Realizing a Soft Real-Time Framework for Supporting Distributed Multimedia Applications

Changpeng Fan

German National Research Center for Information Technology (GMD)
Research Institute for Open Communication Systems (FOKUS)
Hardenbergplatz 2, D-10623 Berlin, Germany

ABSTRACT

Multimedia operating systems must have real-time features and other supporting features in order to meet the QoS requirements of distributed multimedia applications. It is not enough to simply give communication processing activities higher priorities, it is necessary to realize a soft real-time framework which controls all system activities. In addition, feasible soft real-time scheduling schemes are the cornerstones for the functionality and efficiency of the framework. In the paper, the motivation, rationale and realization schemes of our framework proposal are addressed. We have also proposed and are experimenting with several base scheduling methods which are suitable for supporting continuous media communications and applications. These include a cooperative soft real-time scheduling method (CO-SCHEDULE) and a semi-imperative soft real-time scheduling method (SIM-SCHEDULE). The design and implementation structure of these scheduling methods are described. Preliminary implementation experiences are also presented.

1. Introduction

Distributed multimedia system has become one of the main focuses of computer networking and applications. The trend of current development is to extend and optimize the multimedia applications on endsystems (workstations, PCs), and to further advance the distributed multimedia applications by connecting the endsystems with large-bandwidth, high-speed networks.

Distributed multimedia (especially continuous media) applications need end-to-end performance support from both the networks and the endsystems. Until recently, much emphasis has been put on the networking aspect in order to construct a multiservice network. The provision of quality-of-service (QoS) guarantee has been extensively investigated in the networking field. Relatively less attention seemed to have been paid to end-systems and the operating system which controls the activities on the end-systems.

We note that in multimedia systems, it is not sufficient to guarantee the transmission of multimedia (MM) information flow between two endsystems at the level of networking service. In the end, it is the endsystem (controlled by an OS) that uses this networking service to provide multimedia service to end-users. Enhancements in both the endsystem hardware and software are needed in order to meet the time-constrained high processing requirements of MM applications.

An integrated multimedia system is a system where the CPU has direct control over all media including continuous media. This contrasts to a control-only system, where the continuous media data does not touch the operating system or main memory, but rather uses a separate infrastructure. The integrated solution has many advantages in terms of flexibility, scalability and cost, and is therefore our goal environment. In such integrated environments, the operating systems are critical both in accessing networking services and in controlling the activities on the endsystems. It is therefore vital to enhance the functionality and performance of the operating systems in order to provide feasible supports for multimedia communications and applications [2, 4].

In the MMOSS project (“Multimedia OS Supporting Environment”), we have proposed and are experimenting with a soft real-time framework for the operating systems with supporting features for multimedia. We have also proposed and are experimenting with several base scheduling methods which are suitable for supporting continuous media communications and applications. In the following, we address briefly the motivation, rationale and realization schemes of our framework and scheduling schemes.

2. Multimedia applications and soft real-time framework

2.1 Requirements from multimedia applications

It is well known that multimedia applications have posed a set of new functional and performance requirements on hardware and software components of a computer system. Especially, continuous media(CM) represents a significant departure from traditional applications and places a new set of constraints on the operating system supporting services. Two constraints among them are dominant requirements on the operating systems which try to support multimedia, i.e., the timeliness required to simulate continuous media and the unavailability of brute force resources with which to do so.

The central idea and starting point for supporting multimedia (especially continuous media) communications and applications should be to exploit the special features of them. That is, the support of OS to continuous media should exploit the special features of continuous media which are not present in or are not typical of other kinds of computer and communication applications. Some of these features can be identified as soft real-time, soft guarantee, periodicity, error-tolerance, adaptability, etc.

The types of MM applications are diverse. We mainly pay our attention to a MM application domain called multimedia collaboration (MMC). Examples of such applications can be found in circumstances such as [1]. In a typical MMC scenario, several users collaborate their work by working on some commonly accessible documents concurrently, and they usually interact with one another by way of live interactive audio/video connections. Because the parallel availability of the continuous media and the discrete media is necessary for the users' collaboration, the key point here is that the requirements from ***both*** should be satisfied at the same time.

2.2 Features of CM applications

In comparison to hard real-time applications, the nonrigidity or flexibility of continuous media (CM) applications and communications can be seen in several aspects. (Of course, we also keep in mind that not all the CM applications have the same flexibilities or the same degrees of flexibilities.)

First, some loss of CM data packets or fail to process some CM data packets or CM-related events are acceptable. Although the permissible loss varies from applications to applications, human eyes and ears apparently can tolerate and smooth some glitches from missing samples or events.

Second, we can identify three kinds of possible flexibilities (adaptabilities) which can be exploited. That is, rate, data volume and playback delay of CM applications are in many cases adaptive and can be adjusted in certain ranges.

With MMC as a background and as long as scheduling is concerned, continuous media applications and communications usually posses the following features:

1) A high degree of guarantee is required but a strict hard guarantee is not needed. That is, some violations of timing constraints can be tolerated. 2) A high degree of CPU-usage should be achieved. That means, for example, the schedulability's test and the scheduling mechanisms for real-time processes or threads should not be too time-consuming; sometimes a trade-off between efficiency and optimum should be made. 3) Real-time (RT) processes as well as non-real-time processes should be accommodated in the same framework. 4) A certain degree of elasticity in describing the real-time properties of a RT process at creation time should be allowed. For example, the worst execution time of the process need not be the exact real upper bound of the process execution at run time.

As is evident, one of the key points is how to exploit the possibility of sacrificing absoluteness for a higher efficiency and utilization, since MM systems are generally no hard real-time systems. And at the same time, a certain degree of guarantee of service provision should be maintained.

2.3 Soft guarantee and process framework

Our soft real-time framework proposal consists of a process framework for categorizing processes, some timing enforcement models and some base scheduling schemes. One of the overall goal of our soft real-time framework is to achieve a good approximation of the timing properties as predicted by the hard real-time scheduling theory so that the whole system is run in a more or less predictable manner.

We propose to use the following process framework. Processes in this framework will be treated in terms of processing possibility in three categories: "sure" (but not absolute) guarantee, "maybe" guarantee and "best-effort". The processes in the first two categories have to make an explicit claim about their timing constraints and, if accepted by the admission control of the scheduling subsystem, will then receive a corresponding service -- with "sure" and "maybe" guarantees of their timing constraints respectively. Timing constraint specification contains such real-time features as deadline, period duration, worst-case execution time, etc. The rationale for the above categorization is that the most important MM activities can then be favored at most. Less important activities can be less favored. Other activities take what is left.

The following three timing enforcement models are taken into consideration: (1) Cooperative: the scheduling system assumes that the timing constraints claimed by RT processes are also to be observed strictly by them. (2) Imperative: the scheduling system enforces the timing constraint on the running processes. That is, the scheduler's view of period, deadline and execution time will be imposed on the running processes, regardless whether the running processes have really been executed in the claimed periods etc. (3) Semi-imperative: It is assumed that the timing constraints claimed by RT processes are usually observed by them. The scheduling system monitors their executions. In case of timing specification violation, a timing-violation-handler defined by the scheduling system or by the process itself will be invoked. The usage of the models depends on the needs of the applications and the system environment, i.e., on the concrete semantics of the needed soft guarantee.

2.4 Base scheduling methods

As stated above, a scheduling subsystem in our framework contains an admission controller and a dispatcher exercising some scheduling methods. The basic requirement on a base scheduling method is that it should be simple, flexible yet powerful. Similar to the idea of [6], our work emphasizes on both predictability and flexibility. In our current implementation, two base scheduling methods (CO-SCHEDULE and SIM-SCHEDULE) are used to reflect the cooperative timing enforcement model and the semi-imperative timing enforcement model respectively. Both scheduling methods are designed by integrating some elements from the rate-monotonic scheduling, the priority-based scheduling and the weighted round-robin scheduling. The Generalized Rate Monotonic Theory [9] is used in both methods as a basis for admission control in creating new processes. The other related results of the theory are also used in other aspects of the system implementation such as preventing unbounded priority inversion by way of some forms of priority inheritance.

The choice of rate monotonic scheduling as a cornerstone of our scheduling methods lies in its simplicity and flexibility. In contrast to a pure fixed-priority scheduling method, rate monotonic scheduling has two attributes: (1) it can do admission control in a very simple way by checking the processing capacity required by the processes, especially in the context of soft real-time; (2) the priorities of the processes can be dynamic and will be adjusted when new processes are admitted into the system.

3. Overall realization schemes

As mentioned above, one of the overall goal in implementing our soft real-time framework is to achieve a good approximation of the timing properties as predicted by the hard real-time scheduling theory so that the whole system is run in a more or less predictable manner. By exploiting the soft real-time properties of the CM applications, this approximation can be realized in simple and efficient ways.

- **Scheduling subsystem**

Above all, the first realization issue is the efficient implementation and provision of the base scheduling services and diverse soft guarantee semantics. In order to facilitate the estimation of process processing capacity usage, our scheduling subsystem is designed and implemented to be time-constraint predictor and time-constraint enforcer in one. We have also designed the programming interfaces for process management in such a way that the applications can access the soft guarantee real-time services flexibly.

In addition, the following issues are also under our current investigation. The basic ideas to solutions are sketched below.

- **Implementation schemes for applications**

Since periodicity is a typical feature of the MM activities, the proposed scheduling methods provide “sure guarantee” and “maybe guarantee” only to virtual periodic processes. Naturally periodic applications can be implemented in the periodic scheme easily. Non-periodic applications which need guarantee can be rendered as pseudo-periodic processes to fit into the framework.

- **Structure of OS**

We advocate a “monolithic micro-kernel” structure with many traditional system call-like kernel functions being implemented as user-space libraries. In this way, the advantage of micro-kernel can be achieved to some extent. For example, rescheduling / context switch can be done without the influences of some long-delaying system call executions. At the same time, the accounting complication by a pure micro-kernel structure [6] can be avoided to some extent. First, the cost of invoking library functions is accounted as part of the capacity reserved by the user processes. Second, the capacity used by the system kernel on behalf of a user process should be “paid” by the user process and it is far easy to do such accounting in a monolithic way. The system should provide “advisory” cost information with regard to the execution of MM-related library functions and system calls in order to aid the user processes in reserving their processing capacities. It is sometimes quite difficult to estimate their costs for all circumstances accurately. This difficulty can be alleviated to some extent since our scheduling subsystem is designed to be time-constraint predictor and time-constraint enforcer in one. With the aid of such a scheduling subsystem, the user processes can undergo a “try and revise” procedure to attain a good estimation of their actual processing capacities.

- **Considerations on the cost of scheduling and interrupt processing**

In the case where the amount of such cost is quite small, it can be taken as part of the system variations which should be tolerated by the soft-guarantee semantics of the framework. In the case where the amount of such cost is too significant to be ignored, the capacity for such cost can be reserved by reserving the processing capacities for one or two virtual “system-overhead” periodic processes with very short virtual periods. (According to the rate-monotonic theory, these virtual processes can break the executions of other processes at any time when they are ready, because they have the highest priorities resulting from their very short periods. This has the practical effect that the virtually arbitrary breaks caused by the scheduling and interrupt handling are, to a large extent, compensated by these virtual reservations.)

- **Application-driven protocol processing architecture**

For a communication-intensive application, the cost of interrupt processing of incoming packets can be quite significant if a BSD style (SOFTINT) protocol processing method is used. An application-driven protocol processing architecture is needed in that the protocol processing functions are implemented as user-space libraries as much as possible. Such an architecture makes it simple to schedule protocol processing using the capacity reserved by its corresponding user processes. Techniques and measures are taken to reduce the influences and randomness of interrupt processing caused by incoming packets so that the cost for such interrupt processing can be more closely reserved in the form of the virtual “system-overhead” periodic processes mentioned above.

- **The FAST adaptive service model**

Soft real-time techniques can not only provide some degree of soft guarantee but can also accommodate the flexibility of adapting to changing system environments. The latter takes the form that it is possible to support a flexible and adaptive service model (the FAST model) in our soft real-time framework [3]. The FAST model can support both the application-initiated adaptations and the supporting-system-initiated adaptations needed by multimedia communications and applications. Our soft real-time schedulers can support the on-line change of the timing-constraints of processes and indicate system overload to application processes.

4. Soft real-time scheduling algorithms in experiment

Several soft real-time scheduling algorithms are under our current experiment. These include a cooperative soft real-time scheduling method (CO-SCHEDULE) and a semi-imperative soft real-time scheduling method (SIM-SCHEDULE). Another example is an elastic time-scale soft real-time scheduling scheme (ET-SCHEDULE). The CO-SCHEDULE is designed by integrating some elements from the rate-monotonic scheduling, the priority-based scheduling and the weighted round-robin scheduling. The SIM-SCHEDULE is an extension of the CO-SCHEDULE to run under semi-imperative enforcing model. And the ET-SCHEDULE is a novel scheduling framework into which many scheduling methods including CO-SCHEDULE and SIM-SCHEDULE can be placed to function. The soft real-time scheduling algorithms deal with both real-time processes and non-real-time processes.

4.1 Design overview of CO-SCHEDULE and SIM-SCHEDULE

A high-level presentation of the CO-SCHEDULE scheduler is shown in the following figure. The CO-SCHEDULE scheduler supports a set of high-priority periodic processes and a set of normal processes. A high-priority periodic process denoted as P^0_i has a higher priority than any normal processes and can preempt them at any time. Inside the set of the high-priority periodic processes, the processes are scheduled according to the rate-monotonic scheduling algorithm. The mapping of periods to the priorities are dynamic and are managed by the scheduling subsystem. The normal processes are again classified into different priority classes -- normal priority class 1, normal priority class 2, normal priority class 3,......, respectively, where a small number denotes a higher priority. Inside a normal priority class n, the processes denoted as P^n_i are scheduled according to a weighted round-robin scheduling algorithm. That is, when scheduled to run, a process P^n_i with weight w^n_i is given w^n_i time budget. The process runs until it is blocked or the time budget is used up. In the latter case, the process is placed to the end of the class n waiting queue.

The CO-SCHEDULE scheduler runs according to the cooperative time model. That is, the processes are assumed to run not longer than the worst-case-execution claimed by them. The scheduler schedules the processes simply according to their priorities and weight. No monitoring of the execution time of the current process is conducted.

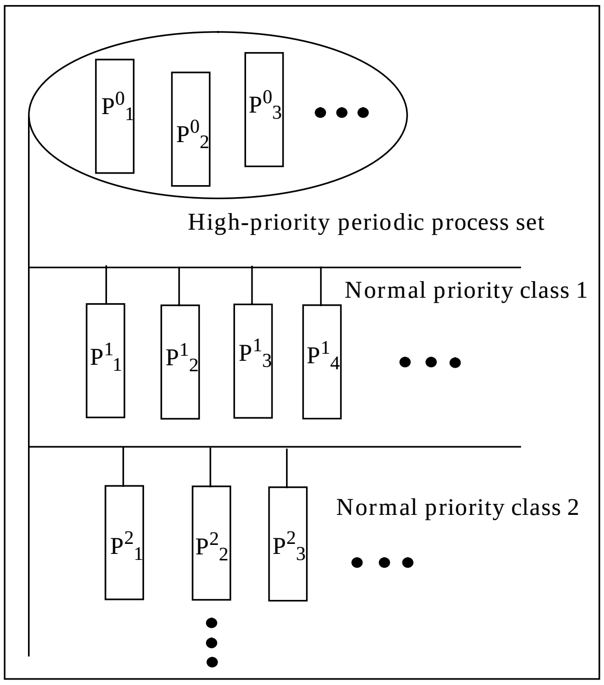

Figure 1. CO-SCHEDULE scheduler overview

- **SIM-SCHEDULE: a semi-imperative soft real-time scheduling method**

The SIM-SCHEDULE scheduler extends the CO-SCHEDULE method by monitoring the execution of the high-priority periodic processes. The scheduler preempts a high-priority periodic process if this periodic process has not completed its task in one of its period after using up its claimed C^0_j execution time. Several following actions are then possible. For example, it can be rendered into a process of a certain normal priority class m to execute its emergency-handler or it can be simply allowed to complete its current execution in its next period.

- **Usage of the schedulers**

Periodicity is a common property of real-time tasks in continuous media applications. The high-priority periodic process set can, for example, be used to implement functions for isochronous data transmission and consumption. The set of the high-priority periodic processes can be (soft) guaranteed to meet their timing specification if the processor usage of the set are checked to be in the capacity limit given by the theorems in [9]. Note the processes in the normal process class can also be scheduled to run in some sense of real-time if carefully planned.

The implementations of the schedulers can be derived from a priority-based scheduler. The high-priority periodic processes are assigned different priorities according to their periods (if necessary, do a period-transformation). The processes are allowed to mask some small parts of its process body as non-preemptable.

- **Role of emergency handler**

System-defined and user-defined emergency handlers are provided in the SIM-SCHEDULE scheme. By emergency handler, the scheduling subsystem can, for example, notify an application process that it is unlikely to schedule its activity according to its time-constraints. This mechanism allows the application processes to react to the missed deadlines in an application-specific manner. Such a notification mechanism is one of the key features with which an adaptive service supporting model can be built.

Under the control of the SIM-SCHEDULE scheduler, the mal-function of one of the high-priority periodic processes will not have negative effects on other high-priority periodic processes and the processes of the priority class 1 to (m-1).

- **Implementation of “maybe” guarantee**

The outcome of the decision of the admission controller of the scheduling subsystem can be one of the following three cases: a guarantee of the real-time attributes of the new process can be maintained; a “maybe” guarantee of the real-time attributes of the new process can be maintained; or a “best-effort” maintenance of the real-time attributes of the new process will be done. We have proposed two algorithms to implement the semantics of “maybe” guarantee. “Sure” guarantee and “maybe” guarantee processes are not only treated differently in admission control, they are also treated differently in other situations.

4.2 Programming interfaces of usage

- **Cooperative periodic process**

Under the cooperative periodic process model, the periodic processes independently use the timing facilities to reflect and embody their own timing constraints. It is therefore assumed that each instantiation of the task body will be executed within a worst-case execution time and the OS does not need to impose any extra constraints on its execution. The model is direct and simple. But it is then impossible to have a good control of timing violations caused by imprecise estimation of processor usage (execution time) of the processes, by other transient overload conditions, or by synchronization conditions.

A programming interface for the periodic processes is provided so that the user only need to provide task body and specify real-time constraints such as start time, period, execution-time. The real arrangement of the periods will then be done by the OS automatically. The programming interface looks like:

```
CreateCoPeriodicProc (unit_prog, start_time,
  end_time, period, unit_comp_time)
```

where, *unit_prog* is the entry point of a procedure which should be executed in each period, *start_time, end_time, period, unit_comp_time* are the intended start time, end time, period and computation time per period of the intended virtual periodic process.

- **Semi-imperative periodic process**

Under a semi-imperative periodic process model, the periodic processes provide specifications on their timing constraints. The OS monitors the execution of the process to detect possible timing violations. There are several varieties of timing violations and violation handling. If the execution of the task body has ever exceeded the worst-case execution time as claimed by the process in its RT attributes, then it is a timing violation from the side of the process. If the OS detect that the system can not provide enough processing cycles to meet the need of the claimed timing requirements of the process, then it is a timing violation from the side of the OS. In the case of a timing violation, handling can be done by a OS procedure or a process-specific procedure. The method of handling can be roughly classified into abortive and corrective. An example programming interface looks like:

```
CreateSimPeriodicProc (unit_prog, start_time,
  end_time, period, unit_comp_time, excep)
```

where, *unit_prog* is the entry point of a procedure which should be executed in each period, *start_time, end_time, period, unit_comp_time* are the intended start time, end time, period and computation time per period of the intended virtual periodic process, *excep* is the exception handler which should be called by the OS when a timing violation arises. (The simplest form of *excep* is the termination of the whole process). In our opinion, a semi-imperative process model is needed in most systems.

For timing enforcement, we use a simple scheme used in many static priority preemptive schedulers -- a periodic clock interrupt runs the scheduler; a budget timer ensures that control is returned to the scheduler if a task exceeds its permitted worst-case-execution time.

- **Run a non-periodic process in the periodic scheduling framework**

The CO-SCHEDULE and SIM-SCHEDULE scheduler both support a set of high-priority periodic processes and a set of normal processes. The set of the high-priority periodic processes can be guaranteed (“sure” or “maybe”) to meet their timing specification if the processor usage of the set are checked by the admission controller to be in the capacity limit.

However, we also sometimes want to guarantee the execution of a non-periodic process which does not do continuous-media-related processing. For example, in a MMC scenario, we want to guarantee the file transfer accompanying a video connection. Note, in order to check the schedulability bound in a scheduling system using rate-monotonic scheduling, a non-periodic process has also to be considered and scheduled in the same way as a periodic process. In order to run a non-periodic process in our periodic scheduling framework, the scheduler has to schedule the non-periodic process in an “artificially periodic” way.

Since non-periodic programs do not have an explicit periodic structure and requirement, they can be handled by arbitrarily setting the period computation time and period to yield an intended rate. The following implementation, for example, is a possible form:

```
CreateProgAsPeriodicProc (prog, start_time,
  period, unit_comp_time)
```

where, *prog* is the entry point of a program, *start_time, period, unit_comp_time* are the intended start time, period and computation time per period of the intended virtual periodic process.

5. Preliminary implementation experiences

Preliminary experiences have been gained in implementing our soft real-time framework, especially the CO-SCHEDULE and SIM-SCHEDULE scheduling subsystems, in the MMOSS project on a development environment with TI-TMS-C’40 processors[2]. It turns out that it is easy to implement our base schedulers efficiently. Related programming interfaces for process management have been designed and are being implemented. More other work is under way. Practical experiences with the FAST service model are also gained and reported [3].

It is of course very important to have a very low scheduling cost, otherwise the goal of real-timeliness can not be achieved. A scheduling scheme such as SIM-SCHEDULE can indeed be implemented in a very efficient manner -- based in part on a priority-driven dispatcher. The following table and figure give some examples of the scheduling cost of a preliminary version of the SIM-SCHEDULE as we have implemented on a TMS-C’40 processor. The numerical examples listed in the table (for scheduling interval of 5ms) are the maximum costs which are not usually needed in a normal execution scenario. Note that the TMS-C’40 is a RISC processor with many registers. The scheduling cost can be reduced greatly if the operations for register switch can be halved or the processor clock frequency is raised.

Table 1. Maximum Scheduling Cost

Max. Time for Scheduling in µs / Max. Scheduling Cost in %	**Processor**	**Clock (in MHz)**	**Frequency**
No. of Ready Processes in run_q	40	80	120
1	5.70 / 0.11	2.85 / 0.06	1.90 / 0.04
5	7.00 / 0.14	3.50 / 0.07	2.33 / 0.05
20	11.87 / 0.24	5.94 / 0.12	3.96 / 0.08
Normal Case (No Process Switch)	1.20 / 0.024	0.60 / 0.012	0.40 / 0.008

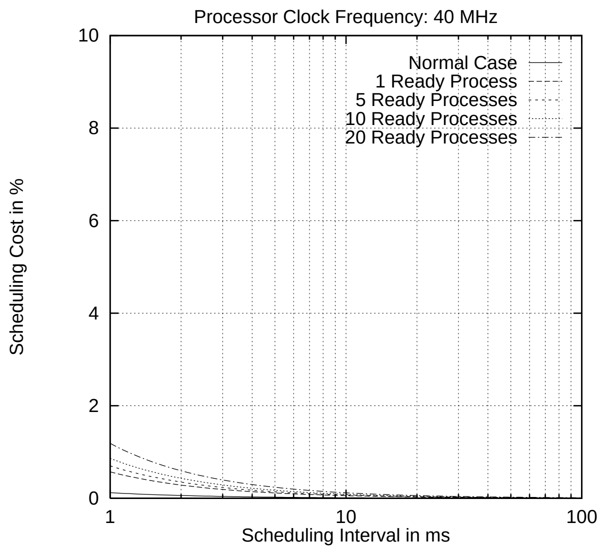

Figure 2. Maximum Scheduling Cost in %

6. Concluding remarks

It is clear that multimedia operating systems must have real-time features and other supporting features for distributed multimedia applications. Only giving the highest priority to communication protocol processing can only achieve the effect that the multimedia data packets are processed as soon as possible. This is, however, not enough for the whole system effect, since the application processes that consume these MM data can not be scheduled to use these data in time and according to their periodicities. Therefore, there is the necessity for a real-time framework such as ours which controls all system activities.

Our soft real-time framework proposal consists of a process framework for categorizing processes, some timing enforcement models and some base scheduling schemes. Feasible soft real-time scheduling schemes are the cornerstones for the functionality and efficiency of the framework. We are experimenting with several base scheduling methods which are suitable for supporting continuous media communications and applications. These include a cooperative soft real-time scheduling method (CO-SCHEDULE), a semi-imperative soft real-time scheduling method (SIM-SCHEDULE) and an elastic time-scale soft real-time scheduling scheme (ET-SCHEDULE). Preliminary experiences with these algorithms are encouraging.

The work in [8] concerning a Sun version of SVR4 UNIX fails partly because they only provide static fixed priority scheduling and neither admission control nor timing-constraint violation control is exerted. Earlier work such as those in [5, 7] has shown the feasibility of real-time techniques to multimedia environments but they generally lack the flexibility for a dynamic system. Our work concerning the realization of a soft real-time framework aims at both predictability and flexibility. It is also simple to implement.

Acknowledgment

Discussions with Radu Popescu-Zeletin have helped to refine the ideas. Cooperative team-work from other MMOSS project members is also gratefully acknowledged.

References

[1] M. Altenhofen, J. Dittrich, et al, “The BERKOM Multimedia Collaboration Service,” *Proc. ACM Multimedia93*, 1993.

[2] C. Fan, “MMOSS: Soft Real-Time Operating System Support in a Multimedia Communication Subsystem,” *Proc. 19th IFAC/IFIP Workshop on Real-Time Programming*, IFAC/IFIP, June, 1994.

[3] C. Fan, “An Adaptive Service Model for Supporting Multimedia Communications and Applications,” *to appear in Proc. APCC’95*, Japan, 1995.

[4] R. G. Herrtwich, The Role of Performance, Scheduling, and Resource Reservation in Multimedia Systems, *Operating Systems of the 90’s and Beyond, LNCS 563*, Springer, 1992.

[5] K. Jeffay, D. L. Stone & F. D. Smith, Kernel Support for Live Digital Audio and Video, *Computer Communications*, Vol. 15, No. 6, July 1992.

[6] C. W. Mercer, S. Savage & Hideyuki Tokuda, Processor Capacity Reserves for Multimedia Operating Systems, *Technical Report CMU-CS-93-157*, Carnegie Mellon University, 1993.

[7] J. Nakajima, M. Yazaki & H. Matsumoto, “Multimedia/Realtime Extensions for Mach 3.0,” *Proc. USENIX Microkernel Conference*, Apr. 1992.

[8] J. Nieh, et al, “SVR4 UNIX Scheduler Unacceptable for Multimedia Applications,” *Proc. 4th Int’l Workshop on Network and Operating Systems Support for Digital Audio and Video*, Nov. 1993.

[9] L. Sha, R. Rajkumar & S. S. Sathaye, “Generalized Rate-Monotonic Scheduling Theory: A Framework for Developing Real-Time Systems,” *IEEE Proceedings Journal*, Jan. 1994.