Priority Scheduler

In this project module you are required to implement a Priority Scheduler. An issue with priority scheduling is a phenomenon called priority inversion: if a high priority thread needs to wait for a low priority thread (for example, for a resource held by the low priority thread), and a medium priority thread is on the ready list, the high priority thread needs to wait for medium priority thread to finish, because the low priority thread won't be scheduled until then. A way to solve this is to implement something called priority donation, where the high priority thread donates its priority to the low priority thread while the resource is held, so that it can be scheduled. Implementing this implies dealing with two issues: multiple donation (multiple threads waiting on the same resource), and nested donation (process A is waiting for B, and B is waiting for C, so A should donate its priority to both B and C).

For the project report, answer these questions:

1. When does priority donation happen?
    a. When a thread A needs access to a resource held by thread B, and A has higher priority than B.
    b. When a thread acquires a resource previously held by another thread, and there are other priority threads waiting on the same resource.
    c. All of the above.
2. When does a thread restore its original priority?
    a. Never.
    b. Not necessary.
    c. When it releases the resource previously held.

Task: Priority Scheduling

Implement Priority Scheduling by completing the PriorityScheduler class. Remember to use the nachos.conf file that uses the PriorityScheduler class. When implementing priority scheduling, you will find a point where you can easily compute the effective priority of thread, bu the computation takes a long time. To get full credit, you need to speed this up by caching the effective priority and only recalculating it when it may be possible for it to change.

As before, do not modify any classes to solve this task. The solution should involve creating a subclass of ThreadQueue that will work with the existing Lock, Semaphore, and Condition classes.

You should implement priority donation first, then add donation, and finally restoration.

Submission

Submit a zip file with your solution. Your submission should include source code and a written report with a detailed discussion of your design and implementation choices. The final grade will be 40% for your written report, and 60% for the correctness of your implementation.

This project is worth 50 points.