Linux/RK: A Portable Resource Kernel in Linux

Shuichi Oikawa and Ragunathan Rajkumar
Real-Time and Multimedia Laboratory
School of Computer Science
Carnegie Mellon University
5000 Forbes Avenue, Pittsburgh, PA 15213 USA
Email: {shui+,raj+}@cs.cmu.edu

1. Introduction

Linux/RK[1] is an implementation of a *resource kernel* [13, 8] based on Linux [3]. Linux/RK consists of two major components, the Linux kernel and a subsystem called the "Portable Resource Kernel". A resource kernel is one which provides timely, guaranteed and enforced access to physical resources for applications. 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 the 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 [15, 12].

Portability is another key issue in the development and deployment of resource kernels. One goal of our approach is the introduction of our resource kernel architecture to commodity operating systems. We chose Linux kernel as a basis of the first portable implementation because of its popularity and availability on a wide variety of platforms. Since Linux source code is freely available, we are able to investigate the support necessary from commodity operating systems with and without source code availability.

To implement the Portable RK within the Linux environment, we have explicitly tried to avoid making extensive modifications to the original Linux kernel. Thus, we have been developing Portable RK as an independent module from Linux kernel, and several *callback hooks* were introduced into the Linux kernel to combine it with Portable RK.

The implementation of Portable RK is ongoing, but has been completed to the extent that preliminary evaluation results can be included in this paper. These results are encouraging and stimulate the need for additional support.

[1] RK stands for a *Resource Kernel*.

1.1. Related Work

RT-Mach has been evolving from the original implementation [16], and now includes resource kernel support [8] that support reservations on multiple resource types. *CPU reserves* [5] were first introduced in RT-Mach, and network and disk bandwidth reservations are also available [2, 7]. Resource kernel functionality in RT-Mach is, however, tightly coupled with the microkernel. Portable RK is, by contrast, an independent kernel subsystem. It is modular and portable, so that it can be easily interfaced to commodity operating system kernels thereby making them resource kernels as well.

Real-Time Linux [1] and KURT [11] are real-time extended versions of Linux. Neither of them provides resource kernel features, such as guaranteed reservation and enforcement.

DREAMS [10] is a user-level real-time extension to Microsoft Windows NT [6]. It took a similar approach to ours for controlling applications' resource utilization, but its capability is very much limited because of its user-level implementation.

2. Software Architecture

In this section, the primary abstractions of a resource kernel, including a reserve and a resource set, are first introduced. Then, the architecture of Portable RK is described.

2.1. Abstractions

2.1.1. Reserves

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 the 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 be time-multiplexed or dedicated. Temporal resources 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: C, D, T, 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.

2.1.2. Resource Sets

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 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.

2.1.3. Reserve Types

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 undepleted 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.

2.2. Portable RK

There are two major components in Linux/RK, the Linux kernel and our Portable RK layer. The Linux kernel is the host operating system kernel of Portable RK. While the Linux kernel provides normal operating system functions, Portable RK augments those functions to make it a resource kernel. Portable RK accomplishes this by implementing reserves and resource sets, along with the necessary mechanisms for admission control, resource scheduling, resource usage accounting, and enforcement.

2.2.1. Callback Hooks

Callback hooks interface the Linux kernel to Portable RK in order to introduce resource kernel abstractions into the Linux kernel. Those hooks catch relevant scheduling points in the Linux kernel, and sends these events to Portable RK. Portable RK uses the well-defined functions in the Linux kernel to control kernel entities, such as processes and device drivers.

create	creates a reserve. Specifying a list of resource reservation parameters, both abstract and real reserve objects are created.
destroy	destroys a reserve. Both abstract and real reserve objects are destroyed.
start_account	starts the accounting on a reserve.
stop_account	stops the accounting on a reserve.
replenish	replenishes a reserve.
enforce	enforces a reserve.

Table 1. Interface provided by Abstract Reserves

2.2.2. Reserves: Abstract and Real

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 decoupled 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.

Table 1 shows the standard interface provided by abstract reserves.

2.2.3. Guarantee Mechanisms

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

- *Admission control*: Portable RK 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*: Portable RK 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*: Portable RK 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.

2.2.4. Resource Sets

A resource set is a container of reserves and 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 resource 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.

2.2.5. API

Portable RK provides an API (Application Program Interface) for use by user-level applications. User applications use this API to create reserves and resource sets, and then attach their processes to these resource sets.

3. Implementation

We now briefly describe the implementation of Linux/RK. Callback hooks, which were inserted in the Linux kernel to interface it with Portable RK, are described first. Then, the support of a timestamp counter and a high resolution timer is described. Such support is more important in resource kernels than in traditional kernels, since accurate time management is a key requirement in resource kernels. Finally, the use of the Proc filesystem is described. The support of the Proc file system makes it easier for applications to obtain information on the current statuses of kernel internals.

3.1. Callback Hooks

At this moment, the following callback hooks have been implemented in the Linux kernel:

- schedule callback hook: is within schedule(), the scheduling function of Linux kernel.
- interrupt callback hook: intercepts Linux interrupt handling.
- interrupt out callback hook: is used to notify Portable RK when processing an interrupt is finished.
- kernel out callback hook: is used to notify Portable RK when the execution is going back from kernel mode to the user level.

The implementation of callback hooks for controlling devices enables the management of non-CPU resources like disk bandwidth and network bandwidth, and is in progress.

3.2. Accurate Time Management

The combination of a timestamp counter and 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 Portable RK. The representation of time in Portable RK for use in accounting and scheduling is based on the values from this timestamp counter.

A high resolution timer [8, 11] 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. Portable RK sets the latch for the next interrupt every time after a timer interrupt occurs.

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

3.3. The *Proc* Filesystem

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

Linux/RK 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.

4. Evaluation Results

We now present some preliminary evaluation results on Linux/RK. All benchmarks were performed on a 200MHz Pentium Pro PC with 256KB L2 cache and 64MB of RAM, and the costs were measured using the built-in high resolution timestamp counter.

In the first experiment, a benchmark program ran with a processor reserve which reserved 200 msec of CPU time every 1 second. Figure 1 shows the CPU cycles actually consumed with and without the support of a high-resolution timer. Without high-resolution timer support, the program always overran for very close to 10 msec. On the other hand, with high-resolution timer support, the consumed CPU cycles are almost always 200 msec during every 1 second interval.

Figure 2 shows the effect of peripheral devices on the precision obtained with the high resolution timer. The benchmark program ran again, when network I/F devices were up and down, with a reserve of 200 msec CPU time every 1 sec. The system was being connected to the busy departmental LAN when network I/F devices are up. The differences are

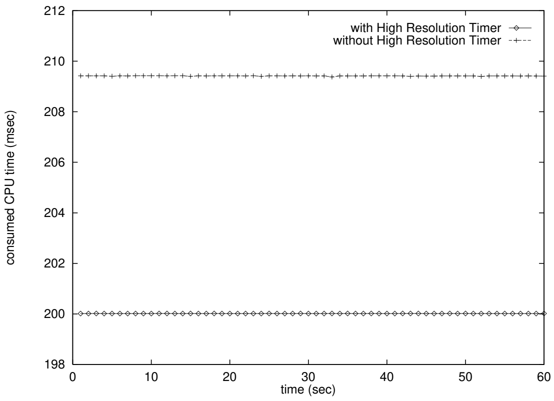

Figure 1. Enforcement of CPU Time Utilization with/without High Resolution Timer Support

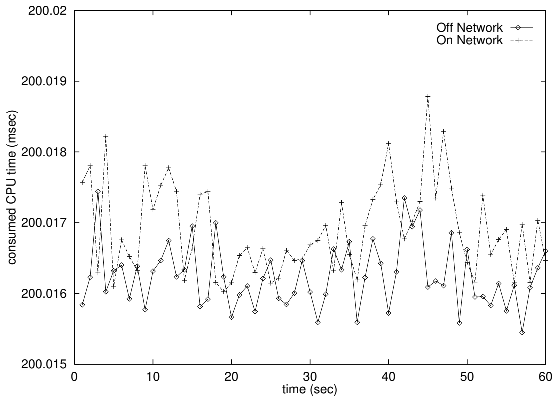

Figure 2. Enforcement of CPU Time Utilization with Network Up/Down

negligible although having no network connection seems to contribute to slightly better results.

5. Summary

This paper described "Portable Resource Kernel", a portable OS subsystem that can be slipped under commodity operating systems to make them resource kernels with support for timely, guaranteed and enforced resource accesses. We showed how Portable RK augments the functionality of the Linux kernel to make it into a resource kernel. Portable RK is an independent module which interfaces with the Linux kernel through callback hooks; thus, the modifications to the Linux kernel are minimal.

We are also currently investigating the use of Portable RK with other commercial operating systems, such as Windows NT and Windows CE, without access to source code.

References

[1] M. Barabanov and V. Yodaiken. Introducing Real-Time Linux. *Linux Journal*, Issue 34, Feburary 1997.

[2] C. Lee, K. Yoshida, and R. Rajkumar. Predictable Communication Protocol Processing in Real-Time Mach. In *Proceedings of IEEE Real-Time Technology and Applications Symposium*, June 1996.

[3] *http://www.linux.org/*

[4] C. L. Liu and J. W. Layland. Scheduling Algorithms for Multiprogramming in a Hard Real Time Environment. *JACM*, Vol. 20. No. 1, pp.46–61, 1973.

[5] C. W. Mercer, S. Savage, and H. Tokuda. Processor Capacity Reserves for Multimedia Operating Systems. Technical Report CMU-CS-93-157, School of Computer Science, Carnegie Mellon University, May 1993.

[6] *http://www.microsoft.com/*

[7] A. Molano, K. Juvva, and R. Rajkumar. Real-Time Filesystems: Guaranteeing Timing Constraints for Disk Accesses in RT-Mach. In *Proceedings of Real-Time Systems Symposium*, December 1997.

[8] R. Rajkumar, K. Juvva, A. Molano, and S. Oikawa. Resource Kernels: A Resource-Centric Approach to Real-Time Systems. In *Proceedings of the SPIE/ACM Conference on Multimedia Computing and Networking*, January 1998.

[9] L. Sha, R. Rajkumar, and J. P. Lehoczky. Priority Inheritance Protocols: An Approach to Real-Time Synchronization. *IEEE Transactions on Computers*, Vol. 39, No. 9, pp. 1175–1185, 1990.

[10] S. Sommer and J. Potter. Operating System Extensions for Dynamic Real-Time Applications. In *Proceedings of Real-Time Systems Symposium*, December 1996.

[11] B. Srinivasan, S. Pather, R. Hill, F. Ansari, and D. Niehaus. A Firm Real-Time System Implementation Using Commercial Off-The-Shelf Hardware and Free Software. In *Proceedings of IEEE Real-Time Technology and Applications Symposium*, June 1998.

[12] P. Steenkiste, A. Fisher, and H. Zhang. Darwin: Resource Management for Application-Aware Networks. Technical Report CMU-CS-97-195, School of Computer Science, Carnegie Mellon University, December 1997.

[13] S. Oikawa and R. Rajkumar. A Resource Centric Approach To Multimedia Operating Systems. IEEE Workshop on Resource Allocation Problems in Multimedia Systems, Washington D.C., December 1996.

[14] R. Rajkumar, C. Lee, J. P. Lehoczky, and D. P. Siewiorek. A QoS-based Resource Allocation Model In *Proceedings of the IEEE Real-Time Systems Symposium*. December, 1997.

[15] R. Rajkumar, C. Lee, J. P. Lehoczky, and D. P. Siewiorek. Practical Solutions for Some QoS-based Resource Allocation Problems In *Proceedings of the IEEE Real-Time Systems Symposium*. December, 1998.

[16] H. Tokuda, T. Nakajima, and P. Rao. Real-Time Mach: Towards a Predictable Real-Time System. In *Proceedings of USENIX Mach Workshop*, October 1990.