

A Conditional Abortable Priority Ceiling Protocol for Scheduling Mixed Real-time Tasks

Kam-yiu Lam
Department of Computer Science
City University of Hong Kong
83 Tat Chee Avenue, Kowloon
Hong Kong
Fax: 852-2788-8614
Tel.: 852-788-9807
Email: cskylam@cityu.edu.hk

Joseph Kee-Yin Ng
Department of Computer Science
Hong Kong Baptist University
224 Waterloo Road, Kowloon
Hong Kong
Fax: 852-339-7892
Tel.: 852-339-7864
Email: jng@comp.hkbu.edu.hk

Abstract

Priority Ceiling Protocol (PCP) is a well-known resource access protocol for hard real-time systems. However, it has a problem of ceiling blocking which imposes a great hindrance to task scheduling in mixed real-time systems where tasks may have different criticality. In this paper, a new resource access protocol called the Conditional Abortable Priority Ceiling Protocol (CA-PCP) is proposed. It resolves the problem of ceiling blocking by incorporating a conditional abort scheme into the PCP. The new protocol inherits all the desirable properties of the PCP and the Ceiling Abort Protocol (CAP) which is yet another modification of the PCP. In the proposed protocol, a condition is defined to control the abort of a job so that the schedulability of the system will not be affected. Performance study has been done to compare the CA-PCP with the PCP. The results indicate that CA-PCP can significantly improve the performance of a system if the lengths of the abortable critical sections are well chosen.

Keywords: Priority Ceiling Protocol, Real-time Scheduling, Task Scheduling

1 Introduction

In a hard real-time system, all the tasks must be scheduled to meet their deadlines. Otherwise, it may result in disasters. In order to meet the deadlines of the hard real-time tasks, different scheduling algorithms have been proposed [3]. One of the well-known algorithms is the *Priority Ceiling Protocol* (PCP) [6]. It is based on the *Rate Monotonic Scheduling* (RMS) analysis [1, 2, 4, 9] which is an algorithm for scheduling periodic hard real-time tasks in an uniprocessor environment. In the RMS, priorities are assigned to periodic tasks based on the periods of the tasks. A task with a shorter period is

assigned a higher priority. Liu and Layland [4] have shown that for a task set with n periodic, independent tasks, each with its deadline equals to its respective period. When we schedule these n tasks with the RMS priority assignment and running on a uniprocessor, each task can always meets its deadline if the total processor utilization of these n periodic tasks is less than or equal to $n(2^{1/n} - 1)$ [4].

However, the RMS has a problem in handling data synchronization as it assumes that all tasks are independent [7]. This assumption is not appropriate for most existing real-time systems as it is commonly found that tasks may share the same critical sections and data structures. Due to the sharing of resources, priority inversion may occur. Priority inversion is a situation where a higher priority task has to wait for a lower priority task because the lower priority task enters the critical section before the higher priority task occurs. Priority inversion can seriously affect the schedulability of the system. The blocking time can be very long and unbounded [6, 9] as the lower priority task may be preempted from using the CPU by other higher priority tasks. One way to reduce the blocking time is to use the priority inheritance protocol [7]. Although the priority inheritance protocol can put a bound on the blocking time, this bound is still high. To resolve this problem, the PCP is proposed in which a priority ceiling is defined for each critical section. By comparing the priority ceiling and the priorities of the tasks, it has been shown that the blocking time of a higher priority task in case of priority inversion can be bounded by the PCP and improve the predictability of the system.

Although the PCP can solve the priority inversion problem, it is still not suitable for those real-time systems where tasks have different criticality [10, 11, 12]. In such systems, some of the tasks are hard real-time tasks while the others are soft real-time tasks. All hard real-time tasks must be completed before their deadlines. Thus, a task with a higher criticality should be assigned a higher priority. On the other hand, although the impact of missing the deadline of a soft real-time task is not catastrophic, it is still undesirable. Another problem of the PCP is the problem of ceiling blocking [6]. This is the situation in which a medium priority task is blocked by a low priority even though they do not have any shared critical section. This is possible in the PCP when a lower priority task is sharing a critical section with a higher priority task. In this case, the lower priority task will inherit the priority of the higher priority task thus, block the medium priority task.

With an attempt to improve the performance of the previously mentioned medium priority tasks, Takada [12] proposed the *Ceiling Abort Protocol (CAP)* to make the tasks in the PCP abortable. In the CAP, a higher priority task is allowed to abort a lower priority task if the lower priority task is in an abortable critical section. However, the CAP has a serious problem of uncontrollable abort. It is possible that a large number of aborts may occur at the same time. This can seriously reduce the system

performance. When the utilization is already high, aborting a task further increases the utilization and can seriously reduce the schedulability of the resource access protocol.

To solve the above problems, we propose a new protocol called the *Conditional Abortable Priority Ceiling Protocol (CA-PCP)*. In the CA-PCP, a condition is defined to control the number of aborts with the attempt to reduce impact of aborting a task on the schedulability of the system and at the same time to reduce the number of ceiling blocking. The remaining parts of the paper are organized as follows. Section 2 reviews the hard real-time resource access protocols: the PCP and the CAP. Section 3 presents the proposed CA-PCP and the schedulability analysis. Section 4 describes the simulation model and the corresponding parameters for our simulation experiments. Section 5 summarizes the major performance results of the CA-PCP. Finally, the conclusions of the paper are presented in Section 6.

2 Real-time Resource Access Protocols

2.1 Priority Ceiling Protocol (PCP)

The *Priority Ceiling Protocol (PCP)* integrates scheduling algorithm with data synchronization for the scheduling of periodic hard real-time tasks in accessing resources. With the PCP, the blocking due to priority inversion is limited to, at most, the duration of a single critical section [6]. Other desirable properties of the PCP are that it prevents deadlocks and transitive blocking [1, 6]. To describe the PCP, we have to introduce the notion of a job within a task. To distinguish a task from a job, we have the following definition. In a system, each thread of execution is a task, and a job is the periodic request or the piece of code to be executed within a periodic task. Hence, the definition of the PCP can be summarized as follows:

1. Each job is scheduled according to the RMS.
2. Critical sections are guarded with semaphores.
3. For a semaphore, S_j , the priority ceiling, $c(S_j)$ is defined as the highest priority of the task which may access S_j .
4. Let S^* be the semaphore with the highest priority ceiling of all semaphores currently locked by all other jobs. Before a job J_i enters a critical section, CS_{ij} , it must first lock the semaphores S_j guarding CS_{ij} . J_i will be blocked and cannot lock S_j if the priority of J_i , $P(J_i)$, is not higher than the priority ceiling of semaphore S^* , $c(S^*)$. In this case, J_i is said to be blocked on S^* by the job J^* which is holding the lock on S^* . Otherwise, J_i will obtain the lock on S_j and enter the critical section, CS_{ij} .

5. When J_i exits the critical section, the semaphore associated with the critical section will be unlocked and the highest priority job, if any, blocked by J_i will be awakened to use the critical section after successfully locking the semaphore.
6. If J_i blocks some other higher priority jobs, J_i inherits P_H , the highest priority of the job blocked by J_i . When J_i exits from a critical section, it resumes its original priority.
7. J_i will preempt another job from using the processor if the priority of J_i , inherited priority or its original priority, is higher and J_i does not need to enter a critical section.

2.2 Ceiling Abort Protocol (CAP)

In the CAP, each critical section, CS_{ij} (the critical section of task τ_i guarded by semaphore S_j) is divided into two sections: the *abortable critical section* (CS_{ij}^A) and the *unabortable critical section* (CS_{ij}^U) as shown in figure 1. The Priority ceiling of CS_{ij}^U and CS_{ij}^A are defined as $c(CS_{ij}^U)$ and $c(CS_{ij}^A)$ respectively. $c(CS_{ij}^U)$ is defined as the highest priority of the task that may acquire a lock on the semaphore guarding CS_{ij} . Thus, $c(CS_{ij}^U)$ is equal to $c(S_j)$. The value of $c(CS_{ij}^A)$ should be determined to be an arbitrary priority which is lower than $c(CS_{ij}^U)$ and higher than $P(J_i)$, the priority of task τ_i which will access the critical section CS_{ij} . If a job, J_i , is in the abortable critical section CS_{ij}^A and another job J_k also wants to enter the same critical section, J_k may abort J_i and enter the critical section if $P(J_k)$ is higher than $c(CS_{ij}^A)$. On the other hand, if J_i is in CS_{ij}^U , J_k cannot abort it no matter how high its priority is.

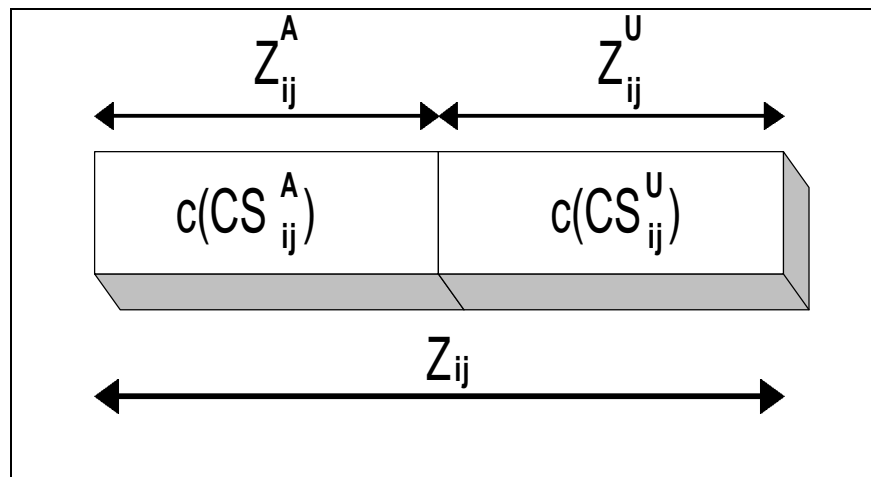


Figure 1: Critical Sections in the CAP

where Z_{ij}^A : the length of abortable critical section, CS_{ij}^A
 Z_{ij} : the length of critical section of task τ_i and guarded by semaphore S_j

Z_{ij}^U : the length of non-abortable critical section, CS_{ij}^U

$$Z_{ij} = Z_{ij}^A + Z_{ij}^U$$

Note: Z_{ij}^A and Z_{ij}^U are defined in terms of time.

The major problem of the CAP is uncontrolled abort that can degrade the system performance. The following is an example to show this uncontrolled abort.

Example. Figure 2 shows the problem of uncontrollable abort in the CAP. Consider there are four jobs, J_1 to J_4 , from four tasks, t_1 to t_4 , respectively, with priorities $P(t_1) > P(t_2) > P(t_3) > P(t_4)$. t_4 has an abortable critical section while t_1 , t_2 and t_3 have unabortable critical sections. J_4 enters its abortable critical section at time t_1 and it is preempted at time t_2 . It is aborted by J_3 at time t_3 . When J_3 finishes, J_4 restarts and enters its abortable critical section. However, it is again being aborted repeatedly when other higher priority jobs J_1, J_2, J_3 get into the system and enters their critical sections. Finally, J_4 misses its deadline at time t_{22} .

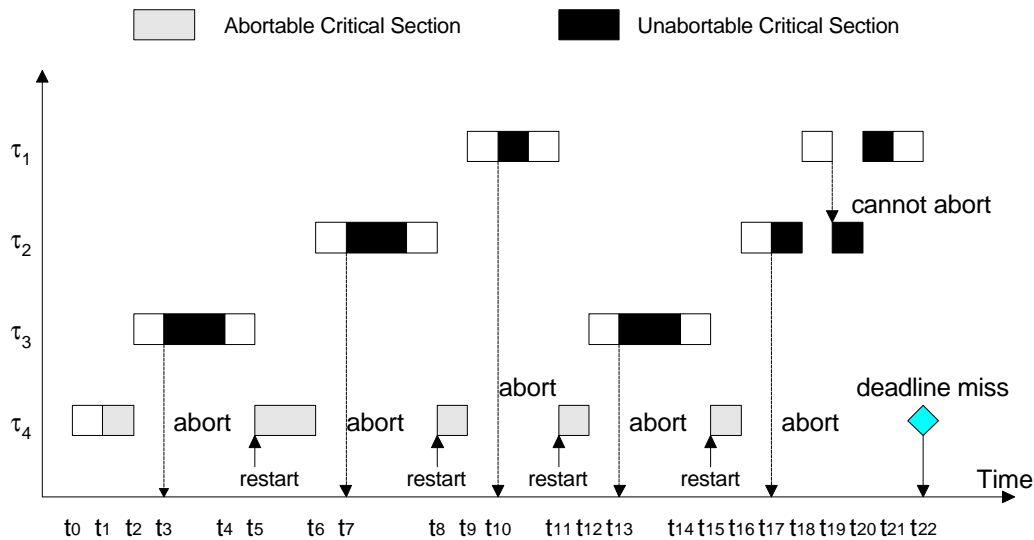


Figure 2: Uncontrollable Abort in the CAP

The above example indicates that aborting a job not only will make it miss its deadline but also affect the performance of other tasks as the extra work for restarting the job increases the system workload and utilization. The abort of J_4 may also affect the performance of other lower priority jobs as well.

3 Conditional Abortable PCP (CA-PCP)

3.1 Definitions

In this section, we propose a new protocol called the *Conditional Abortable Priority Ceiling Protocol (CA-PCP)* which is based on the PCP and the CAP. The CA-PCP is designed for real-time systems with mixed tasks. In order to resolve the problem of ceiling blocking, jobs are abortable if they are in the abortable critical section. However, such kind of abort is conditional to ensure that the schedulability of the system will not be affected.

Similar to the CAP, a critical section, CS_{ij} , is divided into two parts: abortable critical section, CS_{ij}^A and non-abortable critical section, CS_{ij}^U . In the CA-PCP, the condition to abort a lower priority job J_i in the abortable critical section CS_{ij}^A depends on the utilization, the length of the abortable critical section, and the amount of resources already held by the job which is going to be aborted if necessary. These factors are important as they can affect the schedulability of the system. Aborting a job increases the system workload as previous computation is thrown away. Furthermore, the aborted job has to be restarted from the beginning and it may not have sufficient slack time for its re-execution. Consequently, it will have a higher probability of missing its deadline. This is especially true when the length of the abortable critical section is long.

Similar to the PCP, in the CA-PCP, each job is assigned a priority when it is initiated. The job with the highest priority among all the jobs ready to run is assigned to use the processor. If a job, J_i , in its critical section blocks a higher priority job J_H due to a conflict in the critical section, J_i will inherit the priority $p(J_H)$ from job J_H . When J_i leaves the critical section, it resumes its original priority.

Before a job J_k enters a critical section, $CS_{k,j}$, it must first obtain the semaphore S_j guarding the critical section, $CS_{k,j}$. However, if there is another job, J_i , which is holding the same semaphore, S_j , and is in the critical section, $CS_{i,j}$, there are three possible cases of conflict in resolving the conflict:

Case 1: J_i is in abortable critical section and $P(J_k) > c(CS_{ij}^A)$

Case 2: J_i is in abortable critical section and $c(CS_{ij}^A) \geq P(J_k)$

Case 3: J_i is in non-abortable critical section

Case 1

If the priority of J_k is higher than the priority ceiling of the abortable critical section, $c(\text{CS}_{ij}^A)$ and if the condition for abort, CA, is true, J_k aborts J_i and then enters the critical section CS_{kj} and J_i will be re-executed.

The definition of CA will be discussed in the next section.

Case 2

If the priority of J_k is not higher than $c(\text{CS}_{ij}^A)$, the locking on S_j will be denied and J_k will be blocked outside the critical section. Also, the job J_i , which is holding semaphore S_j , will inherit the priority $P(J_k)$ from J_k if its priority $P(J_i)$ is lower than $P(J_k)$. Later on, it will resume its original priority $P(J_i)$ when the blocking no longer exists.

Case 3

Since J_i is in the non-abortable critical section, J_k is blocked. The priority of J_i will be raised to $P(J_k)$ if its priority $P(J_i)$ is lower than $P(J_k)$. Later on, it will resume its original priority $P(J_i)$ when the blocking no longer exists.

When a job J_k leaves a critical section, the priority ceiling of the semaphore guarding the critical section will fall back to the former priority ceiling and the semaphore guarding the critical section, will be unlocked.

3.2 Schedulability Analysis of the CA-PCP

As defined in the previous sub-section, the CA-PCP does not allow a lower priority job to be aborted if the condition for abort, CA, is false. The purpose is to prevent the schedulability of the system being affected by the abort. Three factors have been considered for the condition of abort:

- a) the existing system utilization;
- b) the length of the abortable critical section; and
- c) the amount of resources, being held by the job under consideration.

From here, we would like to focus on the schedulability of the CA-PCP and show how to include the above three factors into the schedulability test of the system for aborting a job.

From Liu and Layland [4], a set of n periodic tasks, $\tau = (\mathbf{t}_1, \mathbf{t}_2, \dots, \mathbf{t}_n)$, ascending ordered by their periods, and having their deadlines set at the end of their periods, when scheduled according to the rate monotonic scheduling (RMS) algorithm, can always meet their deadlines if:

$$\frac{C_1}{T_1} + \frac{C_2}{T_2} + \dots + \frac{C_n}{T_n} \leq n(2^{\frac{1}{n}} - 1) \quad .(1)$$

where C_i and T_i are the execution time and period of task τ_i , respectively.

If we consider different phasing among this task set \mathbf{t} , we have the following:

A set of n period tasks, $\tau = (\mathbf{t}_1, \mathbf{t}_2, \dots, \mathbf{t}_n)$, can be scheduled according to the rate monotonic scheduling algorithm for all task phasing if and only if:

$$\forall_i, 1 \leq i \leq n, \min_{(k,l) \in R_i} \sum_{j=1}^i \frac{C_j}{T_j} \left\lceil \frac{lT_k}{T_j} \right\rceil = \min_{(k,l) \in R_i} \sum_{j=1}^i U_j \frac{T_j}{T_k} \left\lceil \frac{lT_k}{T_j} \right\rceil \leq 1 \quad .(2)$$

where C_j , T_j , and U_j are the execution time, period, and utilization of task \mathbf{t}_j , respectively and $R_i = \{(k,l) \mid 1 \leq k \leq i, l = 1, \dots, \lfloor T_i/T_k \rfloor\}$. [1]

The physical meaning of R_i is clear. Since $T_1 \leq T_2 \leq \dots \leq T_n$, (k, l) is a pair such that a task \mathbf{t}_i with period T_i can be affected by the higher priority tasks \mathbf{t}_k from 1 up to $\lfloor T_i/T_k \rfloor$ number of times according to the value of l .

However, results from the RMS analysis are obtained under the assumption that all tasks are independent. That is, there is no blocking due to the sharing of critical sections. To account for the blocking among a task set, we need to consider the preemption caused by higher priority tasks, its own utilization, and most important of all, the maximum blocking time. We denote the maximum blocking time of task \mathbf{t}_i as B_i . That is, the blocking time of task \mathbf{t}_i is bounded by B_i . With the sharing of critical sections, we have to check the following conditions for a schedule's feasibility:

A set of n period tasks, $\tau = (\mathbf{t}_1, \mathbf{t}_2, \dots, \mathbf{t}_n)$, which share resources, can be scheduled according to the rate monotonic scheduling algorithm if each of the following conditions is satisfied:

$$\forall_i, 1 \leq i \leq n, \frac{C_1}{T_1} + \frac{C_2}{T_2} + \Lambda + \frac{C_i}{T_i} + \frac{B_i}{T_i} \leq i(2^{\frac{1}{i}} - 1) \quad .(3)$$

The above schedulability test can also be generalized to the following:

A set of n period tasks, $\tau = (\mathbf{t}_1, \mathbf{t}_2, \dots, \mathbf{t}_n)$, which share resources, can be scheduled according to the rate monotonic scheduling algorithm if the following condition is satisfied:

$$\frac{C_1}{T_1} + \frac{C_2}{T_2} + \Lambda + \frac{C_n}{T_n} + \max\left(\frac{B_1}{T_1}, \frac{B_2}{T_2}, \Lambda, \frac{B_{n-1}}{T_{n-1}}\right) \leq n(2^{\frac{1}{n}} - 1) \quad .(4)$$

Since τ_n is the lowest priority task and experience no blocking, therefore $B_n = 0$. The above claim is true because $n(2^{\frac{1}{n}} - 1) \leq i(2^{\frac{1}{i}} - 1)$ and $\max\left(\frac{B_1}{T_1}, \frac{B_2}{T_2}, \Lambda, \frac{B_{n-1}}{T_{n-1}}\right) \geq \frac{B_i}{T_i}$ for all $i, 1 \leq i \leq n$.

To account for the different phasings among the tasks, we have:

A set of n period tasks, $\tau = (\mathbf{t}_1, \mathbf{t}_2, \dots, \mathbf{t}_n)$, which share resources, can be scheduled according to the rate monotonic scheduling algorithm for all task phasing if:

$$\forall_i, 1 \leq i \leq n, \min_{(k,l) \in R_i} \left\{ \sum_{j=1}^{i-1} U_j \frac{T_j}{lT_k} \left\lceil \frac{lT_k}{T_j} \right\rceil + \frac{C_i}{lT_k} + \frac{B_i}{lT_k} \right\} \leq 1. \quad (5)$$

Similar to the PCP, the maximum blocking time experienced by a task under the CA-PCP is limited to the length of one critical section. Thus, the maximum blocking time, B_i , of task \mathbf{t}_i can be expressed as:

$$B_i = \max\{Z_{aj} \mid a > i\} \quad .(6)$$

where Z_{aj} is the length of the critical section of task τ_a and guarded by semaphore S_j .

We identify the task τ_a such that τ_a is the task that shares the critical section guarded by semaphore S_j with task τ_i and gives task τ_i a maximum blocking time of Z_{aj} and denotes this blocking time for task τ_i as Z_i^* .

We can abort a job and restart it if it is inside the abortable critical section. Let m_i be the number of aborts this job has been experienced so far. Thus, we have a new blocking time, $B_i' = Z_i^* + m_i Z_i^{*A}$ where Z_i^* is as defined in the previous paragraph, and Z_i^{*A} is the abortable portion of Z_i^* . By replacing B_i with B_i' , we have the following:

A set of n period tasks, $\tau = (\mathbf{t}_1, \mathbf{t}_2, \dots, \mathbf{t}_n)$, which share resources, can be scheduled according to the CA-PCP algorithm if each of the following conditions is satisfied:

$$\forall_i, 1 \leq i \leq n, \frac{C_1}{T_1} + \frac{C_2}{T_2} + \Lambda + \frac{C_i}{T_i} + \frac{Z_i^* + m_i Z_i^{*A}}{T_i} \leq i(2^{\frac{1}{i}} - 1) \quad .(7)$$

This schedulability test can also be generalized to the following:

A set of n period tasks, $\tau = (\mathbf{t}_1, \mathbf{t}_2, \dots, \mathbf{t}_n)$, which share resources, can be scheduled according to the CA-PCP algorithm if the following condition is satisfied:

$$\frac{C_1}{T_1} + \frac{C_2}{T_2} + \Lambda + \frac{C_n}{T_n} + \max\left(\frac{Z_1^* + m_1(Z_1^{*A} + C_1^A)}{T_1}, \frac{Z_2^* + m_2(Z_2^{*A} + C_2^A)}{T_2}, \Lambda, \frac{Z_{n-1}^* + m_{n-1}(Z_{n-1}^{*A} + C_{n-1}^A)}{T_{n-1}}\right) \leq n(2^{\frac{1}{n}} - 1) \quad .(8)$$

where C_i^A is the computation time spent by task \mathbf{t}_i before each abort.

Finally, if we also consider different phasings among the task set, $\tau = (\mathbf{t}_1, \mathbf{t}_2, \dots, \mathbf{t}_n)$, we have the following findings:

A set of n period tasks, $\tau = (\mathbf{t}_1, \mathbf{t}_2, \dots, \mathbf{t}_n)$, which share resources, can be scheduled according to the CA-PCP algorithm for all task phasing if and only if:

$$\forall_i, 1 \leq i \leq n, \min_{(k,l) \in R_i} \left\{ \sum_{j=1}^{i-1} U_j \frac{T_j}{lT_k} \left\lceil \frac{lT_k}{T_j} \right\rceil + \frac{C_i}{lT_k} + \frac{Z_i^* + m_i(Z_i^{*A} + C_i^A)}{lT_k} \right\} \leq 1 \quad .(9)$$

where $C_i, C_i^A, T_j, U_j, Z_i^*, Z_i^{*A}, m_i$, and R_i are as defined in our previous discussion.

After we have defined the schedulability of the system, we can define the condition for abort, CA, based on equations (8) and (9). Whenever there is a conflict in using a critical section, equation (8) and equation (9) will be checked. If both of them are true and the task is in the abortable critical section, the task will be aborted. Note that this condition is dynamic. It is dependent on the defined length of the abortable section and the current system status such as utilization.

4 Simulation Model and Model Parameters

In order to investigate the performance of the CA-PCP, a system that models a real-time system with mixed tasks is designed and implemented. Based on this simulation model, experiments have been performed. The program is implemented with OpNet which is a proprietary simulation package [5]. Figure 3 depicts the system model. It consists of four components: a set of task generators, a task queue, a processor, a set of mechanisms for controlling the semaphores and priority ceiling.

Jobs are generated from the task generator periodically and are sent to the task queue in which they are queued up according to their priorities. The scheduler selects the first job in the task queue to use the processor for processing and the task scheduling is preemptive. That is when a new job arrives, its priority will be compared with the priority of the job which is using the processor. The higher priority job will be assigned to use the processor and the lower one will be put back in the task queue. A job will be blocked if its priority is not higher than the priority ceiling or its requesting semaphore is held by another job which is in a non-abortable critical section.

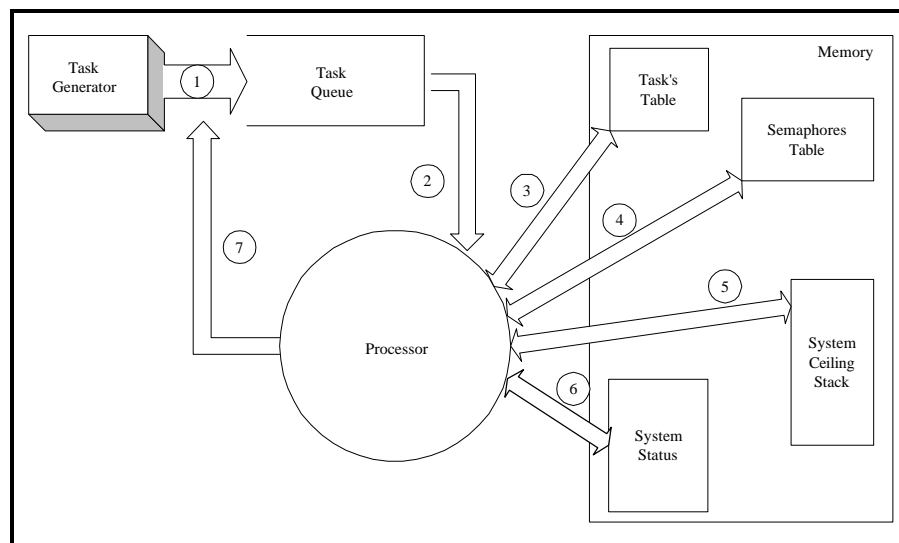


Figure 3: System Model

1. Job arrival to Task Queue.
2. Job at the Head of the Task Queue selected by CPU.
3. CPU retrieval information from Task Table.
4. CPU locks or unlocks semaphores on Semaphore Table.
5. Push or remove system ceiling onto System Ceiling Stack.
6. Retrieval or update system information from System Status Table.
7. Job releases from Processor and returns to Task Queue.

The system has 9 classes of tasks which are categorized into 3 groups according to their priorities – high, medium and low. Each class of tasks has different attributes as shown in Table 1.

Class of priority	High			Medium			Low		
Class of task	T ₁	T ₂	T ₃	T ₄	T ₅	T ₆	T ₇	T ₈	T ₉
# of Semaphores requests	1	1.2	1.2	2.3	2.4	2.4	5.6	5.7	5.8
Task length Distribution	5.6	7...9	10...12	15...18	19...22	23...26	30...34	35...44	45...54
Semaphores (S _i)	1, 3, 5, 7, 9			3, 4, 5, 6, 7			0, 1, 2, 3, 4, 5, 6, 7, 8, 9		
Deadline Period	Short			Medium			Long		

Table 1: Summary on the model parameters and their baseline values

In the determination of the model parameters and model configuration, we aim at investigating the performance characteristics of our proposed protocol, the CA-PCP, and to compare its performance with the PCP. Thus, a simple setup is more preferable in which the system consists of only a few semaphores so that it will have a high probability of semaphore conflict and a high chance of ceiling blocking. In the RMS, the deadlines of the tasks are assumed to be the arrival time of the next job from the same generator. In order to have more deadline missing in the experiments to facilitate the comparison¹, in the assignment of deadlines to the jobs, the deadlines are set at a fixed ratio (say 0.9) of the periods of the tasks. Hence, the higher priority jobs, having shorter periods, will have shorter deadlines and lower priority jobs will have longer deadlines.

The primary performance measure is the number of *deadline missing* for each class of tasks. We also measure the *mean task queue length* which is an indicator on the degree of resource contention in the system. In order to investigate the impact of the length of abortable critical section on its performance, we define the *abortable critical section length (ACSL)* as the length of the abortable critical section over the length of the critical section. The reason of using ACSL to test the performance of the CA-PCP is that its performance should be highly dependent on the length of the abortable critical section. To simplify the model without loss of generality, the same value of ACSL is assigned to all the semaphores in the system.

5 Simulation Results

The main difference between the PCP and the CA-PCP is that the CA-PCP is abortable. It is expected that if the value of ACSL is large, the probability of abort will be high. When ACSL is set at

¹ It should be noted that under normal situation hard real-time tasks must not miss their deadlines.

zero, the protocol becomes the PCP. Three important factors on the protocol performance are: (1) utilization and (2) probability of conflict in using the critical sections and (3) the value of ACSL. Two sets of experiments have been performed to study the impacts of these factors on the performance of the CA-PCP as compared with the PCP. One is under a high conflict probability environment and the other is under a medium conflict probability environment. The reason for not testing their performance under a low conflict probability environment is that if the conflict probability is low, the performance of the CA-PCP should be very close to the PCP. For simulating a high conflict probability environment, a single semaphore is used to simulate the case where many jobs will access the same critical section. For simulating a medium conflict probability environment, the system attributes are the same as defined in table 1.

5.1 High Conflict Probability

Figure 4 depicts the result of changing the value of ACSL on the performance for each class of task under an utilization of 70% with a high semaphore conflict probability. From the figure, it can be seen that in the CA-PCP, the probability of deadline missing for different classes of tasks is highly affected by the value of ACSL. When the value of ACSL is small (e.g., less than or equal to 0.4), the number of deadline missing of higher priority tasks (i.e., class 1 and class 2) is much larger than that of the lower priority tasks. It is because higher priority tasks have tighter deadline constraints and a smaller value of ACSL means more ceiling blockings, since the performance of the PCP is the case when the value of ACSL is equal to zero. Thus, it is not suitable for higher priority tasks because their deadlines are tight due to ceiling blocking.

With an increase in the value of ACSL, the number of deadline missing for the higher priority tasks decreases. On the contrary, the number of deadline missing for the lower priority tasks increases. This is because making a part of the critical section to be abortable can reduce the number of ceiling blocking. Therefore, the higher priority tasks can have higher probability to meet their deadlines. However, increasing the value of ACSL increases the number of aborts and aborting jobs in turn increase the utilization. Higher utilization gives less slack time for the re-execution of the aborted jobs and affects the execution of the lower priority tasks. Thus, the numbers of deadline missing for lower priority tasks will become higher.

Figure 5 shows the total number of deadline missing for all the tasks under different utilization. It can be seen that the use of abortable critical sections also benefits the system performance as a whole and not just the higher priority tasks. At any utilization, when we increase the value of ACSL, the total number of deadline missing decreases. When the value of ACSL is small, the aborted jobs can still meet

their deadlines as they have sufficient amount of slack time for the re-execution. When the value of ACSL is greater than a certain value (e.g., ACSL = 0.6), the number of deadline missing starts to increase especially under high utilization (>70%). When the whole critical section is abortable (e.g., ACSL = 1), the performance of the CA-PCP is similar to that of the PCP. The benefit of a smaller number of ceiling blocking is counter balanced by a larger number of aborts. This is especially true when the utilization is high (> 70%) as aborting a job has a greater impact on the schedulability of the system under high utilization.

From the above results, we can find that the performance of the CA-PCP is much better than the PCP especially when the value of ACSL is small under different utilization. Also, we can see that utilization and the value of ACSL play a very important role on the performance of the CA-PCP and it is highly beneficial to the system performance to control the number of aborts. The above result also has pointed out the problem of the CAP in which the condition for abort is not dependent on the utilization and is uncontrollable.

Figure 6 depicts the mean queue length of the system under different utilization. Increasing the value of ACSL can reduce the probability of conflicts because the probability of a lower priority task holding a semaphore required by a higher priority task is small. However, it also has the undesirable effect of a higher system workload due to the re-execution of the aborted jobs. So, the curves decrease for small values of ACSL. When the number of aborts is large, the effect of the re-execution of the aborted jobs becomes significant and the mean queue length increases gradually especially when the utilization is high.

5.2 Medium Conflict Probability

Figure 7 shows the total number of deadline missing of all the tasks under different utilization and with a medium semaphore conflict probability. With a medium conflict probability, the number of deadline missing is smaller than under a high semaphore conflict probability because the number of blocking and aborts are smaller. Consistent with the results under high semaphore conflict probability, the performance of the CA-PCP is much better than that of the PCP especially when the value of ACSL is small. The performance of the CA-PCP is highly dependent on the utilization of the system and the chosen value of ACSL. If the value of ACSL is set to zero, the number of deadline missing is high even when the utilization is not very high as in the case of the PCP. When we increase the value of ACSL starting from small values, the number of deadline missing decreases. Under this scenario, aborting the job is beneficial to the whole system performance. When the value of ACSL is greater than 0.6, the

number of deadline missing increases. This is because there are too many aborts and the system cannot handle them.

Figure 8 shows the mean queue length under different utilization. It shows that it is about the same for different values of ACSL except when the utilization is very high (i.e., $U = 90\%$). Although increasing the value of ACSL increases the number of abort, the system still has sufficient capacity to handle the additional workload as the existing utilization is not very high and the number of aborts is not very large under medium semaphore conflict probability.

6 Conclusions

The Priority Ceiling Protocol (PCP), which is based on the Rate Monotonic Scheduling algorithm (RMS), can solve the priority inversion problem in hard real-time systems. However, it introduces a new problem called ceiling blocking which is serious for real-time systems with mixed tasks. Although the Ceiling Abortable Protocol (CAP) can solve part of the ceiling-blocking problem, it has the problem of uncontrollable abort. In this paper, a new resource access protocol called the *Conditional Abortable Priority Ceiling Protocol (CA-PCP)* is proposed for the scheduling of mixed tasks in a real-time system. The main difference between the CA-PCP and the CAP is that a condition is defined in the CA-PCP based on the existing system utilization, length of critical sections and length of abortable critical sections to dynamically adjust the condition for abort.

The performance evaluation on the newly proposed CA-PCP has been done using a simulation. The results indicate that the CA-PCP can give a better performance as compared with the PCP especially when a well-chosen value of the length of abortable critical section is used. Also, the performance of the CA-PCP is highly dependent on the utilization. To summarize our simulation results, the CA-PCP can reduce the number of deadline missing effectively especially under medium data contention by introducing a conditional abortable scheme in the protocol. The protocol is very much in favor of the systems that have some tasks with very short deadlines. Moreover, the conditional abort can effectively avoid deadline missing in this kind of a task set.

Reference

- [1] M. H. Klein, T. Ralya, W. Pollak, R. Obenza, M. G. Harbour, *A Practitioner's Handbook for Real-Time Analysis: Guide to Rate Monotonic Analysis for Real-Time Systems*, Kluwer Academic Press, 1993.

- [2] J. Lehoczky, L. Sha, and Y. Ding, "The Rate Monotonic Scheduling Algorithm : Exact Characterization And Average Case Behavior", *Proceedings of the 10th Real Time System Symposium*, pp.166-171, 1989.
- [3] J. Lehoczky, "Fixed Priority Scheduling of Periodic Task Sets with Arbitrary Deadlines", *Proceedings of the 11th Real-Time Systems Symposium*, pp.201-209, 1990.
- [4] C. L. Liu and J. W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment", *Journal of the Association for Computing Machinery*, vol. 20, no. 1, pp.46-61, 1973.
- [5] *OPNET Modeling Manual, 2.0*, MIL 3, Inc.
- [6] R. Rajkumar, *Synchronization in Real-Time System: A Priority Inheritance Approach*, Kluwer Academic Press, 1991.
- [7] L. Sha, R. Rajkumar, and J. Lehoczky, "Priority Inheritance protocols: An Approach to real-time synchronization", *IEEE Transactions on Computers*, vol. 39, no. 9, pp. 1175-1185, 1990.
- [8] L. Sha and S. Sathaye, "A Systematic Approach to Designing Distributed Real-time Systems", *IEEE Computer*, vol. 26, no.9, pp.68-78, September 1993.
- [9] L. Sha, R. Rajkumar and S. Sathaye, "Generalized Rate-Monotonic Scheduling Theory: A Framework for Developing Real-Time Systems", *Proceedings of the IEEE*, Vol. 82, No. 1, January 1994.
- [10] L. Shu and M. Young, "A Mixed Locking/Abort Protocol for Hard Real-Time Systems", *Proceedings of the 11th IEEE Workshop on Real-Time Operating Systems and Software. RTOSS'94*, pp. 102-6, May 1994.
- [11] L. Shu, M. Young and Ragunathan Rajkumar, "An Abort Ceiling Protocol for Controlling Priority Inversion", *Proceedings of the 1st International Workshop on Real-time Computing Systems & Application, Seoul Korea*, pp. 48-52, Decmber 1994.
- [12] H. Takada and K. Sakamura, "Real-Time Synchronization Protocols with Abortable Critical Sections", *Proceedings of the 1st International Workshop on Real-time Computing Systems & Application, Seoul Korea*, pp. 48-52, December 1994.

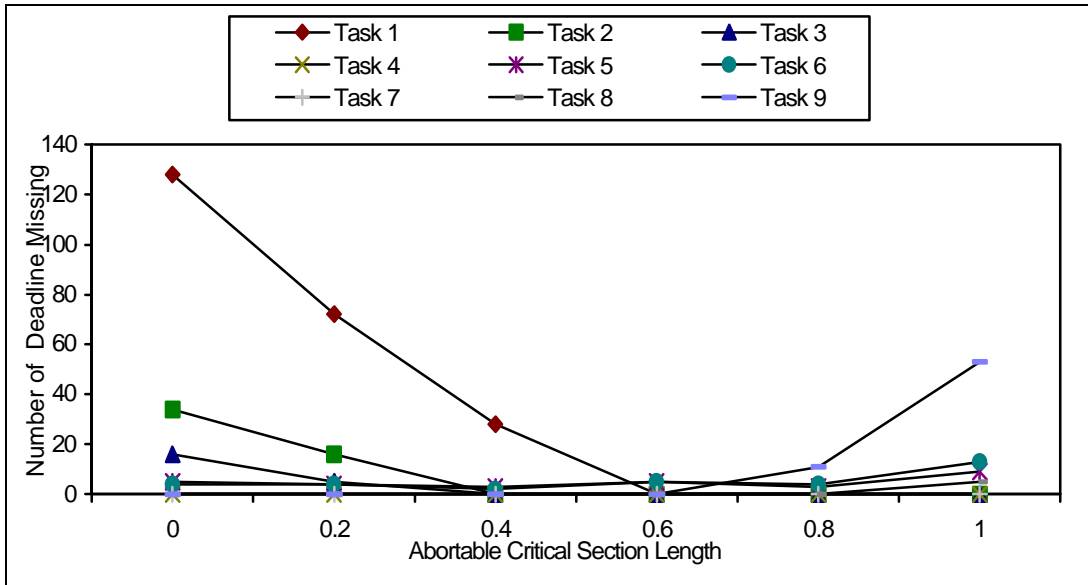


Figure 4: Impact of ACSL on Deadline Missing under 70% utilization and High Conflict Probability.

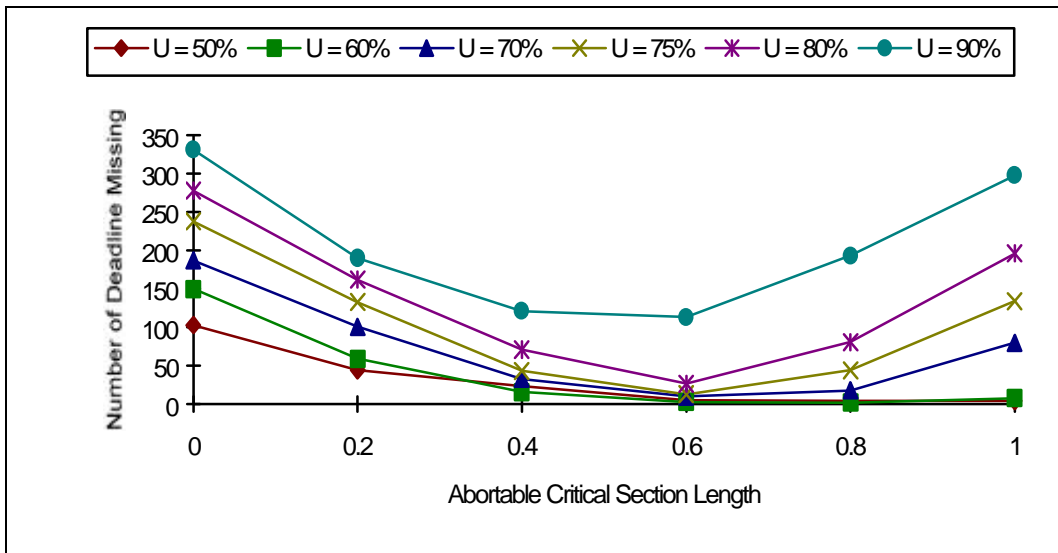


Figure 5: Impact of ACSL on Deadline Missing under High Conflict Probability.

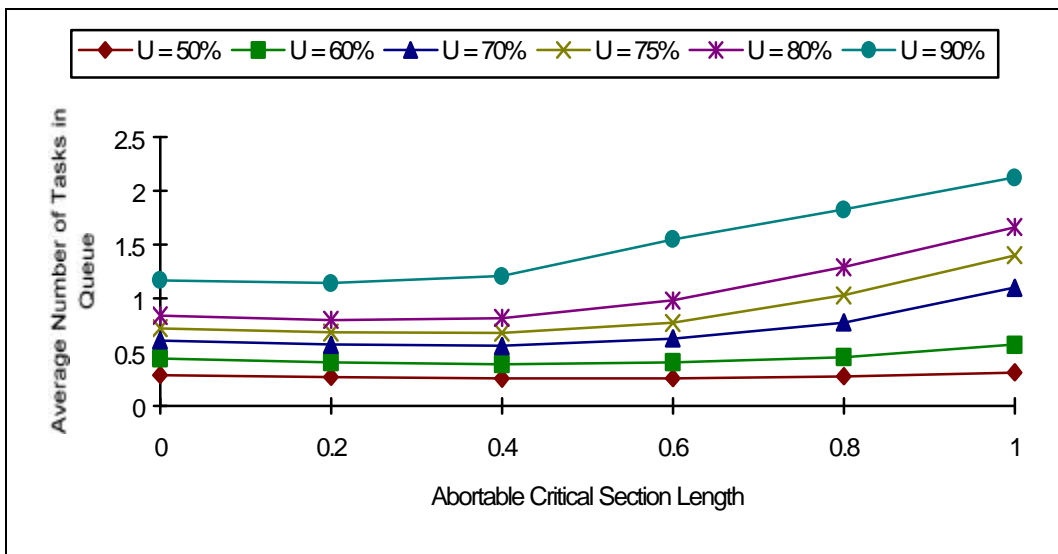


Figure 6: Impact of ACSL on Mean Queue length under High Conflict Probability.

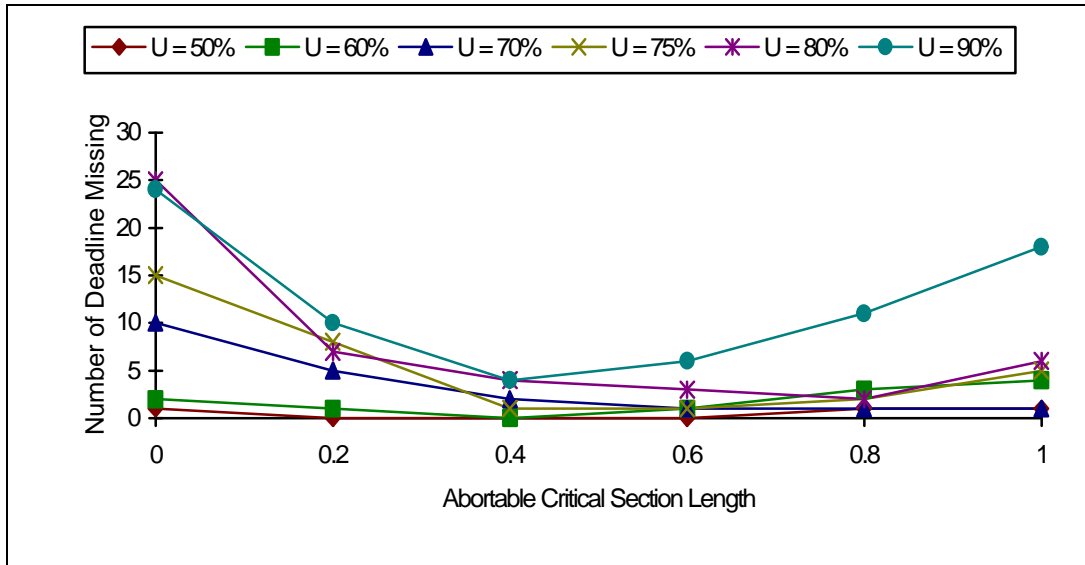


Figure 7: Impact of ACSL on Deadline Missing under Medium Conflict Probability.

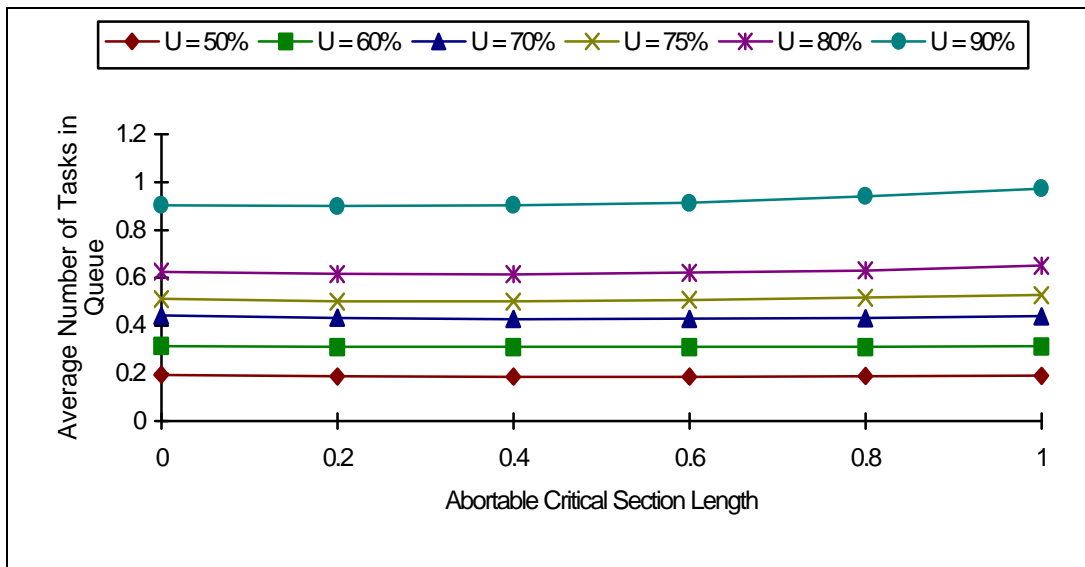


Figure 8: Impact of ACSL on Mean Queue length under Medium Conflict Probability.