An Efficient Semaphore Implementation Scheme for an Event

Bong-sik Sihn[1], Ki-Hee Han[2] and Jong-Wha Chong[2]
[1] Department of Electronic Engineering, Han-Yang University
17 Haengdang-Dong, Sungdong-Gu, Seoul 133-070, Korea
Tel. +82-2-2290-0558, Fax.: +82-2-2290-1886
[2] Department of Information & Communication, Han-Yang University
17 Haengdang-Dong, Sungdong-Gu, Seoul 133-070, Korea
e-mail : sbs69@korea.com, hanq73@empal.com

Abstract: In this paper, we present a novel efficient semaphore implementation scheme which diminishes completion time of high priority tasks and improves reliability of a system. The real-time system is constrained to complete their tasks in time. Especially, the task of a hard real-time system must meet its deadline under unfavorable conditions. In this paper, the number and sort of the locked semaphores, when an event occurred, decide whether the context switch should occur or not, so higher priority tasks diminish in their completion time. The experimental results show that the proposed method gives performance improvements in finish time of high priority tasks of about 11% over the Zuberi.

1. Introduction

Real-time system has time constraint such as deadline, release time, period, etc. Especially, Hard real-time system must complete a task within its deadline.[1] Real-time systems today are much smaller and simpler systems such as in automatic control, cellular phones, and home electronics(camcorders, TVs, and VCRs).

When tasks are willing to share resources, semaphore is implemented mutually exclusively to synchronize relative tasks.[2][3][4][5][6][7] Because semaphore system calls are invoked every time an task releases or resumes out of waiting stage, it becomes essential that the Real-Time OS provide efficient, low-overhead semaphores.

Most researches in the area of reducing the semaphore overheads have inefficient characteristics in time schedules. Semaphore implementations of Zuberi[4] result in priority inversion. Then, a higher priority tasks wait for lower priority task's releasing the semaphore.

In this paper we device a mechanism to reduce the completion time of the higher priority of tasks. We apply this time reduction technique when an event occurs.

2. Conventional Semaphore Schemes

If the semaphore happens to be already locked by some other task, the task making the semaphore lock system call is put on a wait queue and is blocked. It is unblocked due to the semaphore release operation and it then proceeds to reserve the semaphore for itself.

If the caller is blocked by a semaphore, priority inheritance takes place under which the current lock holder task's priority is increased to that of the caller task. This is needed to avoid unbounded priority inversion.

Zuberi[4] proposed an efficient scheme of semaphore Implementation for small-Memory embedded systems which saves one context switch per semaphore lock operation in most circumstances.

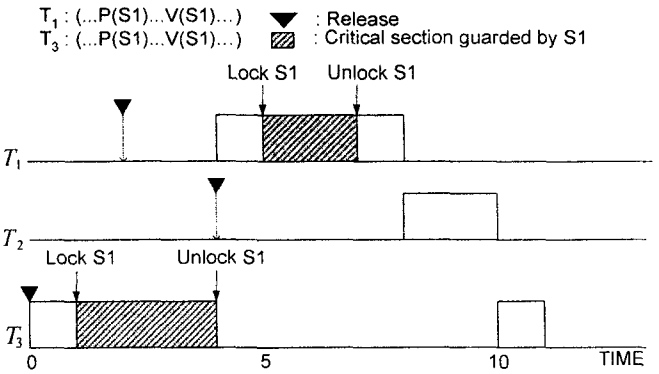

Figure 1. Zuberi's semaphore scheme(1)

Zuberi's scheme is as follows(Figure 1): at first, T3 is released; T3 acquires semaphore S1; when T1 with code which intends to lock semaphore S1 is released, the OS checks if S1 is nessary for T1 or not; If S1 is necessary, then T3 keeps executing instead of context switch to T1; context switch to T1 is made after unlocking S1. As a result, on context switch is eliminated.

First of all, notice that if T1 attempt to lock only S1 semaphore, then the semaphore lock scheme is efficient due to one less context switch. In fact, for this case, the completion time is not delayed.

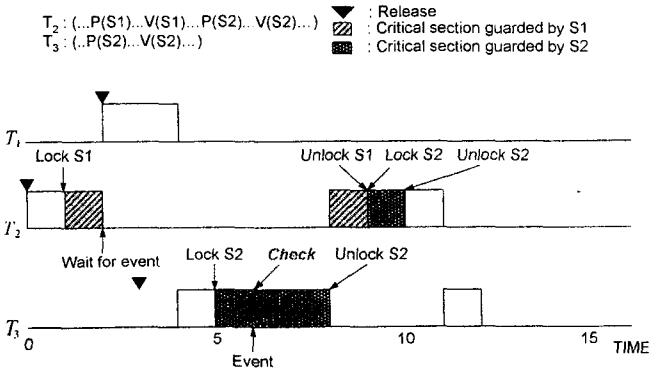

Figure 2. Zuberi's semaphore scheme(2)

The situation is very different when T1 is willing to lock two or more semaphore including S1. Figure 2 shows a typical scenario for this situation. T2 is released; T2 acquires semaphore S1; T2 blocks to wait for an event such as a message arrival; some other tasks execute; T1 is

released; when T1 attempts to lock S1, T1 is blocked, and T3 is executed; T3 acquires semaphore S2; when the event waited for by T2 arrives, the OS checks if S2 is necessary for T2 or not; As S2 is necessary, T3 keeps executing instead of context switch to T2; context switch to T2 is made after unlocking S2; when S2 is unlocked, T1 is executed; As a result, T1 is completed later than traditional semaphore scheme.

3. Proposed Semaphore Scheme

We observe that zuberi's scheme is responsible for late completion time of higher priority tasks. So, we propose a novel semaphore implementation scheme which reduce completion time of higher priority tasks rather than Zuberi's scheme.

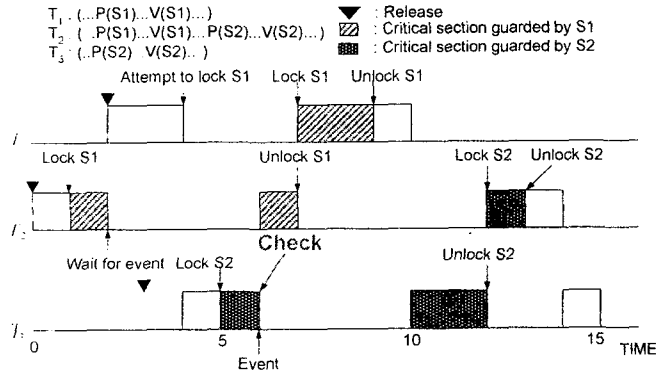

Figure 3. Proposed semaphore scheme

the proposed scheme is as follows(Figure 3): T2 is released; T2 acquires semaphore S1; T2 blocks to wait for an event such as a message arrival; some other tasks execute; T1 is released; when T1 attempts to lock S1, T1 is blocked, and T3 is executed; T3 acquires semaphore S2; when the event waited for by T2 arrives, the OS checks some cases as depicted in Figure 4; only if S2 is necessary for T2 and the number of semaphores necessary for T2 is only one, T3 keeps executing instead of context switch to T2: Otherwise, context switch is made right here to T2; So, if S2 is necessary for T2 and any other semaphore is not necessary for T2, T1 is completed relatively earlier than Zuberi's method.

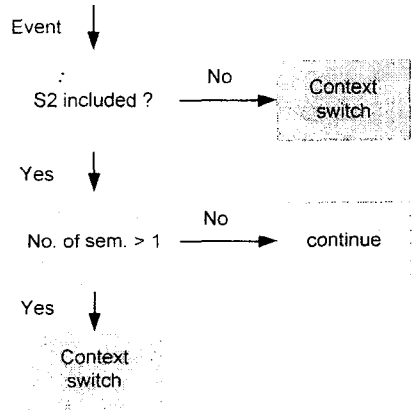

Figure 4. Flow chart of proposed scheme

4. Experimental Results

To measure the improvements in performance resulting from our new semaphore implementation scheme, we simulate it under SUN SPARC60. The time interval between each task block is pseudo random.

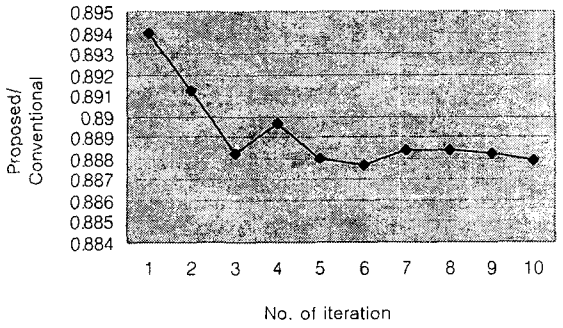

Figure 5. Experimental results

The experimental results in the Figure 5 show that the proposed method gives performance improvements in completion time of high priority tasks of about 11 % over Zuberi's scheme.

5. Conclusions and Future Works

In this paper, we presented a new efficient semaphore implementation scheme which reduces completion time of high priority tasks by about 11% and improves reliability of a system.

Future works include the advantages and disadvantages of extending our scheme to multi-processors. In the future, we plan to investigate adopting the proposed scheme to multi-processor systems.

References

[1] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard real-time environment," *Journal of the ACM*, vol. 20, no. 1, pp. 46-61, 1973

[2] L. Sha, R. Rajkumar and J. Lehoczky, "Priority inheritance protocols : an approach to real-time synchronization," *IEEE Trans. on Computer*, vol. 39, no.3, pp.1175-1198, 1990

[3] J. B. Goodenough and L. Sha, "The priority ceiling protocol : A method for minimizing the blocking of high priority Ada tasks," in *Proc. 2 nd ACM Int. Workshop Real-Time Ada Issues*, 1988

[4] K. M. Zuberi and K. G. Shin, "An efficient semaphore implementation scheme for small-memory embedded systems," *IEEE Trans. on Computer*, pp.25-34, 1997

[5] C. D. Wang, H. Takada and K. Sakamura, "Priority inheritance spin locks for multiprocessor real-time systems," in *2 nd International Symposium on Parallel Architectures, Algorithms, and Networks*, pp. 70-76, 1996

[6] H. Takada and K. Sakamura, "Experimental implementations of priority inheritance semaphore on ITRON-specification kernel," in *11th TRON Project International Symposium*, pp. 106-113, 1994

[7] A. Terrasa, A. Garcia-Fornes, "Real-Time Synchronization between Hard and Soft Tasks in RT-Linux," *IEEE Trans. On Real-time Computing Systems and Applications*, pp. 303-310, 1999