A Rate-Based Execution Abstraction For Multimedia Computing*

Kevin Jeffay, David Bennett
University of North Carolina at Chapel Hill
Department of Computer Science
Chapel Hill, NC 27599-3175 USA
{jeffay,bennettd}@cs.unc.edu

Abstract: Process models for multimedia computing must allow applications to adapt their pattern of execution as resources become scarce or abundant. As processes adapt, it is natural to express their desired performance in terms of a processing rate of application-defined data units or events. We propose a process model wherein processes execute according to a general rate specification of *x* process executions every *y* time units. In addition, a separate parameter is used to specify the desired response time for the completion of each execution. In all cases the real-time performance of a rate-based process is predictable. The model is general enough to encompass or extend many of the existing models proposed for multimedia systems. Our model of rate-based execution is described along with an implementation that detects when processes should adapt their execution rate and minimizes latency in interprocess communication.

1 . Introduction

To be effective, distributed multimedia applications require operating system support to ensure real-time, low latency acquisition, processing, delivery, and playout of audio and video streams. Within the domain of process management, numerous proposals have been made for thread and process models, scheduling algorithms, and IPC mechanisms for supporting multimedia computing [1-8]. From these discussions, several themes and preliminary requirements have emerged. These include:

- *Deterministic execution*. At some level, services must be provided for applications to execute in a predictable manner. This is typically manifested in guarantees of bounded response time to events (*e.g.*, interrupts, message arrivals), or in some form of periodic execution [2, 4, 5].

- *Adaptive resource management*. Processing and communications resources can become saturated in the face of the demands of multimedia applications. Mechanisms for allocating resources to processes must allow applications to dynamically alter their desired service parameters to optimize their execution for the current perceived state of the system [2, 6, 7, 8, 13]. (An implicit assumption here is that applications can provide acceptable user performance across a range of resource requirements.)

- *Feedback on actual performance*. To best adapt to changing execution conditions, applications must be able to determine their actual execution performance in terms of desired service parameters. Operating systems must

* This work supported by grants from the Intel and IBM corporations.

In: Proceedings of the Fifth International Workshop on Network and Operating System Support for Digital Audio and Video, Durham, NH, April 1995, published in *Lecture Notes in Computer Science*, T.D.C. Little and R. Gusella, editors, Volume 1018, pages 64-75, Springer-Verlag, Heidelberg, Germany, 1995.

support a mechanism for an application-kernel dialog on performance parameters [2, 6, 7, 8].

- *Integrated resource allocation and IPC*. The real-time performance of an application should not be a function of its process structure. In principle, the guaranteed real-time performance of an application should be the same when the application is structured as a set of communicating sequential processes as when it is a single monolithic process. This implies that when processes communicate, either on a peer-to-peer or client-server basis, the desired service parameters of the initiator should be propagated to and inherited by, the recipient [1, 16]. This is to ensure, for example, that priority inversions do not occur in interprocess communication.

- *High-level specification of service parameters*. There is a general desire to allow applications to specify desired service parameters in a context that is removed from low-level operating system mechanisms (*e.g.*, priority) and ideally tied to application or system-level concepts such as a *capacity*, *utility* or *added value* function for real-time execution [2, 6].

In this note we describe a “new” process model based on the concept of *rate-based execution* (*RBE*) that addresses each of these requirements. In an *RBE* system, processes specify their desired rate of progress in terms of the number of events they desire to process in an interval of specified duration. Examples of rate specifications include processing two audio frames every 16.6 ms, one video frame every 33 ms, or 30 packets every second. Our *RBE* model is based on a formal resource allocation model that indicates both on-line scheduling and admission control algorithms to use. A process in an *RBE* system is guaranteed to make progress at at least its specified rate whenever work exists for the process. When insufficient resource capacity exists to admit an *RBE* process, the process can either negotiate with the operating system for a reduced rate of progress or wait for sufficient resources to become available. Resources can be made available in two ways. A user can explicitly reduce their performance expectations for an application and request an application to scale back its resource usage (*e.g.*, lower the frame rate of a videoconference). Alternatively, the operating system can request that a process reduce its resource requirements when the system perceives that the process is continually using less resources than they reserved.

We conjecture that the notion of processing rate is a natural and useful abstraction for multimedia systems. Our particular *RBE* model is general enough to encompass the majority of the traditional real-time task models based on periodic or sporadic tasks, as well as several of the models previously proposed for multimedia computing. Moreover, it also allows the specification and integration of soft- and non-real-time activities with hard-real-time activities. The model has been implemented in an experimental microkernel and is presently being ported to the Real-Time Mach kernel. Initial experiences with *RBE* indicate that it is easy to support and provides a simple but effective means for applications to communicate, monitor, and adapt their desired service. While not a panacea, we believe the *RBE* process model is a simple yet powerful paradigm of process execution that is suitable for generic real-time computing in general, and multimedia computing in particular.

In the following, we provide an overview of the *RBE* abstraction and briefly compare it to existing real-time and multimedia process models. We then discuss how *RBE* processes can be used to manage latency in applications and how *RBE* processes can adapt their execution rates as resources become scarce or abundant.

2. The Rate-Based Execution Concept

Our formulation of rate-based execution is designed to integrate three paradigms of repetitive real-time execution. The first is the concept of *software phased-locked loops* as embodied in the Synthesis Kernel [10]. Here the operating system monitors each process's event queue and uses a control theoretic scheduling policy to schedule the processes so as to minimize their event queue length. This mechanism has been demonstrated to execute processes in soft-real-time. Execution is soft-real-time in the sense that all resource allocation is best-effort and no guarantees of performance are possible. The second paradigm is the *linear-bounded arrival process* (*LBAP*) model as defined in the DASH system [5]. Here processes specify a desired execution rate as the number of messages to be processed per second and the size of a buffer pool used to store bursts of messages that arrive for the process. When *LBAPs* are independent it is possible to state conditions under which a collection of *LBAP*s will execute at their desired rate. The third paradigm is the *real-time producer/consumer* (*RTP/C*) process model used in the YARTOS kernel [11]. Here applications are expressed as networks of processes that execute in response to message arrivals. The arrival rate of messages is assumed to be uniform in that the difference between the expected message interarrival time and the minimum interarrival time is small. When this assumption holds it is possible to state conditions under which *RTP/C* processes will execute in real-time (each process will process each message before the next one arrives).

Our model of rate-based execution takes as its starting point the linear-bounded arrival process model and generalizes it to include a more generic specification of rate and adds an independent response time (deadline) parameter to enable more precise real-time control of the execution of *RBE* processes. An *RBE* process is described by three parameters:

- two rate parameters x and y, where x is a number of events to be processed and y is the length of an interval in which x events are expected to arrive.

- a response time parameter d which specifies the desired maximum elapsed time between the delivery of an event to a process and the completion of the processing of the event.

Informally, an *RBE* process (x, y, d) will execute at a rate sufficient to ensure that x events are consumed every y time units and that each event is processed within d time units after its arrival whenever possible. The actual execution rate of an *RBE* process is determined by the rate at which events are actually generated for the process. For example, if events are generated at the rate of no more than one every x/y time units or no more than x events are generated in any interval of length y then every event will be processed within d time units after its arrival. If events are generated at a higher rate (*e.g.*, an unanticipated burst of work arrives), then in the worst case these events will be processed as if they had arrived at the desired rate. (As is the case with unexpected bursts of work for *LBAP*s in DASH.)

More precisely, for all $j \geq 1$, if t_j is the arrival time of the j^{th} event for an *RBE* process, then that event is guaranteed to be processed before a deadline of time $T(j)$, where

$$T(j) = \begin{cases} t_j + d & \text{if } 1 \leq j \leq x \\ MAX(t_j + d, T(j - x) + y) & \text{if } j > x \end{cases}$$

If events are generated at the rate of *x* events every *y* time units, then every event will be processed within *d* time units after its arrival. If events arrive at a faster rate then their guaranteed completion time is based on that for the *x* previous events and not (directly) on the parameter *d*.

We have developed a general theory of rate-based execution that gives conditions under which a set of *RBE* processes are guaranteed to execute in real-time [14]. Real-time here means that the j^{th} event for an *RBE* process is guaranteed to be processed before time $T(j)$. The theory is used both by the operating system for admission control and by a programmer to understand which rate specifications are more likely to result in an admissible process when resources are scarce.

Beyond *LBAP*s, our model of rate-based execution is general enough to encompass traditional real-time process models. For example, when $x = 1$, events are separated in time by exactly *y* time units, and $d = y$, then the above reduces to the commonly held definition of a periodic task [9]. The primary distinction between a rate-based process and a periodic one is that in the rate-based case, no assumptions about the interarrival times of events are made or required (however, in any implementation of an *RBE* process, the set of allowable execution rates for a process would be limited by the number of event buffers available to the process). A secondary distinction between the two models is that for a rate-based process, no assumption is made about the distribution of processes' execution time within an interval of length *y*.

RBE processes can be used to emulate software phased-locked loops, however this is less a function of the *RBE* model *per se* and more a function of the implementation of the model. This is described next.

3. Supporting Rate-Based Execution

We have implemented *RBE* processes in the YARTOS (*Yet Another Real-Time Operating System*) kernel [4]. Here we describe three interesting aspects of the implementation: admission control, the use of *RBE* processes to manage end-to-end latency, and the paradigms and mechanisms used for adapting execution rate.

Programming Model and Admission Control

YARTOS supports a simple data-flow model of computation. Processes receive events, process them, and generate events for other processes. We distinguish between two types of *RBE* processes in YARTOS: *internal* and *source* processes. A source *RBE* process is a process that receives events directly from an external device.

Its rate parameters are the expected rate at which the device will generate work and are specified when the process is created. An internal *RBE* process is a process that receives events generated by other *RBE* processes. For example, a server process would be an internal *RBE* process. Internal *RBE* processes are created without a rate specification and at run-time (transitively) inherit the rate specification of their event generators. That is, at an internal process, the deadline for processing each event is computed using the rate specification of the event's generator. This inheritance of rate parameters is done to ensure that all internally generated events (and by transitivity, all externally generated events) are processed at the rates at which they are generated.

A source *RBE* process is created by specifying:

- a program to execute in response to event arrivals,
- an event type — an input descriptor (port) specifying the logical device that generate events for this process,
- a rate specification — the parameters *x*, *y*, and *d*,
- an optional event queue length — the number of buffers allocated for events,
- an execution cost — the estimated worst case execution time for processing an event,
- a rate adaptation callback — a function called by the kernel to suggest that the process either speed up or slow down.

There is an event queue for each source process that contains events to be processed. By default, for a process that executes at the rate of *x* events every *y* time units, the queue will have *x* entries. If events for this process are never generated at a higher rate than *x* events every *y* time units, then events will never be lost. If event generators (devices) are expected to be ill-behaved, then larger queues can be specified.

An internal *RBE* process is created by specifying:

- a program to execute in response to event arrivals,
- an event type — an input descriptor (port) specifying the other *RBE* process(es) that generate events for this process,
- an execution cost — the estimated worst case execution time for processing an event.

The port descriptors of processes are used by the kernel to construct a directed graph of interprocess communication that is used for admission control. When a new process is created a topological sort is performed on the graph and each internal process (an internal node in the graph) is assigned a rate equal to the sum of the rates of all its immediate predecessors in the graph. The resulting set of rate specifications along with the execution cost parameters of all processes are then used as input to the schedulability test described in [14]. If the result of the test is positive then the *RBE* process is created. If the result is negative then the creator must either reduce the rate specification (in the case of a source *RBE* process) or reduce the rate specification of a source *RBE* process that will generate events for the process to be created.

In general, two useful strategies for gaining admission of a rejected process are to (1) relax (lengthen) the response time parameter of either the process itself (for source processes), or that of a source *RBE* process that will generate events for the rejected process, or to (2) reduce execution rate of either the process or its event generators. When a process is rejected, the kernel can provide hints to the calling process in the form of possible reduced rates or relaxed response times.

Finally, we support non-real-time processing by allowing processes to be created with event queue lengths of zero. In this case the system will enqueue events for the process on a space-available basis and schedule the process so as to minimize its event queue length. As the event queue grows for an adaptive *RBE* process, the system increases the rate at which the process executes (if possible); as the queue shrinks, the rate is decreased. The size and frequency of rate manipulations is controlled by a simple control theoretic model borrowed from the Synthesis Kernel [10].

Minimizing Response Time

A second benefit to having internal processes inherit the rate specifications of their event generators is that it enables the kernel to minimize its guarantees of worst case response time for the processing of events that propagate through a chain of *RBE* processes.

Response time guarantees for event processing are based on the response time parameters of source *RBE* processes and are contingent upon the accuracy of the rate specifications of these processes. If a source *RBE* process *S* is admitted into the system with rate and response time parameters (x, y, d), then so long as events arrive at *S* no faster than x events every y time units, each event is guaranteed to be processed by *S* within d time units of its arrival.

The response time guarantees for internal *RBE* processes are determined as follows. If whenever *S* receives an event e, it generates an event e' for an internal *RBE* process *I*, then the processing of events e and e' is guaranteed to be completed within d time units of the arrival of e at *S*. That is, the worst case guaranteed response time for processing events that propagate through processes *S* and *I* is the same as the worst case guaranteed response time would be at a process S' that combined the functions of processes *S* and *I* and had *RBE* parameters (x, y, d). Thus, the guaranteed response time for an event that propagates through *S* and *I* is given by the response time parameter of *S*. In the general case, the worst case guaranteed response time for an event that propagates through a chain of *RBE* processes is not a function of the length of the chain; only of the response time parameter of the source *RBE* process at the head of the chain.

Note that strictly speaking, this technique of minimizing response time is orthogonal to the concept of rate-based execution. It has been applied to other non-rate-based systems [1, 16] and is related to the concept of priority inheritance. We have demonstrated, however, that it is also applicable in a rate-based framework. Minimizing response time is particularly important in a rate-based system as it is important to be able to control throughput and response time independently.

Adapting Execution Rate

Once admitted, *RBE* processes are guaranteed to execute at a rate that is sufficient to process their events in real-time. Here the interpretation of "real-time" is ultimately given by a set of rate specifications that describe the expected behavior of processes external to the computer. Whenever the external environment deviates significantly from these rate specifications, source *RBE* processes should adapt their execution rates to reflect the changes in the environment.

If work is being generated faster than a source *RBE* process's rate specification, its event queue will overflow and events will be lost. When an event queue overflows the system can use the source process's callback function to suggest that the process increase its execution rate. Rate adjustments are performed by the affected processes since the interpretation of a rate for each process is application dependent. Rate decreases are always accepted (provided that the response time parameter is not reduced), however, rate increases are subject to the admissibility test and thus are not guaranteed to succeed.

The system can monitor the execution rate of a source process by simply keeping a count of the number of times the process has been scheduled in the recent past (a system parameter). Whenever an *RBE* process cannot be admitted into the system, the kernel can check if there exists a source process with a rate specification that is higher than the rate at which work is currently being generated for the process (*i.e.*, the process is executing below its rate specification because of a lack of events). The process can be contacted and requested to reduce its rate specification. Since internal *RBE* processes are reactive, no mechanism is needed to adapt their execution rate.

These rate re-negotiations naturally handle the case where processes need to adapt the rate of input processing. A slightly different form of rate adaptation is required when output cannot be performed at a fast enough rate to satisfy *RBE* processes. For example, consider a videoconferencing system. At some point in time video frames may be generated by a camera at a faster rate than they can be transmitted across the network because of network congestion. The symptom of such a problem is the same as in the input case — a queue overflows — however, because of the way output must be handled in a real-time system, the queue is not an event queue.

In a predictable real-time system, it is not possible, in general, to synchronize individual output devices with input devices. For example, a video digitizer for NTSC video will generate one video frame every 33 ms. For an application such as videoconferencing, ideally one would like to transmit 1 video frame every 33 ms. However, on current local-area networks, one cannot guarantee that this will be possible at all times. Thus, given the variability of sustainable network transmission rates and the determinism of the NTSC video generation process, it is not possible to synchronize the camera with the network so that video frames are transmitted as they are generated. That is, events generated by the camera cannot be guaranteed to be processed by the network in real-time. Therefore, events generated by the camera cannot be treated as events for a network process.

Because of this (generic) inability to synchronize input and output devices, somewhere between the camera and the network there must exist a set of buffers that are written by a process whose execution rate is derived from events generated from the camera/digitizer (*e.g.*, vertical blanking interrupts) and read by processes whose execution rate is derived from events generated by the network (*e.g.*, transmission complete interrupts). (Operations on this buffer must be non-blocking to ensure response time guarantees are met.) It is this buffer that overflows when frames are produced faster than they can transmitted. Since this buffer cannot serve as the event queue for a network process, a separate mechanism is required to detect and respond to overflows of this buffer.

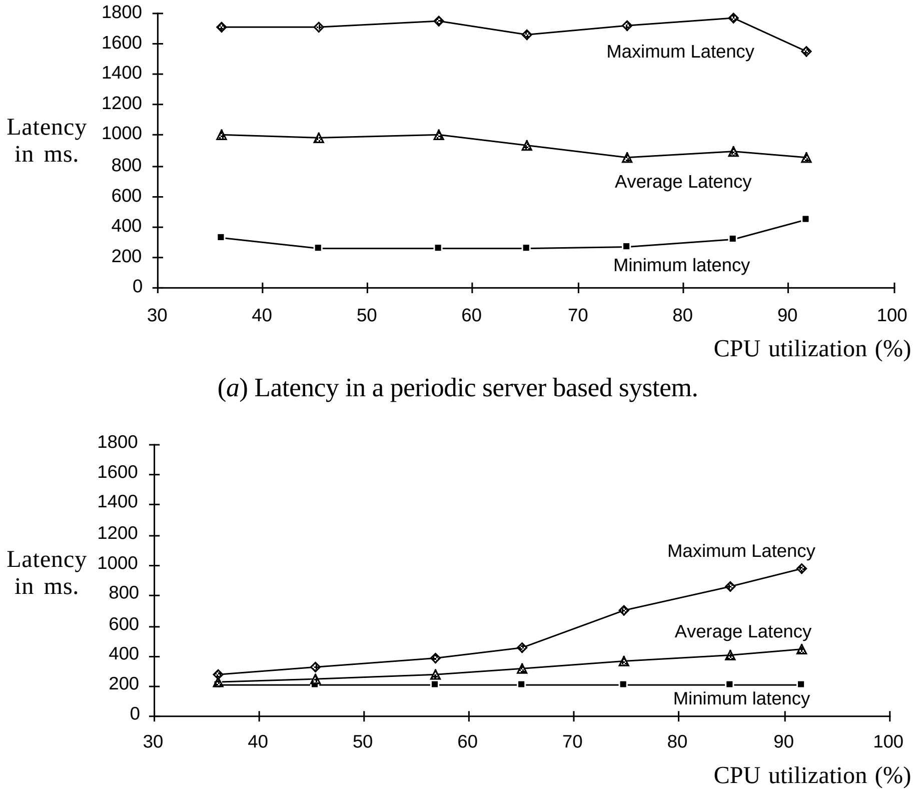
In general it is difficult to systematize these interactions. The buffer can either be constructed to return its current length when items are deposited or removed, in which case processes can learn of rate mis-matches themselves, or system processes (*e.g.*, device drivers) can be designed to report buffer overflows to the operating system which can in turn notify the appropriate processes.

Finally, applications themselves may choose to adapt their execution rates on their own. For example, a videoconferencing application that is able to sustain a sub-optimal frame transmission rate may attempt to increase the frame rate in hopes of producing a higher quality conference. Similarly, the application may receive application-level feedback from a conference receiver that data is being lost in the network and thus the application may elect to reduce its frame transmission rate [12].

4. Using Rate-Based Execution

Our *RBE* system has been used to re-implement a desktop videoconferencing system [12, 13]. The original system used a more traditional hard-real-time process model based on sporadic tasks [4]. In that system, events were required to have a minimum inter-arrival time in order to guarantee real-time performance. Since conferences spanned internetworks made up of existing local-area networks, it was not possible in general to guarantee that network packets containing conference data arrived at a receiver with a non-zero minimum interarrival time. Therefore, to ensure that packet processing did not consume all available processing cycles, the system effectively polled the network interface for packet arrivals using a variation of the periodic server concept [17].

There were two immediate benefits to using *RBE* processes in this application. First, the *RBE* system resulted in substantially lower latency media playout in times of severe network congestion. Figure 1 compares the latency of media playout in a simulated conference using *RBE* processes to read data from the network interface versus a periodic server. A trace of packet arrival times was recorded during an actual conference and used to simulate the arrival of events at conference receiver based on *RBE* processes and one using a periodic server and sporadic application tasks. Both systems were executed under a variety of system loads. The conference processes in both cases consumed approximately 25% of the processor. In the *RBE* system, the rate specification for the network's source process was 10 arrivals every 550 ms with a response time parameter of 550 ms. In the periodic server based system, the server executed with a period of 55 ms. No rate adaptation in the *RBE* system, and no period adaptation in the periodic server based system was used.



(*a*) Latency in a periodic server based system.

(*b*) Latency under rate-based execution.

Figure 1

The performance of the periodic server based system was consistent across system loads. Average latency was approximately 1 second with significant deviation. The relatively high latency is due to the fact that a periodic server can only serve its event queue at a constant rate. Thus, once a queue builds up, if work continues to arrive at the expected rate then the server can never work off its backlog of work. In this environment, there were several periods of bursty arrivals that led to the formation of a queue of packets at the server. In fact, the queue overflowed on several occasions resulting in approximately 4% of all packets being lost.

In the *RBE* system, the average latency was well below the desired response time of 550 ms, however, as the CPU utilization reached approximately 70%, arrival rates that exceeded the rate specification caused some events to be processed with latencies of close to 1 second. No packets were lost in the RBE system.

In the periodic server based system, the processing of events that arrive in a burst are spread out over time as if the events arrived periodically. In contrast, the *RBE* system schedules all events (assigns them deadlines) when they arrive. Thus, it is possible for events with deadlines far off in the future to actually be processed well before the time at which a periodic server would first see the event.

The second benefit of using *RBE* processes was a less pessimistic admission control policy. Typically, to achieve low response times for events one must configure a periodic server to execute at a higher rate than work is expected to arrive. The excess CPU capacity that is reserved for the server but rarely used, cannot be allocated to other processes. In the *RBE* system, by specifying the desired aggregate processing rate, a process reserves only the capacity that it expects to actually use.

5. Related Work

Our model of rate-based execution borrows concepts from several other systems including, RT-Mach [1, 2, 7], DASH [5], Synthesis [10], and Concord [15]. Thus, while our system is an amalgam of others, we believe our primary contribution to be a demonstration that *RBE* processes are a simple and effective primitive for constructing multimedia systems and conjecture that the same holds for generic real-time applications. The *RBE* model encompasses those in each of the related works. Moreover, it has a formal underlying model of resource allocation that accommodates interprocess communication and synchronization.

We also believe the *RBE* to be a more natural and easy to use abstraction than those previously proposed. For example, the *reserve* abstraction for a task proposed by Mercer *et al*. [2], is essentially a specification of the processor utilization of a task. We claim that unlike the concept of execution rate, the notion of a reserve is inherently not a natural (or portable across hardware platforms) application-level concept.

Our work represents a counter-point to the work on *imprecise computation* by Lin *et al.* [15]. The imprecise model of computation was developed to primarily deal with the case of insufficient processing resources. The model provides a framework for real-time tasks to dynamically control which sections of their code are executed when the system is overloaded. By executing only fragments of the complete task, an “imprecise,” but nonetheless useful, partial result can be obtained in a statically defined interval. Rather than directly controlling the execution of task code, an *RBE* system manipulates the interval in which task executes.

6. Summary and Conclusions

Process models for multimedia computing must allow applications to adapt their pattern of execution to make the best use of the resources currently available to them while not sacrificing their ability to execute predictably in time. Moreover, applications should be able to express their desired performance in terms of application-defined data units or events. We have proposed a new process model wherein processes execute according to a general rate specification of x process executions every y time units. In addition, a separate parameter is used to specify the desired response time for the completion of each execution. The model is general enough to encompass or extend many of the existing models proposed for multimedia systems.

We have implemented a prototype of processes that adhere to this rate-based execution (*RBE*) model. In the implementation, the system identifies processes whose input events are arriving at a faster or slower rate than expected and requests these processes to modify their rate specification. Moreover, interprocess communication is implemented so that the response time guaranteed for processing an event in the worst case is not a function of the number of processes involved in processing the event.

Preliminary experiences with rate-based execution in the YARTOS kernel indicate that an *RBE* rate specification provides a convenient means of expressing the desired performance of a videoconferencing system and allows the system to easily adapt its execution in response to changes in its environment such as changes in network congestion. We conjecture that the same will hold for other multimedia applications. A comparison of the performance of *RBE* processes and traditional periodic processes shows that the *RBE* model offers significant advantages in terms of both accuracy of admission control and average and worst case response time for the processing of events.

We presently have a primitive rate-based process model implemented in the Real-Time Mach kernel and are working on expanding the implementation and comparing *RBE* processes (threads in Mach) to the real-time threads already present in Mach in an effort to confirm our initial findings of *RBE* performance.

7. References

[1] *Integrated Management of Priority Inversion in Real-Time Mach*, Nakajima, T., Kitayama, T., Arakawa, H., Tokuda, H., Proc. 14th IEEE Real-Time Systems Symp., Durham, NC, December 1993, pp. 120-130.

[2] *Processor Capacity Reserves: Operating System Support for Multimedia Applications*, Mercer, C.W., Savage, S., Tokuda, H., IEEE Intl. Conf. on Multimedia Computing and Systems, Boston, MA, May 1994.

[3] *Scheduling and IPC Mechanisms for Continuous Media*, Govindan, R., Anderson, D.P., Proc. ACM Symp. on Operating Systems Principles, ACM Operating Systems Review, Vol. 25, No. 5, October 1991, pp. 68-80.

[4] *Kernel Support for Live Digital Audio and Video*, K. Jeffay, D.L. Stone, F.D. Smith, *Computer Communications*, Vol. 15, No. 6, (July/August 1992) pp. 388-395.

[5] *Support for Continuous Media in the DASH System*, Anderson, D.P., Tzou, S.-Y., Wahbe, R., Govindan, R., Andrews, M., Proc. Tenth Intl. Conf. on Distributed Computing Systems, Paris, France, May 1990, pp. 54-61.

[6] *Adaptive Real-Time Resource Management Supporting Modular Composition of Digital Multimedia Services*, M.B. Jones, in Network and Operating System Support for Digital Audio and Video, Proceedings, Fourth Intl. Workshop, Lancaster, UK, November 1993, D. Shepherd, *et al.* (Eds.). Lecture Notes in Computer Science, Vol. 846, pp. 21-28, Springer-Verlag, Heidelberg, 1994.

[7] *Dynamic QOS Control Based on Real-Time Threads*, H. Tokuda, T. Kitayama, in Network and Operating System Support for Digital Audio and Video, Proceedings, Fourth Intl. Workshop, Lancaster, UK, November 1993, D.

Shepherd, *et al.* (Eds.). Lecture Notes in Computer Science, Vol. 846, pp. 124-137, Springer-Verlag, Heidelberg, 1994.

[8] *System Support for Time-Critical Applications*, J.D. Northcutt, E.M. Kuerner, in Network and Operating System Support for Digital Audio and Video, Proceedings, Second Intl. Workshop, Heidelberg, Germany, November 1992, R.G. Herrtwich (Ed.). Lecture Notes in Computer Science, Vol. 614, pp. 242-254, Springer-Verlag, Heidelberg, 1992.

[9] *Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment*, Liu, C.L., Layland, J.W., Journal of the ACM, Vol. 20, No. 1, (January 1973), pp. 46-61.

[10] *Fine-Grain Adaptive Scheduling Using Feedback*, H. Massalin, C. Pu, *Computing Systems*, Vol. 3, No. 1, (Winter 1990) pp. 139-173.

[11] *The Real-Time Producer/Consumer Paradigm: A paradigm for the construction of efficient, predictable real-time systems*, K. Jeffay, Proc. 1993 ACM/SIGAPP Symposium on Applied Computing, Indianapolis, IN, ACM Press, February 1993, pages 796-804.

[12] *Two-Dimensional Scaling Techniques For Adaptive, Rate-Based Transmission Control of Live Audio and Video Streams*, T.M. Talley, K. Jeffay, Proc. Second ACM International Conference on Multimedia, San Francisco, CA, October 1994, pp. 247-254.

[13] *Transport and Display Mechanisms For Multimedia Conferencing Across Packet-Switched Networks*, K. Jeffay, D.L. Stone, F.D. Smith, *Computer Networks and ISDN Systems*, Vol. 26, No. 10, (July 1994) pp. 1281-1304.

[14] *A Theory of Rate-Based Scheduling*, K. Jeffay, University of North Carolina, Department of Computer Science, Technical Report, *in submission*.

[15] *Imprecise Results: Utilizing Partial Computations in Real-Time Systems*, Lin, K.-J., Natarajan, S., Liu, J.W.-S., Proc. of the Eighth IEEE Real-Time Systems Symp., San Jose, CA, December 1987, pp. 210-217.

[16] *On Latency Management in Time-Shared Operating Systems*, K. Jeffay, Proc. 11th IEEE Workshop on Real-Time Operating Systems and Software, Seattle, WA, May 1994, pp. 86-90.

[17] *Enhanced Aperiodic Responsiveness in Hard Real-Time Environments*, J.P. Lehoczky, L. Sha, J.K. Strosnider, Proc. of the Eighth IEEE Real-Time Systems Symp., San Jose, CA, December 1987, pp. 261-270.