to be inserted before your module will execute properly.

RTAI was designed to work with both single and multiple processors (SMP), including restricting task execution to individual processors on a multiple processor system (MUP). It includes functions for creating, timing, and scheduling tasks, ones for inter-processes communication, remote procedure calls, collaborating Linux and RTAI threads, and also maintaining semaphores. One set of functions (rtf_) creates a real-time FIFO that writes to a device and can be read by Linux processes using standard input functions. All the functions and the data structures related to them are discussed in length in the programmer's manual. RTAI's purpose is to present the programmer and those who use the programs with a unified API that can be used for the spectrum of real-time programming: hard, firm, or soft.

RTAI programs, or modules (as they will be referred to from now on) are written in C. There are a number of functions necessary for each module to work properly. Each RTAI module needs an `init_module` and `cleanup_module` to work properly. There is no `main()` in an RTAI program. The initialization of all the threads and the program's environment is taken care of by the `init_function`. Without this function, your module will not run correctly. When the module is removed from the kernel, the `cleanup_module` function takes care of the messy details, such as destroying the module's threads and freeing any devices that might have been utilized by the module.

Most of the work that will be done by your module is included in the task functions. They can be included right in the primary file or could be spread across several files. Tasks are declared at the top of the primary file and as mentioned before, are initialized in the `init_module` function. This is where the characteristics of the tasks, such as periodic rate and synchronization, are stated. Task routines are a very important part of the module — if they didn't exist, the module would not be able to accomplish anything. This should provide you with a good idea what comprises an RTAI program and how it is used. For a more in-depth look at programming in RTAI, please consult the TimeSys Linux/RT Programmer's Guide.

Appendix: Linux and TimeSys Linux/RT Commands

A. 1 Linux Commands

The following are some of the most common and useful Linux commands. If you need more information about using Linux, we suggest that you grab one of the many good Linux manuals on the market.

`cat [`*`filename`*`]`	The `cat` command scrolls the contents of the file *filename* across the screen.
`cd [`*`directory name`*`]`	The `cd` command changes the directory you're in. There are a variety of different parameters that you can put into *directory name*:
`cd ..`	Moves you up one directory.
`cd ~`	Moves you to your home directory. You can also move to your home directory by putting nothing in the directory name parameter.
`cd` *`name`*	Move you to the *name* directory. For more details on these commands, such as options and parameters, please read the man pages supplied in the Linux distri-

	bution.
`cp [oldfile] [newfile]`	The `cp` command lets you copy *oldfile* to *newfile.*
`dir [directory name]`	The `dir` command displays the contents of the directory *directory name.* If you leave *directory name* blank, it will display the contents of the current directory.
`echo [string]`	The `echo` command prints the string `string` to the display or can be redirected to a file, device, program, or your shell.
`find [directory to start search] [filename] [action for list]`	The `find` command searches the directory *directory to start search.* and all subdirectories, for the file *filename* and *action for list* is what the command does with the list.
`grep [text] [file]`	The `grep` command searches the file *file* for the text pattern *text* and prints to the screen all of the portions of the file in which *text* was found.
`insmod [module]`	The `insmod` command inserts the module *module* into the kernel. You must be logged in as root or have super-user privileges to use this command.
`less [filename]`	`less` is a program which displays the contents of the file *filename* to the screen like

	the `more` program. `less` allows you to move backwards in the file as opposed to `more`, which only allows you to move forward through the file.
`ls [`*`directory name`*`]`	The `ls` command lists the contents of the directory *directory name*. You can change the format of the printed list via options which can found in the man page for `ls`. If you leave *directory name* blank, it will list the contents of the current directory.
`lsmod`	The `lsmod` command lists all of the modules that have been inserted into the system.
`make`	`make` is a utility that finds out which parts of a large program need to be recompiled and issues the commands needed to do the recompilation.
`man [`*`subject`*`]`	The `man` command formats the online manual pages for the subject *subject* and displays that information to the screen. It is very useful because it gives very detailed information about commands and other things. It is advised that you read the man pages on any of the commands you look up in this appendix.

`mkdir [`*`directory name`*`]`	The command `mkdir` creates the directory *directory name* in the current directory you are in, unless you give a full path name for the *directory name*, which will then create it there.
`more [`*`filename`*`]`	`more` is a program which displays the contents of the file *filename* to the screen like the less program. `more` only allows you to move forward in the file as opposed to `less`, which allows you to move in both directions through the file.
`mount [`*`directory name`*`]`	The `mount` command attaches the filesystem to the directory *directory name*. If *directory name* is left blank, the command will list all of the currently mounted filesystems.
`mv [`*`object1`*`] [`*`object2 or destination location`*`]`	The `mv` command moves *object1* into *object2* or into the destination location. In other words, you can move a file into another file, or you can move a file into a directory.
`ps`	The `ps` command displays a snapshot of all the current processes.
`pwd`	The `pwd` command displays the path of the current directory you are in.
`rm [`*`filename`*`]`	The `rm` command removes the file *filename* from the

	system. Be very careful with this command because there is no way of retrieving the file once it has been removed.
`rmdir [`*`directory name`*`]`	The `rmdir` command allows you to remove the empty directory *directory name*. Remember, *directory name* must be empty.
`rmmod [`*`module`*`]`	The `rmmod` command removes the module *module* from the kernel. You must be logged in as root or have super-user privileges to use this command.
`su`	The `su` command allows you to have superuser privileges. It will ask you for a password. When it does, you must put root's password in. It is now like you logged in as root.
`umount [`*`directory name`*`]`	The `umount` command detaches the filesystem from the directory [*directory name*].

A. 2 TimeSys Linux/RT RK Commands

`clockfreq`	Prints the clock frequency at which the system processor is running. The units are in MHz.
`RKcleanRS`	Destroys all resource sets and their associated reserves in the

	system.
rkattach <*resource set in hex*> <*process id*> [*process id*] …	Attach to the specified resource set the list of specified processes.
rkdestroy <*resource set in hex*> [*resource set*] …	Destroy the specified resource set(s).
rkdetach <resource set in hex> <process id> [process id] …	Detach from the specified resource set the list of specified processes.
rkexec --newrset --cpu [time in us] –period [period in us] --deadline [deadline in us] --hard (or [--soft] --exec '<args>'	Execute the specified process creating a new resource set (or specify an existing resource set to use using an —rset option). The CPU reservation parameters (CPU time, period and deadline) can be specified.
rklist	List the parameters of the current resource sets and their reservations in the system.

In addition, TimeSys Linux/RT also provides a powerful set of APIs for providing guaranteed QoS to new and legacy Linux applications (using its RK subsystems) as well as for high-performance requirements (using RTAI). These APIs are documented in detail in the TimeSys Linux/RT Programmer's Manual.

Glossary

The following definitions apply to terms used throughout this manual. Most terms are derived from the RMA terminology defined in the Handbook of Real-Time Systems[4]. A clear understanding of these terms is expected to be very useful to any user of Real-Time Systems using TimeSys Linux/RT and associated tools.

Action	The smallest decomposition of a response; a segment of a response that cannot change system resource allocation. In RMA terms, an action must be bound to a (physical) resource before it is analyzed. An action can also use zero, one or more logical resources.
Admission Control	The process of testing a new reserve request and accepting it, if possible. If the request can be admitted, a reserve based on the requested

[4] "*A Practitioner's Handbook for Real-Time Analysis: Guide to Rate Monotonic Analysis for Real-Time Systems*" from Software Engineering Institute and Carnegie-Mellon University by Mark H. Klein, Thomas Ralya, Bill Pollak and Ray Obenza and is published by Kluwer Academic Publishers.

	parameters is created.
Aperiodic event	An event sequence whose arrival pattern is not periodic.
Average-case response time	The average case response time of a response's jobs (instances). Also, see Output Jitter.
Blocking	The act of a lower priority task delaying the execution of a higher priority task; more commonly known as priority inversion. Such priority inversion takes more complex forms in distributed and shared memory implementations.
Blocking time	The delay effect (also called the "duration of priority inversion") caused to events with higher priority responses by events with lower priority responses.
Bursty arrivals	An arrival pattern in which events may occur arbitrarily close to a previous event, but over an extended period of time the number of events is restricted by a specific event density; that is, there is a bound on the number of events per time interval. Bursty arrivals are modeled in RMA using their minimum interarrival time and their resource consumption in that interval.
Data-Sharing Policy	A policy specific to a (physical) resource that determines how logical resources bound to the (physical) resource can be accessed. Some schemes do not provide any protection against priority inversion, while others provide varying degrees of protection. RMA supports multiple data-sharing policies including FIFO (no protection against priority inversion), priority inheritance protocol, priority ceiling protocol, highest locker priority protocol and kernelized monitor (non-preemptive execution) policies.

Deadline-monotonic scheduling algorithm — A fixed-priority algorithm in which the highest priority is assigned to the task with the earliest relative delay constraint (deadline) from each instance of its arrival. The priorities of the remaining tasks are assigned monotonically (or consistently) in order of their deadlines. This algorithm and the earliest deadline scheduling algorithm are not the same. In this algorithm, all instances of the same task have the same priority. In the earliest deadline scheduling algorithm, each instance of the same task has a different priority, equal to the absolute deadline (time) by which it must be completed. The rate-monotonic scheduling algorithm and the deadline-monotonic algorithm are one and the same when the relative deadline requirement and periods are equal (which happens very often).

Deterministic System — A system in which it is possible to determine exactly what is or will be executing on the processor during system execution, given any specific time. Determinism is a consequence of the scheduling policies supporting a group of processes.

Dynamic-priority scheduling policy — An allocation policy that uses priorities to decide how to assign a resource. Priorities change from instance to instance of the same task (and can also vary during the lifetime of the same instance of a task). The earliest deadline scheduling algorithm is an example of a dynamic priority scheduling policy.

Earliest deadline scheduling — A dynamic priority assignment policy in which the highest priority is assigned to the task with the most imminent deadline.

Event — A change in state arising from a stimulus within the system or external to the system; or because of the passage of time. An event is

	typically caused by an interrupt on an input port or a timer expiry. See also trace and trigger.
Execution time	Amount of time that an action or a response will consume on a CPU.
Firm Reserve	A reserve that is scheduled for execution on depletion only if no other undepleted reserve or unreserved resource uses can be scheduled
Fixed-priority scheduling policy	An allocation policy that uses priorities to decide how to assign a resource. The priority (normally) remains fixed from instance to instance of the same task. Rate-monotonic and deadline-monotonic scheduling policies are fixed-priority scheduling policies.
Hard Reserve	A reserve that will not be scheduled on depletion until they are replenished.
Hardware priority scheduling policy	An allocation policy in which the priority of a request for the backplane is determined by a hardware register on each card that plugs into the backplane. Presumably, the hardware priority value reflects the importance of the device that is connected to the adapter.
Highest Locker Priority	A data-sharing policy in which an action using a logical resource is executed at the highest priority of all actions that use the logical resource (i.e. at the priority ceiling of the resource). This protocol provides a good level of control over priority inversion.
Input Jitter	The deviation in the size of the interval between the arrival times of a periodic action.
Kernelized Monitor	A data-sharing policy in which an action using a logical resource is executed in non-preemptive fashion (i.e. at kernel priority). This protocol provides a good level of control over priority inversion except when one or more actions using a logical resource has a long execution time (relative to the timing

	constraints of other higher priority tasks).
Kernel Module	Synonymous with Loadable Kernel Module.
Loadable Kernel Module	A software subsystem that can be dynamically made part of the operating system kernel. With TimeSys Linux/RT, the RK and the RTAI components are provided in this form allowing the user to dynamically load or unload them while the Linux kernel is running.
Logical Resource	A system entity that is normally shared across multiple tasks. A logical resource must be bound to a physical resource like a processor, and is modeled in RMA as an action with a mutual exclusion requirement. Also, see Data-Sharing Policy.
Output Jitter	The deviation in the size of the interval between the completion times of a periodic action.
Period	The interarrival interval for a periodic event sequence. Also, see Input Jitter.
Periodic event	An event sequence with constant interarrival intervals. Described in terms of the period (the interarrival interval) and a phase value.
Predictable System	A system in which it is ***NOT*** possible to determine exactly what is or will be executing on the processor during system execution, given any specific time; however, it is possible to determine if deadlines associated with events can or can not be met.
Preemption	The act of a higher-priority process taking control of the processor from a lower-priority task.
Priority	Priority determines the execution characteristics of a thread under TimeSys Linux/RT. Priority is associated with each action under RMA, such that varying priorities within an event response is allowed.

Priority Ceiling	This is associated with each logical resource and corresponds to the priority of the highest priority action that uses the logical resource.
Priority Ceiling Protocol	A data-sharing policy in which an action using a logical resource can start only if its priority is higher than the priority ceilings of all logical resources locked by other responses. This protocol provides a good level of control over priority inversion.
Priority Inheritance Protocol	A data-sharing policy in which an action using a logical resource executes at the highest of its own priority or the highest priority of any action waiting to use this resource. This protocol provides an acceptable level of control over priority inversion.
Priority Inversion	This is said to occur when a higher priority action is forced to wait for the execution of a lower priority action. This is typically caused by the use of logical resources, which must be accessed mutually exclusively by different actions. Uncontrolled priority inversion can lead to timing constraints being violated at relatively low levels of resource utilization. Also see Blocking and Blocking Time.
Priority Levels	The number of distinct priorities available from the operating system.
Process	A collection of schedulable units of processing in the RK component of TimeSys Linux/RT, composed of one or more Threads.
Process Clone	Synonymous with Thread under TimeSys Linux/RT.
Rate-monotonic scheduling algorithm	Algorithm in which highest priority is assigned to the task with the highest rate (in other words, with the shortest period) and the priorities of the remaining tasks are assigned monotonically (or consistently) in order of their rates.

Rate-monotonic scheduling	A special case of fixed-priority scheduling that uses the rate of a periodic task as the basis for assigning priorities to periodic tasks. Tasks with higher rates are assigned higher priorities.
Real-time system	A system that controls an environment by receiving data, processing it, and taking action or returning results quickly enough to affect the functioning of the environment at that time.A system in which the definition of system correctness includes at least one requirement to respond to an event with a time limitation.
Reserve	Represents a share of a single computing resource. Such a resource can be CPU time, physical memory pages, a network bandwidth, or a disk bandwidth.
Resource	A physical entity such as a processor, a backplane bus, a network link or a network router which can be used by one or more actions. A resource may have a resource allocation policy (such as rate-monotonic scheduling) and a data-sharing policy.
Resource Set	Represents a set of reserves. A resource set is bound to one or more programs, and provides the exclusive use of its reserved amount of resources with those programs. A resource set groups necessary resources for the job of user applications; thus, it is easy to examine and compare the utilization of each resource in it.
Resource Kernel (RK)	A resource kernel is one which provides timely, guaranteed, and enforced access to system resources to applications.
Response	A time-ordered sequence of events arising from the same stimulus. In RMA, an event can trigger one or more actions to be exe-

	cuted.
Responses	Multiple time-ordered sequences of events, each arising from a distinct event. Event sequences that result in responses on the same resource often cause resource contention that must be managed through a resource allocation policy.
Soft Reserve	A reserve, that can be scheduled for execution on depletion along with other unreserved resource use and depleted reservations.
System Event	An Operating System generated event that is reported externally and can be monitored and visualized through TimeTrace, an Integrated Measurement and Visualization Environment from TimeSys.
Task	A schedulable unit of processing in the RTAI component of TimeSys Linux/RT, composed of one or more actions.
Thread	A schedulable unit of processing in the RK component of TimeSys Linux/RT composed of one or more actions. Synonymous with Process Clones.
TimeTrace	An Integrated Measurement and Visualization Environment from TimeSys that allows the visualization and reporting of fine-grained information in real-time systems.
TimeWiz	An Integrated Design Environment from TimeSys, that allows design, timing analysis and simulation of real-time Systems.
Tracer	A stimulus. Synonymous with a single instance of an "Event" within RMA, and is used to represent an end-to-end data flow sequence spanning multiple physical resources. An end-to-end timing constraint is normally associated with a tracer event. TimeWiz, an Integrated Design Environment for Real-

	Time Systems from TimeSys, computes both worst-case and average-case response times to a tracer using analysis and simulation respectively. Also see Trigger.
Trigger	A stimulus with an arrival pattern. Mostly synonymous with the term "Event" within RMA but is used to name an event whose response consists of a chain of actions executing on at most a single resource. In RMA, a trigger is bound to a (physical) resource when one or more actions in its corresponding response are bound to a (physical) resource. Also see Tracer.
User Event	An application generated event that is reported externally and can be monitored and visualized through TimeTrace, an Integrated Measurement and Visualization Environment from TimeSys.
Utilization	The ratio of a response's usage to its period, usually expressed as a percentage. For a CPU resource, this is execution time divided by period.
Worst-case response time	The maximum possible response time of a response's jobs (instances). Also, see Output Jitter.

Index