CSCI 460 Operating Systems
Assignment 3

This assignment is related to priority inversion. For this assignment, you could use any programming language like Java, Python, C, C++, etc. The purpose of this assignment is to understand priority inversion and how to prevent priority inversion using priority inheritance.

Background on priority inversion:

In any priority scheduling scheme, the system should always be executing the task with the highest priority. Priority inversion occurs when circumstances within the system force a higher-priority task to wait for a lower-priority task. A simple example of priority inversion occurs if a lower-priority task has locked a resource (such as a device or a binary semaphore) and a higher-priority task attempts to lock that same resource. The higher-priority task will be put in a blocked state until the resource is available. If the lower-priority task soon finishes with the resource and releases it, the higher-priority task may quickly resume, and it is possible that no real-time constraints are violated.

The first part of the assignment is to simulate a priority inversion scenario. The second part of the assignment involves using priority inheritance to prevent priority inversion. Basically, you will be given a set of jobs and their arrival time. You are required to simulate how they would execute using the following rules. In the first part, you can execute the jobs without priority inheritance; in the second part, you need to execute the same set of jobs with priority inheritance.

Suppose you have three types of jobs in the system: T_1, T_2, T_3. And the priority of each of these types of jobs are $P(T_1) = 3, P(T_2) = 2, P(T_3) = 1$. $P(T_i)$ indicates the priority of each type of job. Suppose jobs $T_1$and T_3 shares a common buffer B that can store 4 elements. Note that only one job should be able to access this buffer at a given time (mutual exclusion must be enforced to access this buffer). **Suppose the system uses a pure priority-based scheduling algorithm—meaning that whenever a new process with higher priority comes in, the current job is preempted and the new job with higher priority is executed.** Suppose the buffers initial value is <0,0,0,0>. When a job with T_1type executes and tries to get hold of the buffer—if it does, all the elements of the buffer will be replaced to 1, in other words, the buffer would contain <1,1,1,1>. If T_1 cannot acquire the buffer, it immediately goes to the blocking state. If a job of type 3 gets hold of the buffer, it would update all the elements of the buffer to <3,3,3,3>. Suppose that $T_1, or\ T_3$ job takes 3 time slices to run. (You can pick anything to be your time slice; it can be 1s, 0.5s, etc.). Whenever a job of type $T_1\ or\ T_3$ runs it prints the content of the buffer after it modifies their values. For example, job of type T_1 runs and gets control of the buffer, then it will modify the content of the buffer to <1,1,1,1> and prints $T_1 \cdot [1111] \cdot T_1$ in a new line. If a job of type T_3 runs and gets hold of the buffer, it modifies the buffer to <3,3,3,3> and prints $T_3 \cdot [3333] \cdot T_3$ in a new line.

On the other hand, a job of type T_2 takes 10 time slices to run. Whenever T_2 runs it prints T_2 and it will print out N for each of the time slice it executes when it finishes executing it will print T_2 again. For example, if T_2 runs for 6 time slices and then gets preempted by a job of type T_1then it prints outs: $T_2 \cdot NNNNNN$. Then once it is resumed after T_1 is executed, then rest of the values are printed: $NNNN \cdot T_2$. If T_2 runs without any interruptions, then it should print $T_2 \cdot NNNNNNNNNN \cdot T_2$. Unlike jobs $T_1\ and\ T_3$, the T_2 prints a N for each of the time slices it runs. Note that T_2 does not try to get hold of the buffer; therefore, it executes independently from T_1, T_2.

The input to this system is in the form of $< arrivalTime, j >$, where j indicates the type of the job $j = \{1,2,3\}$. When your simulation gets the following inputs, it should print out the following.

Input: $< 1,1 >, < 3,2 >, < 6,3 >$

Output:

Time 0:

Time 1: $T_1 \cdot [1111] \cdot T_1$

Time 4: $T_2 \cdot NNNNNNNNNN \cdot T_2$

Time 14: $T_3 \cdot [3333] \cdot T_3$

This is an example of a situation where no priority inversion.

Input: $< 0,2 >, < 3,2 >, < 6,3 >$

Output:

Time 0: $T_2 \cdot NNN$

Time 3: $T_1 \cdot [1111] \cdot T_1$

Time 6: $NNNNNNN \cdot T_2$

Time 13: $T_3 \cdot [3333] \cdot T_3$

This is an example of a situation where no priority inversion occurs, but job T_2gets preempted by the job T_1.

Input: $< 0,3 >, < 1,1 >< 2,2 >$

Output:

Time 0: $T_3 \cdot [3333] \cdot T_3$

Time 1: T_1 arrives and tries to access the buffer and goes to blocked state. T_3 resumes immediately.

Time 2: $T_2 \cdot NNNNNNNNN \cdot T_2$

Time 3:

Time 4:

.

.

.

.

Time 12:

Time 13: $T_1 \cdot [1111] \cdot T_1$

Note for the above input priority inversion occurs. T_3 starts executing at time 0 and locks the buffer. Then the T_1 arrives at time 1, and tries to acquire the buffer, since the buffer is already locked by T_3, T_1 immediately goes to blocked state, and T_3 resumes. (You can assume this happens instantaneously.)

Input: $< 0,3 >, < 1,3 >, < 2,2 >, < 3,1 >, < 8,3 >$

Output:

Time 0: $T_3 \cdot [3333] \cdot T_3$

Time 1: T_1 arrives and tries to access the buffer and goes to blocked state. T_3 resumes immediately.

Time 2: $T_2 \cdot NNNNNNNNN \cdot T_2$

Time 12:

Time 13: $T_1 \cdot [1111] \cdot T_1$

Time 16: $T_1 \cdot [1111] \cdot T_1$

Time 17: $T_3 \cdot [3333] \cdot T_3$

In this example you can see multiple Type 3 jobs coming to the system.

Then you need to implement the system using priority inheritance. The basic idea of the priority inheritance is that if higher priority task tries to access a resource that is locked by a lower priority task, then immediately higher priority task goes to blocking and priority of the lower priority task is incremented to the highest level possible. In this case priority is updated to priority level 4 (even higher than the priority of T_1). Then the lower priority task can quickly finish accessing the shared resource.

For example, once you implement the priority inheritance the following input should get the following output.

Input: $< 0,3 >, < 1,1 >< 2,2 >$

Output:

Time 0: $T_3 \cdot [3333] \cdot T_3$

Time 1: (T_1 comes to the system by immediately gets blocked and T_3 priority is incremented to highest priority level).

Time 2: T_2 comes to the system by does not execute as the current process executing is of higher priority.

Time 3: $T_1 \cdot [1111] \cdot T_1$

Time 6: $T_2 \cdot NNNNNNNNN \cdot T_2$

Note that at time 1 T_1tries to access the shared buffer, but since it is locked by the T_3, T_1 immediately goes to blocked state and T_3 priority is increased to level 3. Therefore, at time 2, when T_2 arrives to the system, it will not be able to preempt the T_3 as it has higher priority. Then at time 3, T_3 completes. There are two jobs available now: T_1, T_2. Since T_1has higher priority system will execute T_1 first and then T_2.

Following are 6 inputs to your simulation. Simulate them in both systems.

Input 1: $< 1,1 >, < 3,2 >, < 6,3 >$

Input 2: $< 0,2 >, < 3,2 >, < 6,3 >$

Input 3: $< 0,3 >, < 1,1 >< 2,2 >$

Input 4: $< 0,3 >, < 1,3 >, < 2,2 >, < 3,1 >, < 8,3 >$

Input 5: $< 0,3 >, < 1,1 >, < 2,2 >, < 3,2 >, < 8,3 >$

Input 6: $< 0,2 >, < 1,1 >, < 2,1 >, < 3,1 >, < 4,2 >, < 7,3 >$

Each line contains a schedule to run. Your goal is to simulate each of these schedules in a system without priority inheritance and a system with priority inheritance. **READ THE BOOK PAGES 486-489 to understand more about priority inversion and priority inheritance.**

Rubric: Points 20

This is the rough idea of the rubric; there may be changes to the rubric when the TA grades this assignment, but this rubric should cover the basic areas of the assignment.

Part 1: System without priority inheritance. (8 points)

1. Implementation of the simulation to run the three types of tasks. (2 points)
    a. Use of a data structure to keep track of each type of job and the time it has been executing.
2. Correctly preempts the job when a new job with a higher priority comes in. (correctly implements the priority-based scheduling) (2 point)
3. Correctly uses a mutex or a binary semaphore to simulate locking of the buffer. (1 point)
4. Prints out the correct values for each of the jobs when it executes. (2 point)
5. Prints out what happens at each time slice. (You can print nothing if nothing interesting happens) (1 point)

Part 2: System with priority inheritance. (9 points)

6. Implementation of the simulation to run the three types of tasks. (2 points)
    a. Use of a data structure to keep track of each type of job and the time it has been executing.
7. Correctly preempts the job when a new job with a higher priority comes in. (2 points)
8. Correctly uses a mutex or a binary semaphore to simulate locking of the buffer. (1 point)
9. Correctly updates the priority of the lower priority job when a higher priority job tries to lock a shared resource that is already locked by a lower priority job. (1 points)
10. Prints out the correct values for each of the jobs when it executes. (2 points)
11. Prints out what happens at each time slice. (You can print nothing if nothing interesting happens) (1 point)

12. Output is submitted in 2 output files (one output file for the system without priority inheritance and one for the system with priority inheritance). (2 points)
13. Readable code (1 point).