Evaluation of Process Scheduling Mechanism for a Web Server based on Its Behavior while Executing

Sukanya Suranauwarat Taniguchi Hideo Ushijima Kazuo
Graduate School of Information Science and Electrical Engineering,
Kyushu University
{sukanya,tani,ushijima}@csce.kyushu-u.ac.jp

Abstract

Traditional operating systems control the sharing of the processor resources among processes using a fixed scheduling policy based on the utilization of a computer system such as real-time or timesharing systems. Since the control over the processor allocation is based on a fixed policy, not based on processes' execution behavior, this can hinder an effective use of a processor or can extend the processing time of a process unnecessarily. Thus, we proposed a couple of process scheduling policies which respond to processes' execution behavior. One of these policies is the policy for improving a Web server's response time. This policy controls multiple processes of a Web server by adjusting the execution of these processes according to their predicted behavior. And we evaluated the performance of a Web server using this policy in simple cases.

In this paper, we evaluate the performance of a Web server when it is busy which is likely to be a realistic case. This could be the case in which it is most desirable to improve the response time of a Web server. Our experimental results show that the mean response times are improved greatly (up to 33.8% in the best case). They also show that the scheduling parameter is effectively predicted and updated by our mechanism based on the Web server's execution behavior.

1. Introduction

Traditional operating systems control the sharing of the processor resources among processes using a fixed scheduling policy based on the utilization of a computer system such as real-time or timesharing systems. A real-time system's scheduling policy must be able to analyze or handle data faster than they come in and it must also respond to time events. There are many applications in which computations must be completed before specified deadlines [4] and missing the specific deadlines are catastrophic. Therefore, scheduling such applications has been an important area of research in real-time system (e.g., [1]-[8]) . A timesharing system's scheduling policy is to provide good response to interactive users. Many commonly-used systems such as Unix, Mach, and Windows NT generally use conventional priority-based timesharing schedulers [9].

Since traditional schedulers control the execution of processes based on fixed policies, not based on processes' execution behavior, this can hinder an effective use of a processor or can extend the processing time of a process unnecessarily. For example, in timesharing system, if a process does not complete before its quantum (time-slice) expires, the processor is preempted and given to the next waiting process. Thus, a process that needs just a little bit more of processor time will not be completed until its next quantum. Because of this, the processing time and the context switching cost of the process increase unnecessarily. If we had predicted the process's execution behavior and delayed the process switch based on the predicted behavior that the process needs a little bit more of processor time to complete its job, then the extra costs mentioned above would have been avoided.

Therefore, we proposed the idea called POS (Program Oriented Schedule) [13]. The idea of POS is by increasing operating system ability to alter the program's execution behavior, the operating system could optimize program's execution behavior allowing user requirements (e.g., a performance enhancement) to be satisfied without making any changes to the existing program. In order to grasp a program's execution behavior, the idea of POS requires that the operating system has the ability to observe the execution of processes, to log the execution results, and to create/update the predicted behavior of the program based on the log.

One function in Windows 98, which users can get "faster program start up" as a performance enhancement [11], uses an idea similar to that of POS. The function improves the performance of a user's programs based on observing usage habits of programs. In other words, the function creates a log file to determine which programs user runs most frequently. All such frequently used files are then placed in a single location on the user's hard disk, which further reduces the time needed to start those programs [12].

We have already applied POS to the process scheduler [13, 14]. In [13], we proposed the process scheduling policy that controls the time-slice length of the object process in order to minimize its processing time and/or its context switching cost. However, the target programs of this policy are the programs that consist of only a single process. So, we extended our work to the programs composed of multiple processes such as servers. Server performance is crucial to client/server applications [10]. We used a Web server as a sample server and proposed the process scheduling policy for improving response time of a Web server [14]. We also evaluated the performance of a Web server when using this policy in simple cases, such as when the number of browsers accessing the Web server ranged from 1 to 3 and just two machines (client machine and server machine) were used.

In this paper, we evaluate the performance of a Web server when it is busy, i.e., when it is accessed by a lot of browsers at the same time which is likely to be a realistic case. This could be the case in which it is most desirable to improve the response time of a Web server. Our experimental results show that the mean response times are improved greatly (up to 33.8% in the best case). They also show that the scheduling parameter is effectively predicted and updated by our mechanism based on the Web server's execution behavior.

The rest of the paper is organized as follows. Section 2 gives an easy example to show how the performance of a program will be improved when POS is applied to the process scheduler. Section 3 briefly overviews how we observe and alter a Web server's execution behavior. Section 4 discusses the results of three quantitative experiments. Section 5 summarizes our conclusions.

2. An Efficient Example of POS

We used an easy example shown in Figure 1 to identify how performance will be improved when POS is applied to the process scheduler. In Figure 1, process A and process B need respectively 3.4 seconds and 2.1 seconds of processor time to accomplish their jobs. Both processes have the same priority and the time-slice is 1 second. Figure 1(a) shows the processing times of process A and process B when using a traditional priority-based timesharing scheduler. The processing times of process A and process B are 5.5 seconds and 4.1 seconds respectively. On the other hand, Figure 1(b) shows the processing times of process A and process B when POS is applied. Based on the predicted behavior that process B needs only 0.1 seconds more of processor time and then process B will finish its job, the process scheduler delays the process switch 0.1 seconds to allow process B to finish its job. Delaying the process switch causes the processing time of process B to be reduced to 3.1 seconds while that of process A is still the same as in Figure 1(a). Moreover, the context switching cost of process B also decreases.

This example also shows how the process scheduling policy, we proposed in [13], controls the time-slice length of the object process, and how the processing time and/or context switching cost of the object process will decrease when using this policy.

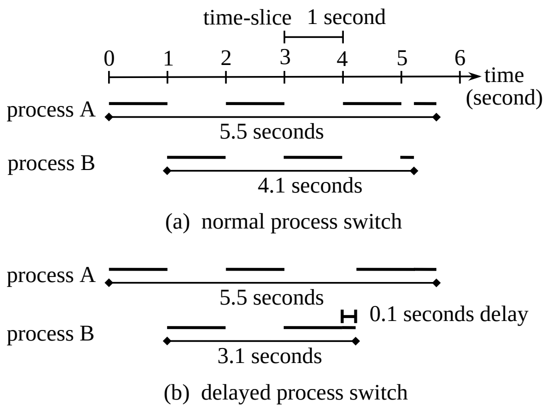


Figure 1. An Effcient Example of POS.

3. Overview

We observe the Web server's execution behavior, log the execution results, and create/update the predicted behavior called PFS (Program Flow Sequence) through the logging mechanism. And we alter the Web server's execution behavior by using the process control mechanism. Both mechanisms are integrated into the operating system.

3.1. logging mechanism

When a Web server is running, a log is collected recording the information necessary to determine the optimal execution. A log is a sequence of entries describing process identifier, process state and time. Then a sequence called PFS, which we use to predict behavior, is created or updated for each process. Note that a Web server is normally composed of multiple processes. PFS is a sequence of entries describing process state and time spent.

3.2. process control mechanism

For this paper, the content of a Web page is pretty simple, that is, one which is composed of text data and image data only. Text data and image data are separately saved in a file written in HTML (HTML file) and an image-formatted file (Image file) respectively. After making a request for a Web page, the browser will interpret and process the HTML file sent back by the Web server it requested and then display the text data. During the interpretation, if the Web page is also composed of image data, then the browser will request the Web server again for the Image file. When a Web site becomes busy (i.e., when a Web server is accessed by a lot of browsers), it takes time even for the text data which is normally much smaller than image data to show up on browsers. This kind of situation could cause users to give up waiting or to get tired of accessing such popular Web sites. So, while a Web site is busy, we thought that it is much better if the text data shows up on any browsers relatively faster. Therefore, we considered improving the time from requesting a Web page until text data displays (response time).

Most processes, except the currently executing process (i.e., process that is in the run state), are in one of two queues: a ready queue or a sleep queue. Processes that are waiting for the processor to become available (i.e., in the ready state) are placed on a ready queue, whereas processes that are blocked awaiting an event (i.e., in the wait state) are located on a sleep queue associated with the event. When a process is blocked awaiting an event to happen, if the resources (e.g., a hard disk) needed for the event are being used by any other process, then that process needs to wait first for those resources to become available. Next, that process needs to wait again for the operation (e.g., input/output) it initiated to be completed. By reducing the time waiting for the processor to become available at a ready queue or for the resource needed for an event to become available at a sleep queue, we can achieve enhanced response time. According to this, we proposed the policy that when a processor (a hard disk or a network communication) becomes bottlenecked, any server process handling an HTML file be moved to the head of the ready queue (sleep queue associated with the event) [14].

When we discussed the process control mechanism that implements the above policy focused on the bottleneck of the processor [14], we had two problems: how to detect which processes are server processes handling HTML files, and how to operate the ready queue.

To answer these questions we found that we needed to look at the detailed behavior of a Web server. We analyzed the Web server's processes behavior based on PFS and found out that any server process handling an HTML file has 2 characteristics: it runs after waiting for a long time in the wait state (characteristic 1) and it tends to cycle between run state and wait state fewer times than that of server process handling an Image file (characteristic 2).

To deal with the fist problem, we introduced two parameters into our process control mechanism in order to determine which processes are server processes handling HTML files: long wait threshold (its value is denoted by SLP) and run state/wait state threshold (its value is denoted by RW). If the time spent by a process in the wait state before moving to the run state is more than SLP, and the number of times the process changes between run state and wait state is less than RW, then we determine that it is an server process handling an HTML file. By these parameters, we can detect which process appears to be a server process handling an HTML file.

To deal with the second problem, our process control mechanism puts any process that has characteristic 1 at the head of ready queue and moves that process to the back of the ready queue when that process loses characteristic 2. The reason processes that lose characteristic 2 are moved to the back of the ready queue is that sometimes server processes handling Image files are mistaken as server processes handling HTML files because they exhibit characteristic 1.

How to predict and update SLP and RW is described below.

1. How to predict and update SLP:

   We analyzed the Web server's execution behavior based on PFS and found that a server process handling an HTML file is the process that waits for a request from a browser. The time it waits for a request is relatively long. Therefore, the longest time of each server process in the wait state is determined from PFS for each period, then SLP for the next time period is set to the smallest of these values. SLP is updated every time period.

2. How to predict and update RW:

   We analyzed the Web server's execution behavior based on PFS and found that the number of times a server process handling an HTML file or Image file changes between run state and wait state is proportional to the size of the HTML file and Image file respectively. In fact, the number of times a server process handling an Image file changes between run state and wait state is greater than that of a server process handling an HTML file, because in general Image files are bigger than HTML files. Therefore, the smallest number of times of each server process changing between run state and wait state is determined from PFS for each period, then RW for the next time period is set to the greatest of these values. RW is updated every time period.

4. Performance

In this section, we present experiments designed to evaluate the effectiveness of the process control mechanism we designed. We start with a description of the experimental setup, and proceed to present the results of three experiments.

4.1. Experimental Setup

The software used for the Web server and the browser in our experiment was Apache version 1.2.5 and Netscape Navigator version 3.04 respectively. The Web server ran on the personal computer with a 233 MHz AMD-K6 processor and 64 MB of memory, while browsers ran on three personal computers, each with a 200MHz Intel Pentium Pro processor and 64 MB of memory. All machines were running on BSD/OS version 2.1 in single user mode and were connected by a private 10Mb/s Ethernet. During the experiment, the operating system's I/O buffer cache in the server machine and each browser's cache were disabled in order to see the effect of our process control mechanism clearly.

The Web server was accessed by three browsers from each of the three machines at the same time. All browsers accessed unique URLs all of which have the same content. In three different experiments in which we varied RW in the range from 1 to 10, we measured the time (t1) from requesting a Web page until text data starts displaying and the time (t2) from requesting a Web page until image data displays completely for each access, and then found the mean of the 5 trial times of t1 (response time of text data) and t2 (response time of image data). In experiment 1, all the browsers accessed the Web server simultaneously every 30 seconds when the Web server coexisted with a processor-bound process and SLP was fixed at 20 seconds. The purpose of this experiment is to know how the response time of text data would be improved in the situation that is the best for our process control mechanism (i.e., the server machine's processor is bottlenecked [caused by a coexisting processor-bound process in this experiment] and SLP is set to be predicted 100% correctly). In experiment 2, all the browsers accessed the Web server randomly at the same time and SLP was fixed at 20 seconds, while in experiment 3, SLP was predicted and updated automatically based on PFS every 500 milliseconds. There was no processor-bound process in experiments 2 and 3. The purposes of experiments 2 and 3 are to know how the response times of text data in the more realistic case would be improved when SLP was fixed and when SLP dynamically varies according to the Web server's execution behavior respectively.

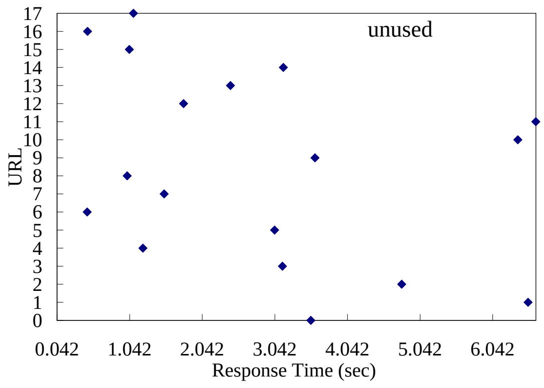
4.2. Experimental Result

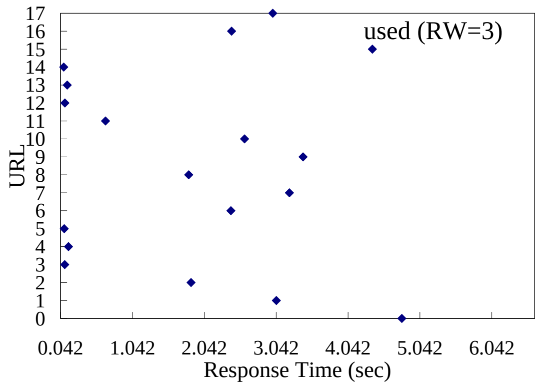
Figure 2 shows some examples of the results of experiment 1, when not using and using our process control mechanism (RW = 3,6,9). Figure 2 plots the URLs in numerical sequence on the y-axis against the response time of text data to a request in seconds. Figure 2 shows that the smallest and biggest response times when RW = 3,6,9 are better than when not using our process control mechanism. It also shows that the range or the distribution of response times becomes narrower when using our process control mechanism.

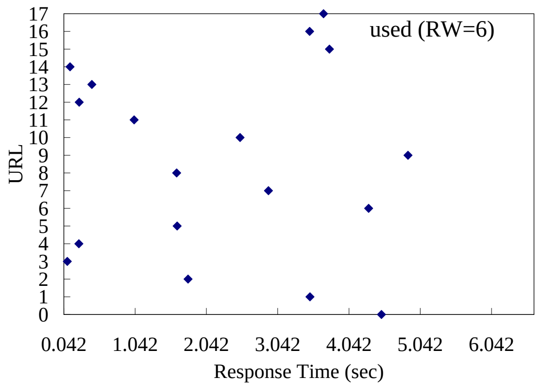
Figure 3(a) illustrates the mean and the range of response times of text data from Figure 2 into one graph. This figure shows that the mean response time of text data when RW = 3,6,9 are improved 33.8%, 21.3% and 26.6% respectively calculated by $\frac{a-b}{a} \times 100\%$ where a and b are the mean response times when not using and using our process control mechanism respectively. This case produced the best improvement of the response time of text data, because the coexisting processor-bound process always caused the processor to become bottlenecked and the server processes waiting for HTML files requests from browsers were always in the wait state at least 30 seconds which was more than SLP (20 seconds).

The rest of the experimental results will be shown like Figure 3(a).

Figure 3(b) illustrates the mean and the range of







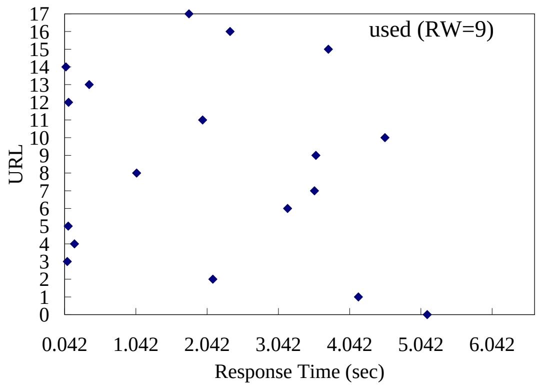
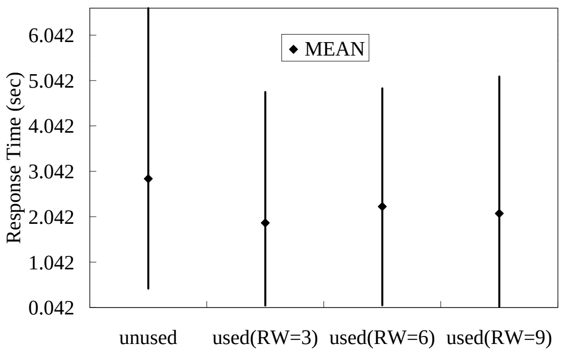
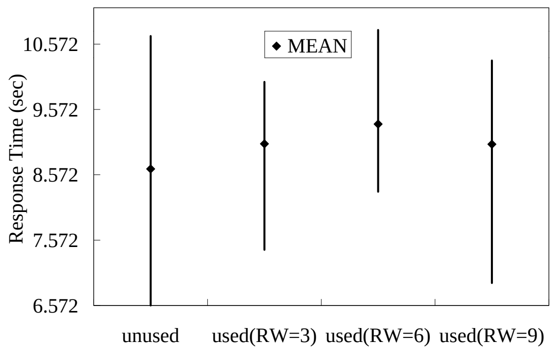


Figure 2. Response Time of Text Data in Experiment 1.

response times of image data to a request in seconds. This figure shows that the smallest response times when RW = 3,6,9 and the mean response times are not so fast as when not using our process control mechanism. This is expected and is due to our policy of giving processes handling HTML files priority over all other processes including server processes handling Image files.



(a) Response Time of Text Data

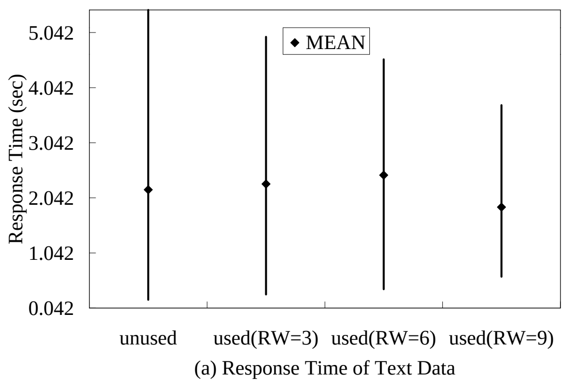


(b) Response Time of Image Data

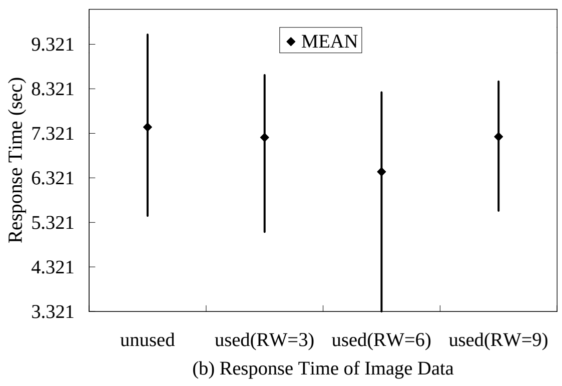
Figure 3. The Effect of Our Process Control Mechanism in Experiment 1.

Figure 4 shows the results of experiment 2, when not using and using our process control mechanism (RW = 3,6,9). Figures 4(a) and 4(b) illustrate the mean and the range of response times of text data and image data to a request in seconds respectively. We did not notice any consistent improvement when using our process control mechanism even though the mean response time of text data when RW = 9 is faster than when not using it. This is because the server processes waiting for HTML files requests from browsers might not be in the wait state more than SLP (20 seconds) because the browsers accessed the server randomly and SLP was fixed at 20 seconds. Therefore, fixed SLP is not responsive to the Web server's execution behavior.

Figure 5 shows the results of experiment 3, when not using and using our process control mechanism (RW = 3,6,9). Figures 5(a) and 5(b) illustrate the mean and the range of response times of text data and image data to a request in seconds respectively. Figure 5(a) shows that the smallest response times for RW = 6 and 9 are not better than when not using our process control mechanism. This could be because sometimes server processes handling Image files were mistaken as server processes handling HTML files when they were in the wait state more than predicted SLP. However, the distribution of response times is narrower and the mean response times of text data when RW = 3,6,9 are improved 22.2%, 17.9% and 6.7% respectively, because SLP was set and updated automatically every 500 milliseconds based on PFS (i.e., predicted SLP is responsive to the Web server's execution behavior). On the other hand, the mean response times of image data in Figure 5(b) are not so different from that when not using our process control mechanism as in experiment 1 (See Figure 3(b)) because of mistaking server processes handling Image files as server processes handling HTML files.



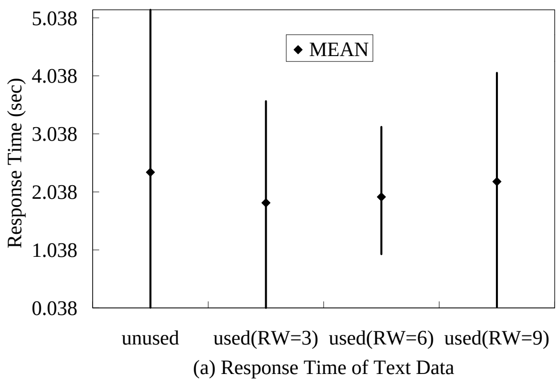
(a) Response Time of Text Data



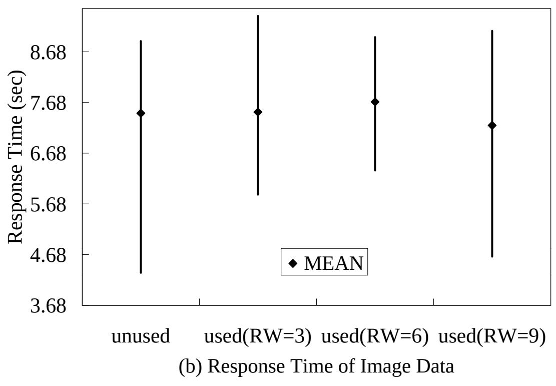
(b) Response Time of Image Data

Figure 4. The Effect of Our Process Control Mechanism in Experiment 2.

The results of experiment 1 show that the mean response times of text data are improve greatly (up to 33.8% when RW=3) when the processor of the server machine is bottlenecked which is the condition that our mechanism favors to and SLP is predicted 100% correctly. The results of experiment 2 shows that the mean response times of text data are not improved at all when SLP is not responsive to the Web server's execution behavior, while the mean response times of text data in experiment 3 are improved (up to 22.2% when RW=3) when SLP is responsive to the Web server's execution behavior. This means that SLP is effectively predicted and updated by our mechanism based on the Web server's execution behavior. Moreover, even though a processor-bound process causing the bottleneck of the server machine does not coexist in experiment 3, our mechanism still produces a good improvement. This might imply that a lot of accesses from browsers at the same time could cause the processor of the server machine to become bottlenecked.



(a) Response Time of Text Data



(b) Response Time of Image Data

Figure 5. The Effect of Our Process Control Mechanism in Experiment 3.

5. Conclusions

Since traditional schedulers control the execution of processes based on fixed policies, not based on processes' execution behavior, this can hinder an effective use of a processor or can extend the processing time of a process unnecessarily. Therefore, we proposed the idea called POS (Program Oriented Schedule). The idea of POS is by increasing operating system ability to alter the program's execution behavior, the operating system could optimize program's execution behavior allowing user requirements (e.g., a performance enhancement) to be satisfied. We have already applied this idea to the process scheduler and proposed a policy for improving a Web server's response time. We had also evaluated the performance of a Web server in simple cases [14].

In this paper, we evaluated the performance of a Web server when it is busy which is most likely to be a realistic situation and could be the case in which it is most desirable to improve the response time of a Web server. Our experimental results show that the mean response times are improve greatly (up to 33.8%) when the processor of the server machine is bottlenecked caused by a coexisting processor-bound process, and scheduling parameter SLP is predicted 100% correctly. They also show that even though a processor-bound process does not coexist, our scheduling mechanism still produces a good improvement (up to 22.2%) when scheduling parameter SLP is predicted and updated automatically by our mechanism based on the Web server's execution behavior. This means that scheduling parameter SLP is effectively predicted and updated by our mechanism based on the Web server's execution behavior.

Future work is required to evaluate the performance of a Web server when scheduling parameter RW is automatically predicted and updated by our mechanism and also when both scheduling parameter SLP and RW are automatically predicted and updated by our mechanism. It is important to evaluate our process scheduling mechanism with a more complex Web environment such as a complicated Web page (consisting of several images, sound, etc.). Also, the effectiveness of our mechanism needs to be evaluated with regard to processor load and number of accesses from browsers.

Acknowledgements

We would like to thank James Michael Perry for his assistance in proofreading this paper.

References

[1] K. Ramamritham, J.A. Stankovic, and P. Shiah: "Efficient Scheduling Algorithms for Real-Time Multiprocessor Systems," *IEEE Trans. Parallel and Dist. Systems*, Vol. 1, No. 2, pp. 184–194, April 1990.

[2] T. Shepard and J.A.M. Gagné: "A Pre-Run-Time Scheduling Algorithm For Hard Real-Time Systems," *IEEE Trans. Software. Eng.*, Vol. 17, No. 7, pp. 669–677, July 1991.

[3] K. Schwan and H. Zhou: "Dynamic Scheduling of Hard Real-Time Tasks and Real-Time Threads," *IEEE Trans. Software. Eng.*, Vol. 18, No. 8, pp. 736–748, August 1992.

[4] J. Xu and D.L. Parnas: "On Satisfying Timing Constraints in Hard-Real-Time Systems," *IEEE Trans. Software. Eng.*, Vol. 19, No. 1, pp. 70–84, January 1993.

[5] W.K. Shih, J.W.S. Liu, C.L. liu: "Modified Rate-Monotonic Algorithm for Scheduling Periodic Jobs with Deferred Deadlines," *IEEE Trans. Software. Eng.*, Vol. 19, No. 12, pp. 1171–1179, December 1993.

[6] M.G.Härbour, M.H.Klein, and J.P.Lehoczky: "Timing Analysis for Fixed-Priority Scheduling of Hard Real-Time Systems," *IEEE Trans. Software. Eng.*, Vol. 20, No. 1, pp. 13–28, January 1994.

[7] A. Burns, K. Tindell, and A. Wellings: "Effective Analysis for Engineering Real-Time Fixed Priority Schedulers," *IEEE Trans. Software. Eng.*, Vol. 21, No. 5, pp. 475–479, May 1995.

[8] W. Feng and J.W.S. Liu: "Algorithms for Scheduling Real-Time Tasks with Input Error and End-to-End Deadlines," *IEEE Trans. Software. Eng.*, Vol. 23, No. 2, pp. 93–106, February 1997.

[9] B. Ford and S. Susarla: "CPU Inheritance Scheduling," In *proceedings of the Second Symposium on Operating Systems Design and Implementation*, pp. 91–106, October 1996.

[10] M.F. Kaashoek, D.R. Engler, G.R. Ganger, H.M. Briceñõ, R. Hunt, D. Mazierès, T. Pinckney, R. Grimm, J. Jannotti, and K. Mackenzie: "Application Performance and Flexibility on Exokernel Systems," In *proceedings of the sixteenth Symposium on Operating Systems Principles*, October 1997.

[11] http://www.microsoft.com/Windows98/guide/Win98/Features/Faster.asp

[12] http://www.microsoft.com/Windows98/usingwindows/maintaining/articles/811Nov/MNTfoundation2a.asp

[13] H. Taniguchi: "POS:Program Oriented Schedule," *IPS Japan Proc. of Computer System Symposium'96*, Vol.96, No.7, pp.123-130, 1996 (in Japanese).

[14] S. Suranauwarat and H. Taniguchi: "Process Scheduling Policy for a WWW Server Based on its Contents," *Trans. IPS Japan*, Vol. 40, No. 6, pp. 2510–2522, 1999 (in Japanese).