# Real-Time Synchronization Protocols with Abortable Critical Sections

Hiroaki Takada and Ken Sakamura

Department of Information Science,
Faculty of Science, University of Tokyo
7-3-1, Hongo, Bunkyo-ku, Tokyo 113, Japan

## Abstract

*Making critical sections abortable is a promising approach to reducing priority inversions. To improve the schedulability of a system using abortable critical sections, the maximum number of abortions should be decreased. In this paper, we propose a real-time synchronization protocol named the* ceiling abort protocol (CAP) *which introduces a new priority-based abort scheme to the priority ceiling protocol. Our proposed protocol can make the maximum number of abortions smaller than with the conventional priority abort scheme. A method to determine an upper bound on the number of abortions under the CAP is presented, and a schedulability analysis of the protocol is illustrated. Some extensions of the CAP are also discussed.*

## 1 Introduction

Priority inversion, the state in which a higher priority job is blocked by lower priority jobs due to resource sharing, is a major cause to degrade schedulability in hard real-time systems.

Making critical sections abortable is a promising approach to reducing priority inversions. Some real-time synchronization protocols using abortable critical sections have been proposed and evaluated [1, 2, 3]. All of these protocols adopt the priority abort scheme [1], in which a critical section can be aborted by any job that has a higher priority than the job executing the critical section. However, the priority abort scheme has a drawback in that it can sometimes cause unnecessary abortions thus degrading the schedulability of a system.

In this paper, we propose a real-time synchronization protocol named the *ceiling abort protocol* (CAP) which introduces an improved priority-based abort scheme called the *ceiling abort scheme* to the priority ceiling protocol (PCP) [4]. With this scheme, a critical section is divided into two segments and a priority ceiling is assigned to each. The critical section can be aborted by jobs that have a higher priority than the priority ceiling of the currently executed segment. A job that has a lower or the same priority as the priority ceiling is blocked, when it tries to enter a critical section. The executed critical section inherits the priority of the blocked job when necessary.

To guarantee the schedulability of a system using abortable critical sections, the maximum number of abortions must be determined. We will present a method to determine an upper bound on the number of abortions under the CAP. A schedulability analysis of the protocol is illustrated using an example, and its effectiveness is demonstrated. We also present some extensions of the CAP that can reduce the maximum number of abortions.

## 2 Assumptions and Notations

A system consists of $n$ periodic tasks $\tau_1, \tau_2, \cdots, \tau_n$. Each task has a *priority* determined by the rate monotonic scheduling algorithm. $T_i$ and $P_i$ denote the period and priority of task $\tau_i$ respectively, and $\tau_1, \tau_2, \cdots, \tau_n$ are sorted in descending order of priority. Hence, $T_i \leq T_j$ and $P_i \geq P_j$ if $i \leq j$.

An execution of a task is called a *job*. $\tau_i$ also denotes a job requested by task $\tau_i$. Each task requests a job at the beginning of each period, and the job's deadline is at the end of the period. The priority of a job is set to the priority of the requesting task when it starts, and is changed during its execution. The priority of a job at any given moment is called its *current priority* or simply its *priority*.

Each job is preemptive and executed according to the priority-driven scheduling. The highest priority job that is ready to run is executed first. Jobs with the same priority are scheduled in a FCFS order. When the execution of a job is delayed due to the execution of other jobs of higher or the same priority tasks, the job is said to be *preempted*.

Each task includes critical sections guarded by binary semaphores. The $j$-th critical section in task $\tau_i$ is represented as $z_{i,j}$. When a job is preempted during a critical section, the critical section is said to be preempted. Two critical sections included in a task must be either disjoint or properly nested. Job $\tau_i$ is called *blocked*, when the execution of $\tau_i$ is delayed by a semaphore locked by a job of lower priority tasks.

The maximum processing time of job $\tau_i$ is denoted as $C_i$ and the maximum blocking time during its execution is denoted as $B_i$. The execution of a job is not delayed due to reasons other than being preempted or blocked. When

the schedulability of a system is discussed, we ignore the overheads of task scheduling, context switches, and other processing needed for task synchronization.

## 3 Abortable Critical Sections

An abortable critical section consists of an abortable segment followed by an unabortable segment. When a job is executed within the abortable segment, the execution of the critical section may be aborted and restarted from the beginning. Once the job enters the unabortable segment, the critical section is not aborted and is executed to the end.

The maximum processing times of the abortable segment and the unabortable segment of $z_{i,j}$ are denoted as $A_{i,j}$ and $U_{i,j}$ respectively. Though the maximum processing time of the entire critical section is less than or equal to $A_{i,j} + U_{i,j}$ in general, we assume that it is equal to $A_{i,j} + U_{i,j}$ in this paper. Also, we ignore various overheads when aborting a critical section for simplicity. In the following, we refer to an abortable critical section as defined above simply as a critical section. A critical section $z_{i,j}$ is called unabortable when $A_{i,j} = 0$.

If abortable critical sections are nested arbitrarily, evaluation of the maximum processing time of the outer critical section becomes complicated. Thus, when $z_{i,j}$ and $z_{i,j'}$ are nested, we allow only the following two cases ($z_{i,j'}$ is assumed to be the outer critical section, here).

- $z_{i,j}$ is unabortable and is included in the unabortable segment of $z_{i,j'}$.
- The abortable segment of $z_{i,j}$ corresponds to that of $z_{i,j'}$ and the unabortable segment of $z_{i,j}$ is included in that of $z_{i,j'}$. When one of the critical sections is aborted, the other is regarded to also be aborted.

As a result of these simplifications, when $z_{i,j}$ is aborted $m$ times during its execution, the maximum processing time of its abortable segment is prolonged by $mA_{i,j}$. The maximum duration that $z_{i,j}$ can block a higher priority job that can abort $z_{i,j}$ is reduced to $U_{i,j}$.

## 4 The Ceiling Abort Protocol

With the ceiling abort scheme, the priority ceiling assigned to the abortable segment of a critical section differs from that assigned to its unabortable segment. The priority ceiling of the abortable segment of $z_{i,j}$ and that of its unabortable segment are represented as $P^A_{i,j}$ and $P^U_{i,j}$ respectively.

**Definition 1 (Priority Ceilings)** For each critical section $z_{i,j}$, $P^A_{i,j}$ and $P^U_{i,j}$ are determined as follows.

- $P^U_{i,j}$ is equal to the highest priority task that may lock the semaphore guarding $z_{i,j}$.
- $P^A_{i,j}$ should be determined to be an arbitrary priority which is lower than $P^U_{i,j}$ and higher than or equal to $P_i$[1]. □

[1]When $z_{i,j}$ is included in another critical section $z_{i,j'}$, $P^A_{i,j}$ must be higher than or equal to $P^A_{i,j'}$.

Note that $P^U_{i,j}$ is the same as the priority ceiling of $z_{i,j}$ determined in the PCP.

With the ceiling abort scheme, a critical section $z_{i,j}$ is aborted in its abortable segment, when the semaphore guarding $z_{i,j}$ is requested by a job with a higher priority than $P^A_{i,j}$. We introduce this scheme to the PCP and define the CAP as presented below.

**Definition 2 (The Ceiling Abort Protocol)** When a job $\tau_i$ tries to enter a critical section $z_{i,j}$, $\tau_i$ must follow the following steps.

1. If any critical sections are preempted with priority ceilings higher than the priority of $\tau_i$, $\tau_i$ is blocked by the critical sections. The job executing these critical sections[2] inherits the priority of $\tau_i$ and is executed first. When the job finishes the execution of the critical sections and releases the semaphore, its priority is reset to the priority before the inheritance and the execution of $\tau_i$ resumes.
2. If the semaphore guarding $z_{i,j}$ is locked by another job, all the critical sections that the job is executing are aborted.
3. $\tau_i$ locks the semaphore guarding $z_{i,j}$ and begins to execute $z_{i,j}$.

When $z_{i,j}$ is aborted during the execution of its abortable segment, the semaphore guarding $z_{i,j}$ is released and the priority of $\tau_i$ is reset to the priority it had before entering $z_{i,j}$. Then, $\tau_i$ tries to reenter $z_{i,j}$ from Step 1. □

When all critical sections are unabortable, the semaphore guarding $z_{i,j}$ is never locked by another job in Step 2, and the behavior of the CAP corresponds to that of the PCP. In this sense, the CAP is an extension of the PCP.

The CAP inherits the important properties of the PCP presented in the following theorems [4].

**Theorem 3** The CAP prevents deadlocks. □

**Theorem 4** When the CAP is used, a job is blocked at most for the duration of one execution of the critical sections or their unabortable segments that can possibly block the job. □

In other words, each job is blocked no more than once. Therefore, $B_k$ can be determined by calculating the maximum processing time of the critical sections or their unabortable segments that can possibly block $\tau_k$. $\tau_k$ can be blocked by a critical section $z_{i,j}$ included in a lower priority task, if $\tau_k$ locks the semaphore guarding $z_{i,j}$ and if $P^A_{i,j}$ is higher than or equal to $P_k$. $\tau_k$ can be blocked by the unabortable segment of $z_{i,j}$, if $\tau_k$ locks the semaphore guarding $z_{i,j}$ and if $P^U_{i,j}$ is higher than or equal to $P_k$.

In the CAP, the priority of a job is changed according to the priority inheritance policy. Another protocol adopting

[2]It is proved that all of these critical sections belong to a job.

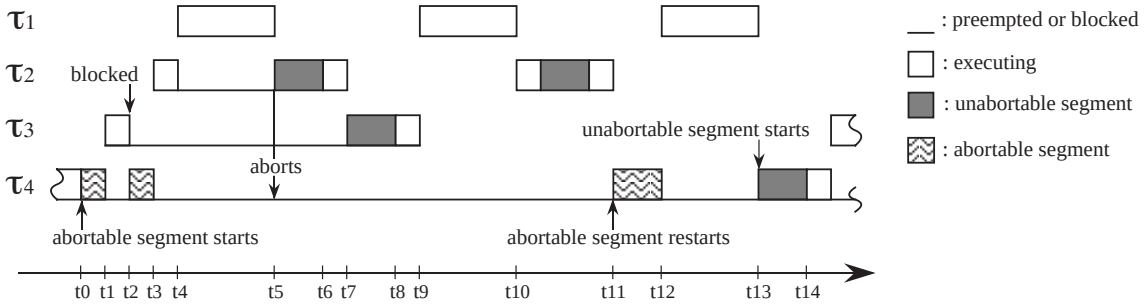

Figure 1: Scheduling Example under the CAP

| | $C_i$ | $T_i$ | $P_i$ | $A_{i,1}$ [$P^A_{i,1}$] | $U_{i,1}$ [$P^U_{i,1}$] |
|---|---|---|---|---|---|
| $\tau_1$ | 4 | 10 | $P_1$ | - | - |
| $\tau_2$ | 4 | 15 | $P_2$ | 0 | 2 [$P_2$] |
| $\tau_3$ | 4 | 30 | $P_3$ | 0 | 2 [$P_2$] |
| $\tau_4$ | 10 | 100 | $P_4$ | 2 [$P_3$] | 2 [$P_2$] |

Table 1: Task Set for Example 5

the stack resource policy (SRP) [5] instead is possible and has the same properties as the CAP presented above. It is also possible to apply the SRP to either the abortable segment only or to the unabortable segment only.

**Example 5 (Scheduling under the CAP)** We present an example of task scheduling using the CAP. An example task set consisting of four tasks is presented in Table 1. Here, $\tau_2$, $\tau_3$, and $\tau_4$ include critical sections $z_{2,1}$, $z_{3,1}$, and $z_{4,1}$ respectively. These critical sections are guarded by a binary semaphore. $z_{4,1}$ is abortable, while $z_{2,1}$ and $z_{3,1}$ are not. The value in the brackets represents the priority ceiling of each segment.

Figure 1 illustrates an example scheduling of the tasks. We will describe each event in the figure.

- At time $t_0$, $\tau_4$ enters the abortable segment of $z_{4,1}$.
- At time $t_1$, $\tau_3$ is requested and begins to execute.
- At time $t_2$, $\tau_3$ tries to enter $z_{3,1}$. Because $z_{4,1}$ is preempted in the abortable segment and its priority ceiling is $P_3$, $\tau_3$ is blocked by $z_{4,1}$. Now $\tau_4$ inherits the priority of $\tau_3$ and continues to execute the abortable segment.
- At time $t_3$, $\tau_2$ is requested and begins to execute, just before $\tau_4$ finishes the execution of the abortable segment.
- At time $t_5$, $\tau_2$ tries to enter $z_{2,1}$. Because the semaphore is locked by $\tau_4$, $\tau_2$ cannot enter $z_{2,1}$ immediately. As the priority of $\tau_2$ is higher than $P^A_{4,1}$ ($= P_3$), $z_{4,1}$ is aborted and $\tau_2$ begins to execute $z_{2,1}$. The priority of $\tau_4$ is reset to $P_4$ by the abortion.
- At time $t_7$, the execution of $\tau_2$ is finished and $\tau_3$ resumes execution. As $z_{4,1}$ has been aborted, $\tau_3$ can enter $z_{3,1}$.
- At time $t_{11}$, $\tau_4$ resumes execution. As $z_{4,1}$ has been aborted, $\tau_4$ restarts $z_{4,1}$ from the beginning.
- At time $t_{13}$, $\tau_4$ enters the unabortable segment of $z_{4,1}$.
- At time $t_{14}$, $\tau_4$ finishes the execution of $z_{4,1}$.

In this example, every job meets its deadline.

## 5 Schedulability Analysis of the CAP

To analyze the schedulability of a system using the CAP, we can apply the results of the PCP presented below [4]. $\mathcal{P}_i$ in the following theorems denotes the index set of the tasks whose priority is higher than or equal to $P_i$, i.e. $\mathcal{P}_i = \{\ j \mid j = 1, \cdots, n,\ and\ P_j \geq P_i\ \}$.

**Theorem 6** A set of $n$ periodic tasks using the PCP is schedulable by the rate monotonic scheduling algorithm, if

$$\forall i, 1 \leq i \leq n, \quad \sum_{r \in \mathcal{P}_i} \frac{C_r}{T_r} + \frac{B_i}{T_i} \leq i(2^{1/i} - 1). \qquad \square$$

**Theorem 7** A set of $n$ periodic tasks using the PCP is schedulable by the rate monotonic scheduling algorithm for all task phasings, if and only if

$$\forall i, 1 \leq i \leq n,$$
$$L_i = \max_{(k,l) \in \mathcal{R}_i} \left[ lT_k - \sum_{r \in \mathcal{P}_i} C_r \left\lceil \frac{lT_k}{T_r} \right\rceil - B_i \right] \geq 0,$$

where $\mathcal{R}_i = \{(k,l) \mid k \in \mathcal{P}_i,\ l = 1, \cdots, \lfloor T_i/T_k \rfloor\}$. $\square$

The latter theorem presents the necessary and sufficient condition, when $C_i$ is equal to the maximum processing time of $\tau_i$ and $B_i$ is equal to its maximum blocking duration. When larger values are used, this theorem provides a sufficient condition for the schedulability. We call $L_i$ the *schedulable laxity* of $\tau_i$.

Next, we examine the effects of abortions on each parameter that appears in the theorems. By making a critical section $z_{i,j}$ abortable, the effects on the schedulability of $\tau_k$ can be examined for each of the following four cases. The maximum number of times that $z_{i,j}$ is aborted is represented as $m_{i,j}$ below.

- When $P_k > P^U_{i,j}$, making $z_{i,j}$ abortable has no effect on any parameter of $\tau_k$.

- When $P^A_{i,j} < P_k \leq P^U_{i,j}$, the maximum duration that $\tau_k$ can be blocked by $z_{i,j}$ is reduced to $U_{i,j}$. Consequently, $B_k$ can possibly be decreased by making $z_{i,j}$ abortable.
- When $P_i < P_k \leq P^A_{i,j}$, the maximum duration that $\tau_k$ can be blocked by $z_{i,j}$ remains $A_{i,j} + U_{i,j}$. Therefore, making $z_{i,j}$ abortable has no effect on any parameter of $\tau_k$.
- When $P_k \leq P_i$, the schedulability of $\tau_k$ may be affected by the increase of $C_i$. $C_i$ is prolonged by $m_{i,j} A_{i,j}$ because one abortion of $z_{i,j}$ requires additional execution time for $A_{i,j}$. $B_k$ remains unchanged.

From these examinations, if $m_{i,j}$ can be determined for each $i$ and $j$, the schedulability of the system using the CAP by the rate monotonic scheduling algorithm can be checked using Theorem 6 or 7.

The following theorem presents a method to determine an upper bound of $m_{i,j}$ when the CAP is used. $\mathcal{Z}_{i,j}$ denotes the index set of the tasks which can possibly abort $z_{i,j}$.

**Theorem 8** $m_{i,j}$, the maximum number of times that $z_{i,j}$ is aborted when a set of $n$ periodic tasks using the CAP is scheduled by the rate monotonic scheduling algorithm, is less than or equal to $m$ that satisfies the following inequality.

$$\max_{(k,l)\in\mathcal{R}_{i,j,m}} \left[ lT_k - \sum_{r\in\mathcal{Q}_i} C_r \left\lceil \frac{lT_k}{T_r} \right\rceil \right] \geq (m+1)A_{i,j}, \quad \cdots (1)$$

where $\mathcal{Q}_i = \{\ r \mid r = 1, \cdots, n,\ and\ P_r > P_i\ \}$
and $\mathcal{R}_{i,j,m} = \{(k,l) \mid k \in \mathcal{Q}_i,\ l = 0, 1, \cdots, \lfloor T_i/T_k \rfloor,$
$and\ \sum_{r\in\mathcal{Z}_{i,j}} \lceil lT_k/T_r \rceil \leq m\}$. □

The notion of worst-case phasing [6] is used to prove this theorem. The worst-case phasing for a task is the system status in which the time until the termination of its job is the longest possible. Under static priority assignments, the worst-case phasing for a task $\tau_i$ has been proved to be the instant when every task with a higher or the same priority with $\tau_i$ requests a job [7]. This instant is called a critical instant, and is determined independently of the execution time of $\tau_i$. From this property, the execution of the abortable segment of $z_{i,j}$ finishes latest, when a critical instant occurs immediately after $\tau_i$ enters the abortable segment of $z_{i,j}$.

Another necessary property to prove this theorem is that $\tau_i$ is not blocked within $z_{i,j}$ once the execution of $z_{i,j}$ begins. This property holds under the CAP as well as under the PCP.

As a result, the left side of (1) represents the minimum processing time given to $z_{i,j}$, while the jobs that can possibly abort $z_{i,j}$ are executed in $m$ times or less. The right side of (1) represents the maximum processing time of the abortable segment, when $z_{i,j}$ is aborted $m$ times. Therefore, if (1) is met, the execution of the abortable segment will be finished despite the abortions.

Because the left side of (1) stops increasing when $m \geq \sum_{r\in\mathcal{Z}_{i,j}} \lceil T_i/T_r \rceil$, the minimum value of $m$ satisfying (1), if any, can be obtained by checking (1) by incrementing $m$ from one by one.

| $m$ | LS of (1) | RS of (1) |
|---|---|---|
| 1 | 0 | 4 |
| 2 | 6 | 6 |
| 3 | 6 | 8 |
| 4 | 12 | 10 |

Table 2: Evaluation of an Upper Bound of $m_{4,1}$

| | $C_i$ | $T_i$ | $P_i$ | $A_{i,1}$ | $U_{i,1}$ | $B_i$ | $C^+_i$ | $L_i$ |
|---|---|---|---|---|---|---|---|---|
| $\tau_1$ | 4 | 10 | $P_1$ | - | - | 0 | 0 | 6 |
| $\tau_2$ | 4 | 15 | $P_2$ | 0 | 2 [$P_2$] | 2 | 0 | 1 |
| $\tau_3$ | 4 | 30 | $P_3$ | 0 | 2 [$P_2$] | 4 | 0 | 2 |
| $\tau_4$ | 10 | 100 | $P_4$ | 2 [$P_3$] | 2 [$P_2$] | 0 | 4 | 4 |

Table 3: Schedulability Analysis Example (Case 1)

The upper bound of $m_{i,j}$ calculated with this method is a pessimistic bound. Accordingly, using the parameters calculated from this upper bound, Theorem 7 gives a sufficient condition for the schedulability.

**Example 9 (Schedulability Analysis of the CAP)** Using the method described above, we illustrate a schedulability analysis of the task set of Example 5. First, we calculate an upper bound of $m_{4,1}$ using Theorem 8. In this task set, only $\tau_2$ can abort $z_{4,1}$, i.e. $\mathcal{Z}_{4,1} = \{2\}$. Table 2 presents the value of the left and right sides of (1) by incrementing $m$ from one by one. From this table, we can see that (1) is first met when $m = 2$. Figure 2 illustrates the scheduling of each task, when $\tau_1$, $\tau_2$ and $\tau_3$ are requested at time $t_0$, which is the worst-case phasing for $\tau_4$. Until time $t_1$, $z_{4,1}$ can be aborted no more than twice, while the processing time given to $\tau_4$ is sufficient to finish the execution of the abortable segment despite the abortions.

The result of calculating the effects on the schedulability of each task is presented in Table 3. In this table, $C^+_i$ denotes the additional execution time required for $\tau_i$ when critical sections are made abortable. Since $m_{4,1}$ is known to be two or less, an upper bound of $C^+_4$ is $2A_{4,1}(= 4)$. $L_i$ in the table is the schedulable laxity of $\tau_i$ as defined in Theorem 7. Because no $L_i$ is negative in this table, this task set can be scheduled with the CAP.

To demonstrate the validity of the CAP, we present the results of schedulability analyses using conventional approaches. The center columns of Table 4 show the result when $z_{4,1}$ is unabortable, i.e. the PCP is used. In this case, $L_2$ is negative and $\tau_2$ may miss its deadline. The right columns of Table 4 show the result when $P^A_{4,1}$ is $P_4$, i.e. the priority abort scheme is used. In this case, $z_{4,1}$ is aborted not only by $\tau_2$, but also by $\tau_3$. As a result, an upper bound of $m_{4,1}$ cannot be determined using Theorem 8 and the schedulability of the task set cannot be shown.

## 6 Extensions of the CAP

The CAP can be extended in the following ways to further reduce the number of abortions.

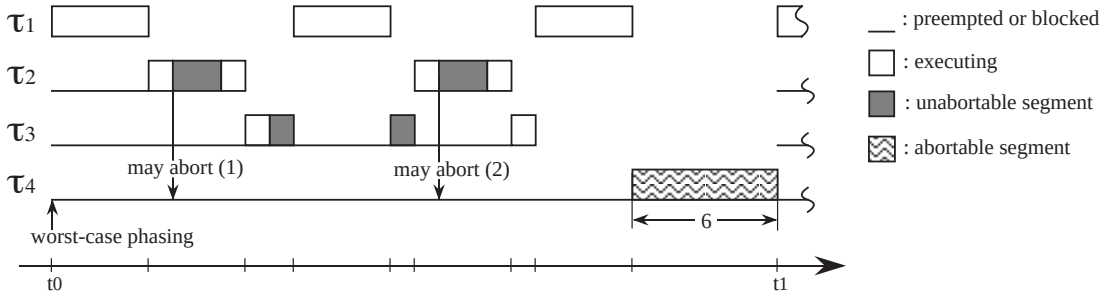


Figure 2: Evaluation of an Upper Bound of $m_{4,1}$

| | $C_i$ | $T_i$ | $P_i$ | $A_{i,1}$ | $U_{i,1}$ | $B_i$ | $C_i^+$ | $L_i$ | $A_{i,1}$ | $U_{i,1}$ | $B_i$ | $C_i^+$ | $L_i$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $\tau_1$ | 4 | 10 | $P_1$ | - | - | 0 | 0 | 6 | - | - | 0 | 0 | 6 |
| $\tau_2$ | 4 | 15 | $P_2$ | 0 | 2 [$P_2$] | 4 | 0 | -1 | 0 | 2 [$P_2$] | 2 | 0 | 1 |
| $\tau_3$ | 4 | 30 | $P_3$ | 0 | 2 [$P_2$] | 4 | 0 | 2 | 0 | 2 [$P_2$] | 2 | 0 | 4 |
| $\tau_4$ | 10 | 100 | $P_4$ | 0 | 4 [$P_2$] | 0 | 0 | 8 | 2 [$P_4$] | 2 [$P_2$] | 0 | - | - |

Table 4: Schedulability Analysis Examples (Case 2 and 3)

- A critical section can be executed at a higher priority level than the priority of the task. When the critical section is aborted, its priority is reset to that priority level instead of the task's priority.

  The execution of the critical section precedes some higher priority jobs and thus the chance of an abortion is decreased. The maximum blocking durations of preceded jobs are prolonged instead.

- The abortable segment of a critical section can be divided into sub-segments and a priority ceiling can be assigned to each sub-segment, so that the priority ceiling of the job will not decrease as the execution proceeds.

  With this extension, the maximum processing time of the abortable segment, the right side of (1), can be reduced.

- Whether a job can abort a critical section or not can be determined by a set of tasks, called an *abort task set*, defined for the abortable segment of each critical section instead of its priority ceiling.

  With the ceiling abort scheme, if $\tau_3$ can abort $z_{4,j}$, for example, $\tau_2$ can also abort it. However, when $\tau_2$ has larger schedulable laxity than $\tau_3$, $\tau_2$ may not need to abort $z_{4,j}$. In this case, $\tau_2$ is excluded from the abort task set of $z_{4,j}$.

  We call this method the *selective abort scheme*. With this scheme, some unnecessary abortions are avoided and the number of abortions can be reduced.

## 7 Conclusion

In this paper, a real-time synchronization protocol called the ceiling abort protocol (CAP) is proposed and a sufficient condition for the schedulability under the protocol is presented. As an example of the CAP's applicability, we present a set of tasks which cannot be scheduled using conventional protocols, but can be scheduled using the CAP. We also discuss some extensions of the CAP.

In this paper, the rate monotonic scheduling is adopted as the base scheduling algorithm in which the deadline of each job is at the end of each period. When the deadline of one of the jobs is earlier due to a jitter requirement or for some other reason, making critical sections abortable is even more effective [3]. Another useful application of abortable critical sections is to incorporate them into the earliest deadline first scheduling algorithm, in which a higher priority task has a smaller schedulable laxity. Combining the ceiling abort scheme with the earliest deadline first scheduling remains as future work.

## References

[1] J. Huang, J. A. Stankovic, K. Ramamritham, and D. Towsley, "On using priority inheritance in real-time databases," in *Proc. Real-Time Systems Symposium*, pp. 210–221, Dec. 1991.

[2] H. Tokuda and T. Nakajima, "Evaluation of real-time synchronization in real-time Mach," in *Proc. USENIX Mach Symposium*, pp. 213–221, Nov. 1991.

[3] L. Shu and M. Young, "A mixed locking/abort protocol for hard real-time systems," in *Proc. Real-Time Operating Systems and Software*, pp. 102–106, May 1994.

[4] L. Sha, R. Rajkumar, and J. P. Lehoczky, "Priority inheritance protocols: An approach to real-time synchronization," *IEEE Trans. Computers*, vol. 39, pp. 1175–1185, Sept. 1990.

[5] T. P. Baker, "Stack-based resource allocation policy for realtime processes," in *Proc. Real-Time Systems Symposium*, pp. 191–200, Dec. 1990.

[6] J. Lehoczky, L. Sha, and Y. Ding, "The rate monotonic scheduling algorithm: Exact characterization and average case behavior," in *Proc. Real-Time Systems Symposium*, pp. 166–171, Dec. 1989.

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