Priority Inversion and Donation

Emil Ong

February 14, 2003

1 Priority Scheduling

Priority scheduling is used to schedule which process or thread next uses the CPU. Priorities (usually integers) are assigned to each thread and among threads that are ready to run, the one with the highest priority is chosen next. The idea behind priority scheduling is that high priority threads are doing the work we want done first, while threads with lower priority can wait. Usually, threads are kept in a queue, sorted by their priorities so that the highest priority thread is "next" on the queue.

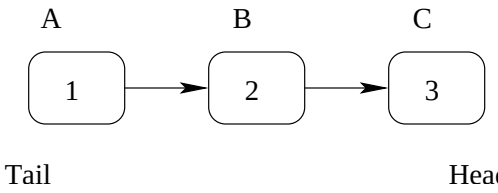

Figure 1: A simple example of a priority queue. There are 3 threads, A, B, and C with priorities 1, 2, and 3, respectively. Thread C will be chosen next because it has the highest priority.

2 Priority Inversion

Suppose we have a shared resource. In order to prevent race conditions and inconsistencies, it makes sense to use a lock (mutex). Locks make sure that only one thread is accessing the resource at a time. If a thread wants to access a resource it must acquire the lock first. If it is unable to acquire the lock (in other words, another thread is using the resource), it must wait until the thread currently accessing the resource releases the lock.

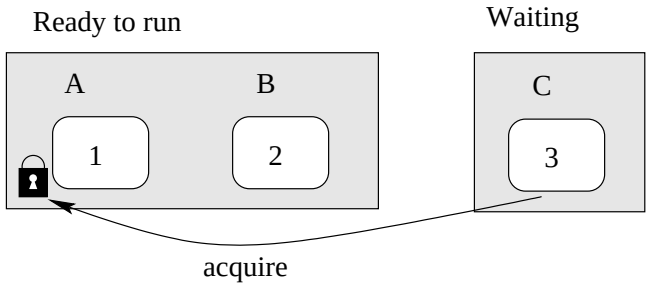

Figure 2: An example of priority inversion. Thread C, with priority 3, is waiting on thread A, with priority 1. Thread B, with priority 2, will run before thread C.

Look at Figure 2. Threads A, B, and C have priority 1, 2, and 3, respectively. Thread A is holding a lock on a resource that Thread C is wants to use. Thread C must wait for thread A to release the lock because it called acquire on the lock. Because thread C is waiting for the lock, it is not available to run. Thread A and B are ready to run. So when the priority scheduler chooses a thread to run next,

it can only choose between A and B. B has a higher priority, so it will go next. Thread A cannot run, so it won't be able to release the lock and thread C will have to wait until B finishes. In other words, the highest priority thread, thread C, is inadvertently being blocked from running by a lower priority thread, thread B.

3 Priority Donation

One solution to this problem is to donate priority. In the case we just looked at, if thread A had the same priority as thread C, it would not be blocked. So thread C "donates" its priority to thread A – thread A's priority effectively becomes 3.

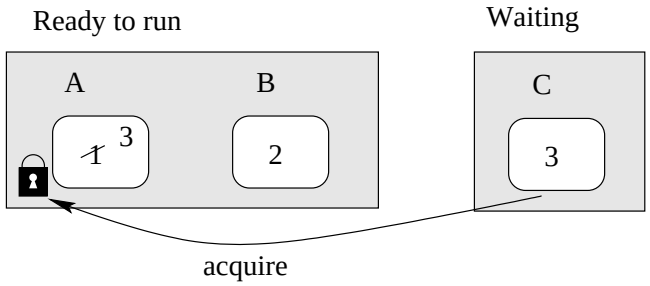

Figure 3: An example of priority donation. Thread C, with priority 3, donates its priority to thread A, with real priority 1. Thread A may now run.

When thread A releases the lock, its priority drops down again to its original priority. Thread C is now able to run and the priorities behave as we intended.

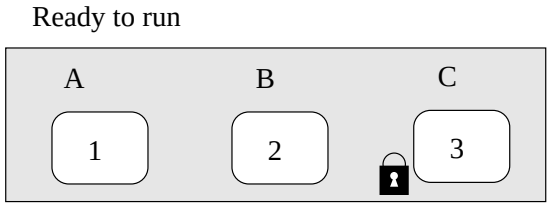

Figure 4: After priority donation. Thread C, with priority 3, is now able to run and thread A's priority dropped back down to 1.