COMP 530 Introduction to Operating Systems

Fall 2017  
Kevin Jeffay

Worksheet 10, October 4

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Your Name: |  | You worked with: |  | +1/blank/-1: |  |
|  | Aaron Zhang |  | Victor Cui |  | +1 |  |
|  |  |  | Brandon Larue |  | +1 |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

1. In the lecture on Monitors I showed how priority inversions were possible when using a Hoare monitor given the semantics of *signal* and *wait* in Hoare monitors. The root cause of the problem is that condition synchronization in a Hoare monitor requires a signaler to “guarantee” that an awaited condition will be true when a waiting process resumes execution and this in turn effectively requires a context switch from a signaling process to a waiting process. Mesa monitors eliminate priority inversions by adopting a different semantic model wherein the “signal” (the *notify* operation) is a “hint” and not a guarantee that the awaited condition is true.

Explain exactly how (1) priority inversions were possible in the original formulation of a Hoare monitor, and (2) explain how the different semantic model of synchronization in Mesa monitors eliminates the priority inversions that can occur in Hoare monitors. Male reference to the code for the implementation of monitors as appropriate.

CANNOT OCCUR WITH TWO PROCESSES, important to understand.

Need a third process

There is a dispatch to the waiter in the code…

**(1)**

**High priority producer dispatchers a waiting, low-priority consumer and puts itself on the urgent queue. Now the low-priority consumer is running. A medium priority process comes in and preempts the low-priority consumer. The low-priority process is put on the ready queue. The high priority producer is still waiting on the preempted low-priority process. These higher priority processes are stuck waiting for the lock, and are effectively running at the medium priority.**

**Low priority consumer might have to wait for some medium priority jobs. If a high priority consumer comes in right after the low priority job, it will have to wait for the medium priority jobs to finish, so that the low priority consumer can execute and release the lock.**

**(2)**

**By not having dispatch, a high priority job doesn’t force a low priority job to execute. A low priority consumer may have to re-wait if a high priority consumer executes first.**

2. It turns out that it is possible to eliminate priority inversions in Hoare monitors by integrating the condition synchronization and mutual exclusion mechanisms in monitors with the operating system scheduler. Describe (in prose and code if you can) how a priority scheduler could be adapted to dynamically adjust the priority of processes to ensure priority inversions do not occur when processes perform condition synchronization inside a Hoare monitor.

**Set the priority of the dispatched (previously waiting) process to the min { signaler priority, waiter priority }**

* **Priority inheritance. Low priority consumer inherits higher priority producer’s priority**

**(Remember, smaller priority value is higher priority)**

**When the inheriting (waiter/consumer) process exits the monitor, it’s priority is demoted back to the original priority of the waiter.**