Programming Assignment 2: Priority Inheritance and Priority Abort Protocols

Instructor: KD Kang

Objective: In this programming assignment, you are required to implement and compare the performance of the priority inheritance and priority abort protocols. Unlike the previous programming assignment, tasks can access critical sections that can result in priority inversions. For example, suppose a low priority task $T_L$ is currently in a critical section $CS_i$ and a high priority task $T_H$ with an earlier deadline tries to enter $CS_i$. In this case, $T_H$ suffers the priority inversion in which it is blocked due to the low priority task $T_L$ already in $CS_i$. The problem may become worse if $T_L$ is preempted by a medium priority task $T_M$ that has a higher priority than $T_L$ but does not need to enter $CS_i$. Note that there could be many medium priority tasks that can preempt $T_L$. As a result, $T_H$ may block indefinitely, possibly missing its deadline.

The priority inheritance protocol (PIP) allows $T_L$ to inherit the priority of $T_H$ while it is in $CS_i$ (and comes back to its original priority after getting out of $CS_i$). Thus, $T_L$ will not be preempted by $T_M$. In this way, PIP can reduce the blocking time of $T_H$.

Unfortunately, PIP may not fully address the priority inversion problem, because $T_L$ can be blocked if it intends to enter other critical sections $C_j$, $C_k$, etc. Consequently, $T_H$ can be blocked for a long time possibly missing its deadline. To address this problem, the priority abort protocol (PAP) takes an aggressive approach. If a critical section $CS_i$ is already taken by a low priority task $T_L$ when $T_H$ wants to enter $CS_i$, the system aborts $T_L$ and releases $CS_i$ so that $T_H$ can enter $CS_i$ immediately. In contrast, the aborted task $T_L$ should restart from the beginning. In this way, PAP aims to minimize the deadline miss ratio due to priority inversions.

By doing this project, students can learn the priority inversion problem and realize its adverse impact to the timeliness, while implementing and evaluating the two existing protocols developed to alleviate the priority inversion problem.

Assignment Due: Through Blackboard, submit your code and short design document including the instruction for compilation and execution of your code by 11:59PM, March 26, 2009. There will be 10% late penalty per day.

Simulation Spec. Basically, you can build upon the simulator developed for the first programming assignment. Compare the deadline miss ratio of PIP and PAP. (You don’t have to show anything else.) Key simulation parameters different from Programming Assignment 1 are specified in the following:

1. Scheduling: EDF.

2. Task Arrival Pattern: All tasks are periodic. There is no aperiodic task. In addition, assume that every periodic task arrives at time $t = 0$. You can simulate this by generating all periodic tasks (called sources in the pseudo code of the previous assignment) at $t = 0$. (Do not use the Poisson arrival patterns used in the Assignment 1.)
3. One simulation runs for 10 minutes as you did in the previous assignment. For each protocol, run the simulation five times and take the average to get the deadline miss ratio.

4. Do not apply admission control or imprecise computation.

5. A periodic task $T_i$ generates a job at every period $P_i$ where $P_i$ is uniformly selected in the range [200ms, 500ms], similar to Programming Assignment 1. Assign the execution time $C_i$ to $T_i$ where $C_i$ is uniformly selected in the range [5ms, 20ms].

6. Fix load = 1 for this assignment; therefore, you should stop generating periodic sources once the total utilization $U_t = \sum_{i=1}^{N} \frac{C_i}{P_i}$ becomes equal to or higher than 1 where $N$ is the total number of the generated periodic tasks. If $U_t \leq 1$ and there is no blocking or aborts/restarts due to critical sections, all deadlines will be met under EDF. However, you will observe deadline misses in this assignment due to task blocking, aborts, and restart.

7. Set the total number of the critical sections in the system = 100 in this assignment.

8. Each job created by $T_i$ will execute for $C_i$ time units, while accessing $N_i$ critical sections determined when the task is generated. Also, determine which critical section(s) $T_i$ will access when $T_i$ is generated in your simulation. Specifically, for $T_i$, select $N_i = 0.1 \times C_i$ critical sections out of the 100 critical sections in the system in a uniform random manner. For example, assume that there is a periodic task $T_i$ whose $C_i = 20ms$ and $P_i = 200ms$. Since $N_i = \lfloor 0.1 \times 20 \rfloor = 2$, $T_i$ accesses two critical sections. Also, suppose $T_i$ is determined to access critical sections $CS_3$ and $CS_5$ when it is generated. Thus, a $T_i$’s job has to access $CS_3$ and $CS_5$ and execute for 20ms at every period $P_i$, i.e., 200ms.

9. Mutual exclusion is required to access a critical section; that is, only one job can be in a critical section at one point of time. To implement mutual exclusion, a job needs to exclusively lock a critical section before it enters the critical section. If the critical section is already locked by another job, it has to block in PIP. In PAP, it can abort the job currently in the critical section if its priority is higher. Also, in PAP, an aborted job, if any, needs to restart from the beginning. A job releases the lock and leaves the critical section(s) when it finishes to execute for $C_i$ time units. In practice, you can support mutual exclusion in your simulation by maintaining a system-wide lock table where each entry indicates which job is currently locking the corresponding critical section, specifying the priority of a job locking the critical section.

10. To prevent a deadlock, require a job to lock all the critical section(s) that it needs to enter before it starts executing. If it cannot lock any of the critical sections that it has to access, it has to release all the locks, if any, it currently holds and block until the lock(s) on the critical section(s) are released. In this way, a deadlock can be prevented. Implement this deadlock prevention scheme for both PIP and PAP. In the previous example, a $T_i$’s job has to lock $CS_3$ and $CS_5$ before starting the execution. It releases the locks on $CS_3$ and $CS_5$ when it finishes executing for 20ms. Note that blocking time—the time for a job to block to get the lock(s)—is not counted as the execution time.

Overall, you are recommended to make the simulation as simple as possible. The key point is comparing the miss ratio between PIP and PAP. Good luck!