Application-controlled Multimedia Scheduling

Young Woong Ko*, Chuck Yoo**

Operating System Lab

Abstract

Real-time scheduling algorithms such as Rate Monotonic and Earliest Deadline First are a basis for many multimedia scheduling algorithms. However, real-time scheduling mechanisms have drawbacks to multimedia applications. One of them is that multimedia application such as digital video has a large variation in execution time, which makes real-time scheduling algorithms very inefficient in terms of resource utilization. To improve the resource utilization, this paper introduces an application-controlled approach. A task passes scheduling information to the scheduler so that the scheduler dynamically changes the priority of the task. In addition, an advantage of our approach is that multimedia tasks are scheduled based on priority only. Therefore, multimedia tasks do not penalize time-sharing tasks. We implement the new scheduler on Linux kernel and present the experiment results with Berkeley MPEG-1 video decoder.

1. Introduction

Multimedia applications have time-constrained data types such as digital audio and video. Audio and video streams periodically change the values of the media data over time, and they must be presented at a specific deadline. A key of designing a multimedia system is whether media data are handled properly in terms of time. Therefore, the importance of a scheduling algorithm is well recognized, and real-time scheduling techniques have been used. But since traditional real-time scheduling algorithms are designed for mission-critical applications, implementing the algorithms is overkill for multimedia systems because continuous media data demand less strict deadline and missing a

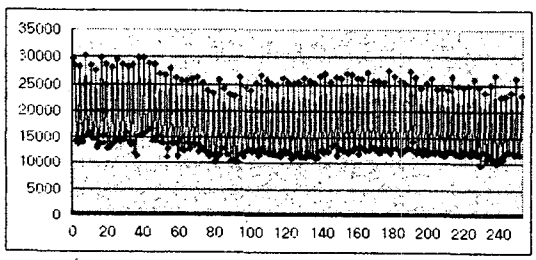
This work was supported by grant No. 97-01-00-09-01-3 from the Basic Research program of the Korea Science & Engineering Foundation.

** Professor of Dept. of Computer Science and Engineering, Korea University

* Doctor course of Dept. of Computer Science and Engineering, Korea University

deadline is not a severe failure. And multimedia applications usually need only statistical time guarantees.

Furthermore, multimedia applications have characteristics that make it difficult for real-time scheduling algorithms to deal with. One of them is the variation in the execution time within a period. An example is a MPEG compressed video stream. The decoding time depends on the type of a frame. Decoding I and P frames takes longer than B frames because MPEG[4] video compression algorithm relies on block-based motion compensation for the reduction of the temporal redundancy and DCT-based compression for the reduction of spatial redundancy. We measured the decoding time of a MPEG-1 movie clip, and Fig 1 represents the variation in the decoding time. The x-axis is the video sequence, and the y-axis is the execution (decoding) time. On the average, the execution time of I and P frames converges to 30 ms, and B frames converge to 15 ms.



[Fig. 1] Decoding time of MPEG-1 video frames

Such a large variation in execution time makes real-time algorithms inefficient because they are based on the worst execution time. Therefore, the schedulability becomes too optimistic and thus reduces the utilization of available resources.

This paper has two goals: 1) we want to design an algorithm that can cope with a large variation in execution time, 2) the algorithm should be implementable and coexist with time-sharing scheduler on general-purpose operating systems such as UNIX. It is well known that general-purpose operating systems lack the time-constrained support for media data. As multimedia applications are widely available, we believe, it becomes more and more important that general-purpose operating systems support them efficiently.

However, implementing a real-time algorithm on general-purpose operating systems is not simple, and it is not clear whether the algorithm works as specified in simulations or proofs due to the non-deterministic nature of general-purpose operating systems. A typical approach is to maintain a separate run queue of high (highest sometimes) priority for multimedia applications. This makes it difficult for time-sharing applications to coexist with multimedia applications. Our goal is to propose a scheduling algorithm that is simple to implement and also does not penalize time-sharing applications on general-purpose operating systems.

Specifically, our approach is based on application -controlled scheduling to cope with the wide variation in execution time. The idea is that the application passes scheduling information to the scheduler, and the scheduler dynamically adapts itself to meet the timing requirement of multimedia task. We have implemented application-controlled scheduling on Linux (as a general-purpose operating system) and run various experiments to verify that our scheduling algorithm can be easily implemented on a general-purpose operating system and does not penalize time-sharing applications.

The paper is structured as follows. In Section 2, the related scheduling algorithms are introduced. Section 3 demonstrates a scheduling problem of Linux to support multimedia tasks. In Section 4, we describe why application-controlled mechanism is needed. Section 5 explains application-controlled

scheduling algorithm in details. Specifically, we use MPEG decoding as multimedia task. Section 6 presents the implementation and experiment results. The paper concludes in Section 7.

2. Related works

We briefly describe multimedia scheduling algorithms [5][7][8][9] and related topics.

RM(Rate Monotonic) scheduling is a simple, static priority policy that is widely used in real-time systems. It uses static-priority driven scheduling, where tasks with shorter periods get the higher priorities. The policy is preemptive and assigns scheduling priorities according to the rate of arrival for a task, with higher priorities for more frequent tasks. In order to use the RM algorithm, we must be able to specify execution time for each task.

EDF(Earliest Deadline First) algorithm is one of the most popular dynamic priority scheduling algorithms which schedules the task with the closest deadline first. EDF scheduling is optimal under the light load situation, but it cannot perform well when the system is overloaded. EDF scheduling does not require task to be periodic.

A more sophisticated version of EDF scheduling is processor capacity reserves [6]. It was added to the real-time Mach. The reserve abstraction characterizes the timing requirement and processor capacity reservation requirement for multimedia applications. The reservation approach is used with admission control to allow real-time tasks to reserve a fixed share of the resource. And resources not reserved are allocated to conventional tasks using a time-sharing scheduler or round-robin scheduler.

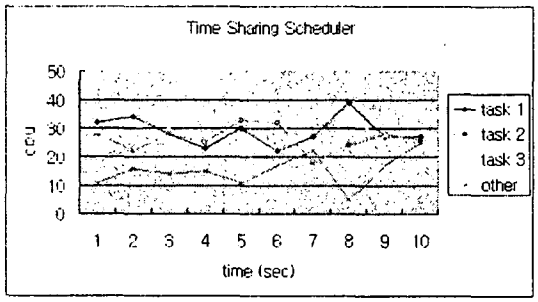
SMART[8] approach achieves diverse multimedia characteristics by differentiating between the importance and urgency of real-time and conventional applications. This is done by integrating priorities and weighted fair queuing for importance, then using urgency to optimize the order in which tasks are serviced based on EDF scheduling mechanism.

3. Scheduling problems in Linux

Originally, Unix-based monolithic operating systems such as BSD[1], SVR4[2], and Linux[11] are designed to provide fairness to every task and insure good responsiveness. In order to support timing characteristics of multimedia tasks, the early design policies of operating system must be changed to guarantee predictable service (e.g. deadline).

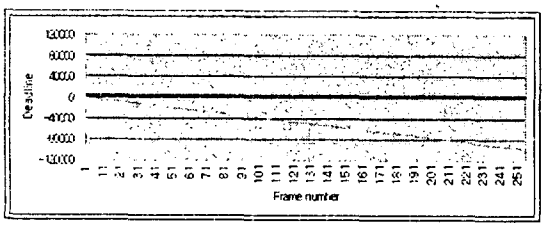
In monolithic operating systems, the most significant problems that interfere that guaranteeing predictable processing of task is the non-deterministic behavior that springs from delays caused by non-preemptable system call, blocking I/O, and unpredictable variation in kernel. Recently, many research groups are pursuing to resolve these problems based on micro-kernel architecture [10]. Currently, Linux operating system supports three scheduling policies: time-sharing, real-time FIFO, real-time RR(Round Robin). Real-time FIFO and RR support soft real time that cannot guarantee context switching and reaction times within a bound. If higher priority process wishes to run, all other processes with lower priorities cannot be scheduled.

To demonstrate a scheduling problem that Linux has, we designed experiments with MPEG-1 data. The task set is composed of four tasks. The task 1 is a multimedia task (Berkeley MPEG-1 player), and task 2 and 3 are two batch tasks (Matrix computation). The task 4 represents background daemons that run on Linux. Fig 2 shows the CPU capacity of each task allocated by the Linux (default) time-sharing scheduler. It shows that time-sharing scheduler fairly assigns CPU capacity to each task.



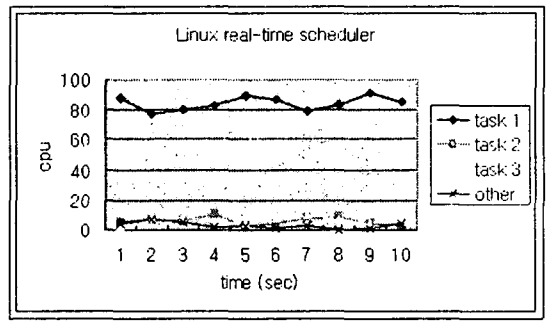
[Fig 2] CPU capacity by Linux time-sharing scheduler

However, the task 1 does not run properly. It is shown in Fig. 3. The y-axis is the completion time from the deadline. A negative value means how much the frame is missed past the deadline. Although the task 1 receives the CPU fairly as shown in Fig. 2, most of its frames do not meet the deadline. Therefore, it is clear that the Linux time-sharing scheduler does not support multimedia applications. If more than one multimedia application run, the situation deteriorates regardless of how much CPU resource is available. So we turn to the real-time scheduling policy of Lunix.



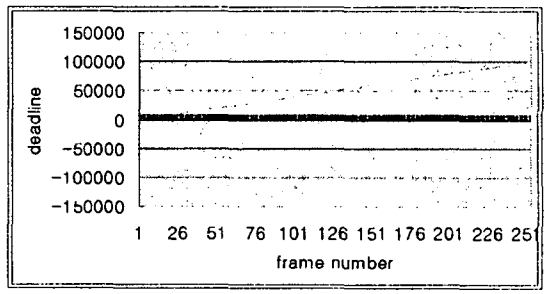
[Fig 3] Completion time with Linux time sharing scheduler

Fig. 4 shows the CPU capacity of each task allocated by the Linux real-time scheduler. What happens is that the real-time scheduler assigns the entire CPU capacity to task 1, and non-real-time tasks get almost no CPU. It shows that the time-sharing applications are seriously penalized by a multimedia task.



[Fig 4] CPU capacity by Linux real-time scheduler

Fig. 5 shows the completion time with the Linux real-time scheduler. It appears that most of frames are executed within the deadline. But, the problem of Fig. 5 is that the completion time keeps increasing over time, which means that frames are executed faster and faster. It is possible that a frame is played even before the period begins. This is because the entire CPU is allocated to task 1. Real-time FIFO and RR have the same behavior.



[Fig 5] Completion time with Linux real-time scheduler

4. Why application-controlled approach?

Various application-controlled mechanisms have been used in computer systems: memory management [13], buffer cache management [15], and scheduling issue [3].

First, Kieran's approach [13] is that the virtual memory system provides application control of physical memory using external page-cache management. Generally, the virtual memory system gives an illusion to application that it has a large main memory at its disposal, although the computer may have a relatively small physical memory. And virtual memory system prevents the application from accessing physical memory. This makes an inefficient use of physical memory in environment where application knows much about the memory usage such as read-ahead, write back characteristics of data. By allowing application control, Kieran achieves significant performance benefits. Secondly, the LRU-like buffer cache management policy exhibits poor performance for multimedia. For example, in MPEG decoding schemes, I frame must be kept in buffer until related B and P frames are decoded and displayed. The LRU replacement policy does not distinguish the type of frames. Chris Maeda [14] presents a framework that allows applications to control buffer cache management decisions. Finally, in scheduler activation [3], processor allocation is influenced by the information passed from application. And the kernel notifies the application of system events that affect the progression of the application. By exchanging scheduling information, the scheduler activation mechanism shows good results.

Our motivation for application-controlled approach is from an observation - applications that do not follow scheduling policies of a general-purpose operating system may result in poor performance. For example, the scheduling decisions of system resources such as disk, memory, and processor are made based upon fair resource distribution. And operating systems are mainly concerned about throughput consideration. Accordingly, time-constrained multimedia application could not properly be executed as demonstrated in Section 3.

Now we want to show that even real-time scheduling algorithms are not efficient for multimedia applications. Real-time scheduling theory relies on the schedulability test based on resource utilization bounds. The bound-based test is applicable to real-time task. But because the execution time of multimedia tasks varies widely, using the maximum value of the task execution time (worst execution time) meets the deadline guarantee, but it may result in very inefficient resource utilization. According to the rate monotonic (RM) scheduling, the task's period, T, is the amount of time between the arrival of one instance of the task and the arrival of the next instance of the task. The execution time, C, is the amount of processing time required for each occurrence of the task. The utilization of the processor is $U = C/T$. For example, if a task has a period of 40 ms and an execution time of 10 ms, its processor utilization is 0.25.

The schedulability test says that the sum of the processor utilization of tasks cannot exceed a value of 1. More accurately, RM scheduling uses equation (4.1) that provides the upper bound of utilization. If 3 tasks are scheduled together, in order to guarantee to meet the deadline, the sum of processor utilization should be less than or equal to 0.779 (from Table 1).

$$\frac{C_1}{T_1}+\frac{C_2}{T_2}+\frac{C_3}{T_3}+....+\frac{C_n}{T_n} \le n(2^{1/n}-1) \quad .....(4.1)$$

n	$n(2^{1/n}-1)$
1	1.0
2	0.828
3	0.779
4	0.756
.	0.743
.	.
.	.
∞	$\ln 2 \cong 0.693$

[Table 1] Upper bounds of RM scheduling

Note that this upper bound is derived for the deadline guarantee of hard real-time systems. We show that the schedulability test based on this

upper bound is very inefficient for multimedia. We picked three MPEG clips randomly and measured the following data: number of frames, average execution time, and worst case execution time. They are in Table 2.

Items	a.mpg	b.mpg	c.mpg
Period (ms)	40	40	40
Number of I frame	65	30	82
Number of B frame	512	238	732
Number of P frame	192	90	163
Number of all frame	769	358	977
I frame Average Exec time	8.981	11.694	10.237
B frame Average Exec time	3.533	7.243	4.789
P frame Average Exec time	8.18	13.043	8.433
Avg Exec time of all frame	5.154	9.074	5.854
Worst Exec time	16.201	24.210	19.906

[Table 2] Execution time of MPEG data

From Table 2, the processor utilization based on worst execution time is:

- a.mpg : $C_1 = 16.201$; $T_1 = 40$; $U_1 = 0.405$
- b.mpg : $C_2 = 24.210$; $T_2 = 40$; $U_2 = 0.605$
- c.mpg : $C_3 = 19.906$; $T_3 = 40$; $U_3 = 0.497$

The total utilization of these three tasks is 1.507. Because this value is greater than the upper bound for the RM scheduling (1.507 > 0.779), the schedulability test says that three MPEG-1 clips cannot be scheduled together to meet the deadline.

We believe that the test result based on the worst execution time is too optimistic. The reason is that multimedia tasks have soft real-time characteristics and that occasional deadline misses can be tolerated. If we use the average execution time instead of the worst execution time, we get the following utilization.

- a.mpg : $C_1 = 5.154$; $T_1 = 40$; $U_1 = 0.128$
- b.mpg : $C_2 = 9.074$; $T_2 = 40$; $U_2 = 0.226$
- c.mpg : $C_3 = 5.854$; $T_3 = 40$; $U_3 = 0.146$

Now the total utilization required for the three tasks is 0.50, and it is less than the upper bound for the RM scheduling, the schedulability test tells that three MPEG-1 clips can be scheduled together. But this approach is too aggressive in that overload situations may occur so that some miss the deadline. We want to overcome this overload problem with the application-controlled approach. When an application executes, it exchanges scheduling information with the scheduler and adjusts itself with the overload situation dynamically. The details are explained in Section 5.

5. Application-controlled multimedia scheduling algorithm

The key idea of application-controlled approach is, before a period begins, the application passes application-controlled information to the scheduler, and the scheduler uses the information to change the priority of the task and to control the overload situation. Application-controlled scheduling algorithm is based on LSF (Least Slack time First) algorithm [12]. In the LSF algorithm, the slack time of a task is the amount of time that the task can be delayed before it misses its deadline. So the slack time is defined below.

Slack time = Deadline - Execution time - current time

The slack time is used to choose the next task to be executed. In other words, the LSF algorithm will choose a task with the smallest slack time. We add application-controlled information and modify the slack time as follows:

Slack time = Deadline - Average execution time - Current time - Application-controlled information

The scheduler computes the slack time using the application-controlled information passed from a task, and decides the task priority. The next question is how to define the application-controlled information. Intuitively, a period with a long execution time should get higher priority because its chance to miss the deadline is higher than one with shorter execution time. We use *variation time* as application-controlled information as defined below.

Variation time = Expected execution time of the period - Average execution time

The bigger the variation time is, the higher the priority of task is. An important question is how to get the expected execution time of the period, and it is up to the specific multimedia application. We will show the calculation of the variation time for MPEG-1 decoding below.

When the slack time gets negative, it means that the overload situation happens. Our application -controlled approach allows the scheduler to inform the overload situation to tasks via an upcall mechanism. If overload situation is detected, the scheduler selectively sends upcalls to applications. Upon the receipt of the upcall, applications can take actions to reduce the load such as dropping frames.

The main advantage of our application-controlled approach is that the task scheduling is uniformly done based on priority. It does not have a separate scheduling queue nor dedicate system resources such as CPU. This advantage makes our approach applicable to general-purpose operating systems. A weak point of our approach is that the scheduling overhead could be high because the slack time is calculated every period.

Our application controlled scheduling mechanism is summarized:

① Variation time is used for the application-controlled information.

② Processor utilization is improved by using the average execution time, not worst execution time.

③ Since the average execution time may cause temporary overload situation, we use an upcall mechanism to inform tasks to take proper actions.

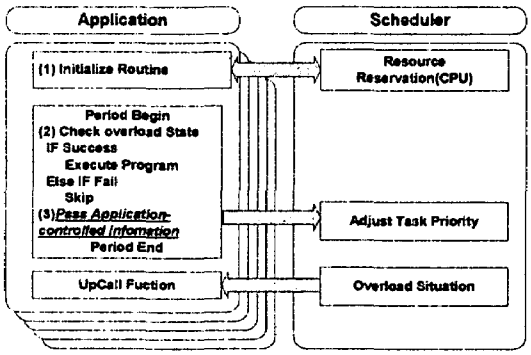
④ Scheduling is unified with priority. There is neither separate multimedia queue nor the dedication of system resources. Therefore, our approach is easily applicable to general-purpose operating systems.

Now we apply the application-controlled approach to MPEG-1 decoding. To calculate the variation time, we use the average execution time of each frame decoded in the period. Since the execution time of a period depends on the frame type, the expected execution time is the average execution time of frame type. When a task decodes a MPEG-1 clip, the priority of a task will vary depending on the frame type it is decoding. Table 3 shows the variation time for I, B, and P frames of three MPEG-1 clips in Table 2.

Items	a.mpg	b.mpg	c.mpg
Average execution time	5.154	9.074	5.854
I frame Average exec time	8.981	11.694	10.237
B frame Average exec time	3.533	7.243	4.789
P frame Average exec time	8.180	13.043	8.433
I frame Variation time	3.7	2.6	4.4
B frame Variation time	-1.6	-1.8	-1.1
P frame Variation time	3.0	4.0	2.6

[Table 3] Variation time for each frame type

The implementation of application-controlled approach for MPEG-1 decoding is described in Fig. 6



[Fig 6] Application-controlled scheduling for MPEG-1

① Initialize

A task specifies media parameters such as period and average execution time to the scheduler. According to the schedulability test, if processor capacity is enough to accept the task, the scheduler executes task. If not, the scheduler rejects the task.

② Check overload situation

In each period of task, it checks the overload state flag. If the flag is set, the application must take some action such as dropping frame or reducing frame rate based on frame type.

③ Pass application-controlled information

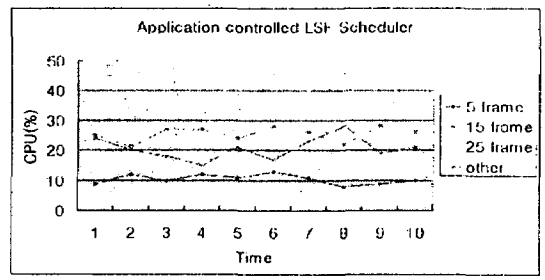
In the end of each period, the application passes the variation time and type of the next frame to the scheduler. Then scheduler adjusts the task priority based on variation time.

6. Implementation and experimental result

We have implemented our approach in Linux kernel (version 2.0.30) and modified the Berkeley MPEG-1 video decoder. The experiments were performed on a personal computer that equipped with Intel Pentium 100 MHz, 32 MB of primary memory, and 1GB of local disk space. The first experiment is to measure CPU time distribution. To test the new scheduler, all the measurements are done in a fully loaded system. Four tasks are used to make the experiment system fully loaded. Task 1-3 are multimedia tasks below and task 4 is a time-sharing task that does matrix computation.

- Task 1 : C_1 = 16.0; T_1 = 200; U_1 = 0.08
- Task 2 : C_2 = 16.0; T_2 = 67; U_2 = 0.24
- Task 3 : C_3 = 16.0; T_3 = 40; U_3 = 0.40

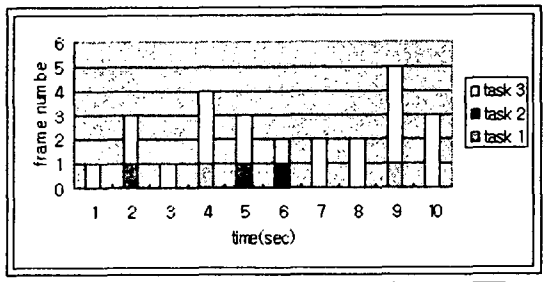
Remember that the Linux real-time scheduler dedicates the entire CPU to a multimedia task as shown in Fig. 4. Fig. 7 shows that the new scheduler distributes CPU time close to the utilization of four tasks.



[Fig 7] CPU distribution in application-controlled scheduler

Fig 8 shows how many frames were dropped in overload situation. In our experiment, totally, 450 frames were decoded in every second. And table 4 shows the ratio of dropped frame, which occurred 28 frames out of 450 frames (6.2 %). We believe that it is fairly acceptable ratio for general-purpose operating system.

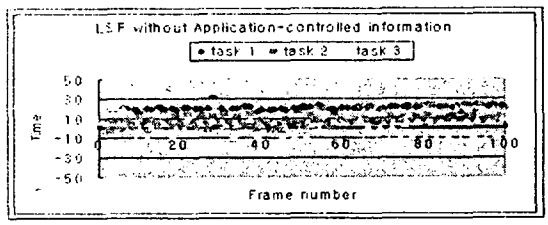
[Fig 8] Dropped frame number in every second



	Task 1	Task 2	Task 3
Number of frame	50	150	250
dropped frame	2	3	23
drop ratio(%)	4	2	9.2

[Table 4] The ratio of dropped frame

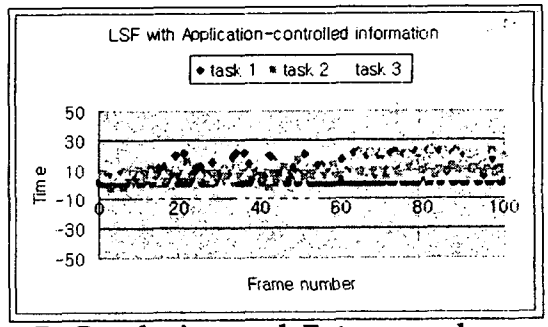
The second experiment is to measure the deadline miss rate. Fig. 9 and 10 show how scheduler precisely guarantees deadline of each period. The x-axis is the video sequence, and the y-axis is the completion time. Fig 9 shows LSF scheduler without application -controlled information. In this situation, deadline miss rate is too high. Deadline miss was occurred 115 frames out of 300 frames. (38.3 %)



[Fig 9] Completion time with LSF scheduler without user information

Fig 10 shows LSF scheduler with application -controlled information. Deadline miss rate was extremely reduced. Deadline miss was occurred 47 frames out of 300 frames.(15.6 %)

[Fig 10] Completion time with application -controlled LSF scheduler



7. Conclusion and Future work

We present a new approach in multimedia scheduling. It allows the interaction between an application and the scheduler. The application passes some scheduling information to the scheduler, and the scheduler dynamically computes the priority of the application. When applying our approach to MPEG video decoding, we show that the resource utilization can be significantly improved, compared with real-time algorithms. Another advantage is that multimedia tasks are scheduled based on their priority only. Therefore, multimedia tasks on a general-purpose operating system do not penalize time-sharing tasks. We believe that this is very important because the current approach of general-purpose operating systems for multimedia tasks typically uses a separate scheduling queue and tends to dedicate resources. We have done the implementation on Linux and verified that the new scheduler works well even in a fully-loaded configuration - the deadline miss rate improves by 23 percent. Some may argue that the interaction between an application and the scheduler is problematic. However, as shown in scheduler activation and open implementation in Section 4, we believe that our approach has its role in multimedia scheduling, specially for general-purpose operating system.

References

[1] Marshall Kirk Mckusick, keith Bostic, Muschael J. Karels and Jone S. Quartman, the Design and Implementation of the 4.4 BSD UNIX Operating System, Addison Wesley, 1996.

[2] American Telephone and Telegraph, UNIX System V Release 4 Internals Student Guide, 1990.

[3] T. Anderson, B. Bershad, E. Lazowska and H. Levy. Scheduler Activations: Effective Kernel Support for the User-Level Management of Parallelism." ACM Transactions on Computer Systems 10, 1 (February 1992), pp. 53-79.

[4] Ralf Steinmetz and Klara Nahrstedt. Multimedia: Computing Communications and Applications. Prentice-Hall, Englewood Cliffs, NJ, 1995.

[5] C.L. Liu and J.W. Layland. "Scheduling algorithms for multiprogramming in a hard real-time environment," JACM, pages 460-461, 1973.

[6] C. W. Mercer, S. Savage, H Tokuda, "Processor Capacity reserves: Operating System Support for Multimedia Applications", Proceedings of the IEEE international Conference on Multimedia Computing and Systems, Boston, MA, pp. 90-99, May 1994.

[7] P. Goyal, X. Guo, H. M. Vin, "A Hierarchical CPU Scheduler for Multimedia Operating Systems", Proceedings of the Second Symposium on Operating Systems Design and Implementation, Seattle, WA, PP. 107-122, Oct, 1996.

[8] J, Nieh and Monica S. Lam, "The Design, implementation and Evaluation of SMART : A Scheduler for Multimedia Applications," Proceedings of 16th ACM Symposium on Operating Systems Principles, St Malo, France, October, 1997.

[9] Raj Yavatkar, K. Lakshman, "A CPU Scheduling Algorithm for Continuous Media Applications", In 6th International NOSSDAV Workshop.

[10] H. Tokuda, T. Nakajima, and P. Rao. "Real-Time Mach: Towards a Predictable Real-Time System," In Proceedings of USENIX 1st Mach Workshop, October 1990.

[11] Michael Beck, et al. Linux Kernel Internals second Edition, Addison-Wesley, 1998.

[12] R. W. Conway, W.L. Maxwell, and L. W. Miller. Theory of Scheduling. Addison-Wesley, 1967.

[13] Kieran Harty and David R. Cheriton. Application-Controlled Physical Memory using External Page-Cache Management. In the Fifth International Conference on Architectural Support for Programming Language and Operating Systems, pages 187-197, October 1992.

[14] Maeda, C. "A Metaobject Protocol For Accessing File Systems", In Proceedings of International Symposium on Object Technologies for Advanced Software, 1996.

[15] John K. Edwards and Pei Cao. User-Oriented Resource Scheduling in UNIX. Technical Report, Univ. of Wisconsin, Department of Computer Science, CS-TR-96-1318.

[16] Pei Cao, Edward W. Felten, and Kai Li, Application-Controlled File Caching Policies. Proceedings of the USENIX Summer 1994.