EECS 211
Advanced System Software
Winter 2008

Assignment 4

Posted: February 3, 2008
Due: February 14, 2008

Topic: Priority scheduling in Nachos

Instructions:

The goal of this assignment is to develop, implement and test task scheduling in the Nachos system. This assignment continues the previous assignments based on the "Nachos Assignment 1" described in the file **`doc/thread.ps`** of the Nachos installation. Again, the instructions below assume that you read **`doc/thread.ps`** in parallel.

Task 1: Implement a priority-based scheduler

See item 8 in **`doc/thread.ps`**.
Again, we will work in the **`threads`** directory. As you have noticed in the previous assignments, the original Nachos scheduler implements a straight-forward first-come-first-served (FCFS) scheduling policy. We will change that now into a priority-based policy. That is, with each thread, we will associate a priority between 0 and 9, 0 being the highest priority (first choice).

Follow the instructions to item 8 in **`doc/thread.ps`** to implement the priority scheduler. You will need to modify the source code only in files **`thread.cc`**, **`thread.h`**, and **`scheduler.cc`**.

Hint: There is not much new code to write!

Deliverable 1: (10 points)

Submit the extended source files **`thread.cc`**, **`thread.h`**, and **`scheduler.cc`** as email attachments. In your email, briefly explain your implementation.

Task 2: Implement a bounded buffer for safe inter-thread communication

See item 2 in **`doc/threads.ps`**. For safe synchronization in the buffer, use locks and condition variables (from Assignment 2), not semaphores. Note that the bounded buffer described in chapter 6.6.1 in the textbook is implemented using semaphores, so that is *not* a solution to this assignment.

The bounded buffer should be implemented as a new class **`Buffer`** (for simplicity, we define the class and its methods at the beginning of the file **`threadtest.cc`**). The buffer size (maximum queue length) should be set at the time of instantiation (as a parameter to the constructor). The class **`Buffer`** should provide two public methods named **`Get`** and **`Put`** which take a single character (type **`char`**) out of the buffer, or place a character into the buffer, respectively (for details, see the provided template file **`threadtest.cc`**).

Make sure to properly synchronize the **`Get`** and **`Put`** methods (using locks and/or condition variables only!). The used locks and/or condition variables should be instantiated as members inside the **`Buffer`** class, and should be properly called by the necessary methods so that the user of the buffer does not need to worry about any synchronization.

Test your buffer implementation using a producer-consumer example. In the provided template file **`threadtest.cc`**, 2 producer and 2 consumer threads are instantiated which communicate via one shared instance of the bounded buffer. Note that the priority of the threads is provided as argument to the **`Fork()`** method call. The first producer and the first consumer get the (high) priority 1, whereas the second producer and the second consumer get the (low) priority 2.

Hint: Don't modify anything but the **`Buffer`** class and its implementation. There is no need to change any code for the consumer and producer threads.
Again, there is not much new code to write.

Deliverable 2: (20 points)

Submit the extended source file **`threadtest.cc`**. Also provide a brief description of your synchronization along with a script of the successfully running program. Briefly explain what happens and why.

Submission instructions:

To submit your homework, send an email with subject “EECS211 HW4” to the course instructor at doemer@uci.edu. Please put your text in the body of the email and supply the source files as attachments.

To ensure proper credit, be sure to send your email before the deadline: February 14, 2008, 11:59pm (before midnight).

--
Rainer Doemer (ET 444C, x4-9007, doemer@uci.edu)