Project 3: Priority Scheduler

Readings: Chapter 4.2, Nachos Tutorial
Due date: midnight, March 8th, 2015
TA: Yuanhao Yu (yhyu.mail AT gmail)

February 23, 2015

1 Getting ready

In the project, you are asked to implement a priority scheduler with priority donation. Recall that an issue with priority scheduling is priority inversion. If a high priority thread needs to wait for a low priority thread (for instance, for a lock held by a low priority thread), and another medium priority thread is on the ready list, then the high priority thread will not get the CPU till the medium priority task finishes because the low priority thread will not get any CPU time before so. A partial fix for this problem is to have the waiting thread donate its priority to the low priority thread while it is holding the lock. To implement priority donation, one needs to handle both multiple donations, i.e., multiple threads waiting on the same resource; and nested donations, where H is waiting on a lock that M holds and M is waiting on a lock that L holds, then H should donate its priority to both M and L. Answer the following questions:

1) When does priority donation happen? (Select one from the following choices) a) when a thread A needs to access the resource currently held by thread B, and thread A has a higher priority than B; b) when a thread acquires a resource that was previously held by another thread and there are other higher priority threads waiting for the same resource; c) all above

2) When does a thread restore its original priority? a) never; b) not necessary; c) when it releases the resource it previously held.

Justify your answer.

2 Task

Implement priority scheduling in Nachos by completing the `PriorityScheduler` class. Priority scheduling is a key building block in real-time systems. Note that in order to use your priority scheduler, you will need to change a line in `nachos.conf`

that specifies the scheduler class to use. The `ThreadedKernel.scheduler` key is initially equal to `nachos.threads.RoundRobinScheduler`. You need to change this to `nachos.threads.PriorityScheduler` when you're ready to run Nachos with priority scheduling.

Note that while solving the priority donation problem, you will find a point where you can easily calculate the effective priority for a thread, but this calculation takes a long time. To receive full credit for the design aspect of this project, you need to speed this up by caching the effective priority and only recalculating a thread's effective priority when it is possible for it to change.

It is important that you do not break the abstraction barriers while doing this part – the `Lock` class does not need to be modified. Priority donation should be accomplished by creating a subclass of `ThreadQueue` that will accomplish priority donation when used with the existing Lock class, and still work correctly when used with the existing `Semaphore` and `Condition` classes.

Hint: use the tester codes for unit testing. Implement the priority scheduler first without priority donation, next deal with priority donation followed by priority restoration.

3 Testing

Test your codes with the tester classes in *ag/* directory. You can modify the tester codes to include more test cases. Your codes will be tested automatically with a more comprehensive set of testers by the TAs. For testing, you can change the random seed by including "-s seed" in the command line (where seed is an integer).

4 Submission

You should **commit** your codes through the SVN repository. *Make sure the directory structure is intact.* Under the current nachos project directory, include a file called "report.pdf" (e.g, nachos/proj2/report.pdf for this project). In the report, you should include the following two parts:

1. Answers to the questions in the project description.
2. Key data structures used and a block diagram of your implementation (limited to 2 pages).

Grade of each project is divided as 20% for Q&A, 40% for key data structures/block diagram on the implementation and 40% for the correctness of your implemention with the respective percentage for each task.