Temporal Protection in Real-Time Operating Systems

Cliff Mercer[1], Ragunathan Rajkumar[2] and Jim Zelenka[1]
[1]Department of Computer Science
[2]Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

Real-time systems manipulate data types with inherent timing constraints. Priority-based scheduling is a popular approach to build hard real-time systems, when the timing requirements, supported run-time configurations, and task sets are known *a priori*. Future real-time systems would need to support these hard real-time constraints but in addition (a) provide friendly user-interfaces with audio and video data types (b) be able to communicate with more global networks and systems on demand, and (c) support critical command and control services despite potential risks introduced by such added flexibility and dynamics. In this paper, we argue that temporal protection mechanisms can be as beneficial in these systems as virtual memory protection. The processor reservation mechanism that we have implemented in Real-Time Mach, for example, provides guaranteed timing behavior for critical activities.

1. Introduction

In real-time systems, the correctness of a computation depends upon both its logical and temporal correctness. As a result, earlier real-time systems were often hand-crafted in order to meet stringent timing constraints. Recently, more flexible priority-based scheduling approaches have become popular [5, 8, 15]. However, the design of such systems still requires *a priori* knowledge of tasks and their timing requirements, and therefore these systems tend to be very static in nature. Many recent trends indicate that future systems increasingly need to be much more dynamic in nature:

- Real-time applications are becoming more pervasive and complex. For example, the next generation of naval systems are expected to support many data types including analog, discrete, graphics, audio, video and voice, with integrated communications and control with low latency requirements. This trend has been accelerated by two related mainstream factors. First, the explosive surge of multimedia applications has literally brought time-critical data types (audio and video) to the desktop. Secondly, the continuing growth of computing power at ever falling prices enables more and more applications, with multitasking becoming a natural candidate to use up available cycles.

- The advent of high-performance networks such as ATM [13] opens up new applications with high bandwidth, low latency, and guaranteed service requirements. These applications include tele-medicine, distance learning, advanced air traffic control, sophisticated defense systems and networked patient monitoring. In these applications, any overload conditions because of dynamic requests and/or connections must not disrupt their basic mission.

- There is a rush towards universal connectivity and access, where a piece of information (or a person) is just a call away. Such high-degree of connectivity between systems and networks (both via cable and wireless networks) provides global access to information databases. Future systems would therefore be hard pressed not to exploit such information availability and reachability. In addition, such information accesses may need to be setup dynamically on user demand.

The above trends are expected to result in flexible, friendly, dynamic and informative systems but this flexibility and accessibility bring about new problems such as issues of security and privacy. In addition, real-time (such as multimedia) connections must be established online. These multimedia interfaces require real-time behavior but not at the cost of adversely affecting those critical activities with hard real-time constraints. In this paper, we argue that robust temporal protection mechanisms are needed to supplement and complement spatial protection mechanisms in order to achieve this goal.

2. Protection Mechanisms for Real-Time Programs

Early on, real-time systems were often built without virtual memory protection because the stochastic nature, long delays and potential overhead introduced by demand paging were considered incompatible with real-time requirements. However, processors and memory have become faster, and real-time operating systems [6, 14, 18, 19] have become more sophisticated. Many modern real-time operating systems (such as Real-Time Mach [19], LynxOS, etc.) allow virtual memory pages to be "locked" into physical memory. Any associated overhead is now considered to be worth the address space protection enforced across process boundaries. As a result, logical misbehavior on the part of one process (such as the use of incorrect address pointers) does not necessarily mean that an entire processor will fail. We refer to such protection as *spatial protection*.

Traditional non-real-time operating systems also provide a simple notion of *temporal protection* because of the fact that fairness is a primary motivation. The scheduler in time-sharing systems [7] typically uses a multi-level feedback queueing mechanism such that a process executing for a long time typically has its priority lowered in order to let other waiting processes execute. Hence, even a process which enters an infinite loop can normally be stopped or killed when its scheduled time quantum expires and another process is scheduled.

In real-time systems, where timeliness of critical activities (and not fairness) is the primary motivation, the issue of temporal protection needs to be substantially re-considered. For example, consider fixed priority scheduling approaches such as rate-monotonic scheduling [9, 17] to build real-time systems. Under a fixed priority scheduler, a process which enters an infinite loop will keep preempting all its lower priority tasks, which can then never run again. The only recourse is often to reboot the machine.

Spatial virtual memory protection has been adapted to real-time systems by providing the ability to lock down memory pages. Similarly, it is reasonable to expect that temporal protection schemes need to be adapted to real-time systems. We believe that such temporal protection mechanisms are critical to support the flexible and dynamic application environments described earlier. In the next section, we consider temporal protection schemes in detail.

2.1. Temporal Protection for Real-Time Programs

Many real-time operating systems support a multitasking environment due to its inherent modularity, ease of program development and debugging, and programming (as well as conceptual) compatibility with traditional operating systems such as Unix [3]. The timing behavior of a real-time process in these multitasking environments depends upon its own behavior *and* its level of resource-sharing with other processes. Resources shared by processes can be either physical or logical. Physical resources shared across processes include the CPU, buses, networks, memory pages, memory heap, I/O interfaces etc. Logical resources can include servers, shared queues, communication buffers etc.

Scheduling theory for processors [9], buses [16] and networks [1] provides the means to determine whether a set of tasks using a physical resource can meet its timing requirements. Similarly, synchronization protocols [15, 4] provide the ability to analyze the needs of real-time tasks to share logical resources. However, these analytical techniques must necessarily make assumptions such as the worst-case execution time of a task, the maximum duration of a critical section, or the maximum bus transaction time. If these assumptions are violated, undesirable consequences can occur. In static systems, it may be relatively easier to ensure at development time that these assumptions are indeed satisfied. However, as real-time systems and applications become more dynamic and flexible in nature, the robustness of the system's ability to deliver its critical functionality may be compromised by errors and violations in a relatively new and untested process.

3. Guaranteed Processor Reservation for Real-Time Programs

Processor and memory sharing are two critical pieces which can substantially affect (and dominate) the timing behavior of a real-time program. We have been investigating an operating system abstraction called *processor reserve* [10, 11] to provide temporal protection to a real-time process at the level of CPU sharing. In this abstraction, we view processor capacity as a quantifiable resource which can be reserved like physical memory or disk blocks. A processor reserve represents a claim on processor capacity over time (e.g. 10 *ms* of computation time out of each 50 *ms* of wall-clock time). An admission control policy determines whether a reservation request is accepted or not, and once the processor reservation is established, it is enforced by the operating system. Together, reservation and the enforcement mechanism provide a scheduling firewall which protects reserved programs from outside interference in much the same way as memory protection isolates a program address space from access by other programs.

Our processor reservation scheme has been implemented in Real-Time Mach [19]. Real-Time Mach supports a priority-driven paradigm to schedule real-time tasks[1]. The reservation scheme includes an admission control policy to prevent overload and a mechanism to accurately measure computation time consumed by programs. In addition to measuring computation time usage, the reservation mechanism enforces computation time limits over the short-term in order to ensure that a program which attempts to use more computation time than its allocation does not interfere with the timing behavior of other programs.

---

[1] Other CPU scheduling policies such as round-robin are also provided as dynamic configuration options.

3.0.1. Experimentation with Processor Reservation

We now describe an application built on our reservation scheme. The application consists of a number of instantiations of a QuickTime video player [20], each of which displays a video stream on the screen. Each program reads a short video clip and then begins to output frames to the screen using a memory-mapped frame buffer. The video resolution is 160x120 with 8 bits of color. The program applies a noise filter to each frame before it is displayed. By itself, one instantiation of the program can run at 23.2 frames per second on a 486-based machine.

When we run two instantiations of the program under a time-sharing policy, each program averages 11.6 frames per second. Under this policy, the programs get variable service: first one video stream gets preference from the time-sharing scheduler and then the other, and they alternate getting better and worse service as their priorities change in the multi-level feedback queue.

Our reservation system allows us to go further in controlling the timing execution behavior of these two programs. If we consider one of the programs as the "focus" in the same way a video teleconference has one video stream as the "focus," we can reserve more processor capacity for that stream at the expense of the second stream. For example, when we give one video stream a reservation of 80% of the processor and allow the other to consume the remaining processor capacity, we get 18.6 frames per second on the "focus" video stream and 4.2 frames per second on the other stream.

3.0.2. Processor Reservation Manager

Fine-grained feedback on performance and the status of the reservation can help the application adapt to its own behavior and to the behavior of other parts of the system. Also, reservations may be changed by forces external to the reserved program, and the program must be informed of the change so that it can adjust its behavior. A "reservation manager" that manages the reservations on a system based on user input via a reservation user interface might make such external reservation adjustments.

3.0.3. Processor Reservation in Distributed Systems

Processor capacity reserves can support reservation in distributed real-time systems by having each reserve contain reservations for various resources around the distributed system. Then messages containing requests for remote service will contain these "sub-reserves" which can be used to "charge" the remote service. Another aspect of reservation in distributed systems concerns the reservation of communications protocol processing on each of the hosts [12].

3.0.4. Other Applications of Processor Reservation

The temporal protection offered by processor reservation can be exploited to achieve other goals as well. Processor reservation provides a way of allocating processor time to the user-level schedulers enabled by scheduler activation types of interfaces [2]. A scheduling policy which allocates a percentage of the processor to various user-level scheduling domains alleviates the need for global scheduling decisions in such systems. Also, small reservations can be dedicated for critical activities like emergency system administration, and quick interactive response to operators in compute-bound mobile-computing platforms.

4. Conclusion

Real-time systems need predictable timing behavior, and predictability is often achieved by exploiting the *a priori* knowledge of supported system functionality. These systems therefore tend to be static in nature. However, due to recent trends towards multimedia applications, high-performance networking and wide connectivity, it can be expected that future real-time systems will support a highly dynamic mix of applications and connections. These flexible and dynamic systems can be susceptible to errors and misbehavior on the part of some task(s) and/or network/bus traffic. It is highly desirable that protection mechanisms be available in these systems to ensure that critical functionality is still provided by preventing temporal interference from other activities.

Address space protection offered by virtual memory is often desirable in today’s systems because of the logical fence that it builds between processes. The robustness provided by such "spatial protection" has led to its use in real-time systems as well. However, in order to avoid the adverse timing effects of demand paging in real-time systems, the ability to wire down memory pages becomes necessary. We argue that temporal protection mechanisms are also crucial to ensure that the timing behavior of critical real-time activities is not interfered with. We described an abstraction for temporal protection called the *processor reserve* which ensures that a real-time task is guaranteed a required fraction of the processor. This abstraction has been implemented in Real-Time Mach.

References

1. Agrawal, G., Chen, B., Zhao, W., Davari, S. "Guaranteeing Synchronous Message Deadlines in High Speed Token Ring Networks with Timed Token Protocol". *Proceedings of IEEE International Conference on Distributed Computing Systems* (1992).

2. T. E. Anderson, B. N. Bershad, E. D. Lazowska and H. M. Levy. "Scheduler Activations: Effective Kernel Support for the User-Level Management of Parallelism". *ACM Transactions on Computer Systems 10*, 1 (Feb 1992), 53-79.

3. M. J. Bach. *The Design of the UNIX Operating System.* Prentice-Hall, 1986.

4. Baker, T. "Stack-Based Scheduling of Realtime Processes". *Journal of Real-Time Systems 3,* 1 (March 1991), 67--100.

5. Burns, A. "Scheduling Hard Real-Time Systems: A Review". *Software Engineering Journal* (May 1991), 116-128.

6. K. Jeffay, D. L. Stone and F. D. Smith. "Kernel Support for Live Digital Audio and Video". *Computer Communications (UK) 15*, 6 (July-August 1992), 388-395.

7. Leffler, S. J., McKusick, M. K., Karels, M. J. and Quarterman, J. S. *The Design and Implementation of the 4.3BSD UNIX Operating System.* Addison-Wesley, 1989.

8. Lehoczky, J. P. "Fixed Priority Scheduling of Periodic Task Sets with Arbitrary Deadlines". *IEEE Real-Time Systems Symposium* (Dec. 1990).

9. Liu, C. L. and Layland J. W. "Scheduling Algorithms for Multiprogramming in a Hard Real Time Environment". *JACM 20 (1)* (1973), 46 - 61.

10. C. W. Mercer and S. Savage and H. Tokuda. Processor Capacity Reserves for Multimedia Operating Systems. Tech. Rept. CMU-CS-93-157, School of Computer Science, Carnegie Mellon University, May, 1993.

11. C. W. Mercer and S. Savage and H. Tokuda. Processor Capacity Reserves: An Abstraction for Managing Processor Usage. Proceedings of the Fourth Workshop on Workstation Operating Systems (WWOS-IV), Oct, 1993.

12. C. W. Mercer, J. Zelenka and R. Rajkumar. "Resource Reserves for Operating System Protocol Processing". *Submitted for publication* (January 1994).

13. S. E. Minzer. "Broadband ISDN and Asynchronous Transfer Mode (ATM)". *IEEE Communications Magazine 27*, 9 (Sep 1989), 17-24.

14. *IEEE Standard P1003.4 (Real-time extensions to POSIX).* IEEE, 345 East 47th St., New York, NY 10017, 1991.

15. Sha, L., Rajkumar, R. and Lehoczky, J. P. "Priority Inheritance Protocols: An Approach to Real-Time Synchronization". *IEEE Transactions on Computers* (September 1990), 1175-1185.

16. Sha, L., Rajkumar, R. and Lehoczky, J. P. "Real-Time Scheduling Support in Futurebus+". *IEEE Real-Time Systems Symposium* (Dec. 1990).

17. Sha, L., Rajkumar, R. and Sathaye, S. "Generalized Rate-Monotonic Scheduling Theory: A Framework for Developing Real-Time Systems". *IEEE Proceedings Journal* (January 1994).

18. Stankovic, J. A. and Ramamritham, K. The Design of the Spring Kernel. Proceedings of the Real-Time Systems Symposium, Dec, 1987.

19. H. Tokuda and T. Nakajima and P. Rao. Real-Time Mach: Towards a Predictable Real-Time System. Proceedings of USENIX Mach Workshop, Oct, 1990.

20. H. Tokuda and T. Kitayama. Dynamic QOS Control Based on Real-Time Threads. ART Project Technical Memo, School of Computer Science, Carnegie Mellon University.

Table of Contents