EECS 211
Advanced System Software
Winter 2007

Assignment 3

Posted: January 25, 2007
Due: February 8, 2007

Topic: Priority scheduling in Nachos

Instructions:

The goal of this third assignment is to develop, implement and test task scheduling in the Nachos system. This assignment continues the previous assignment 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/threads.ps`** in parallel.

Task 1: Implement a priority-based scheduler

See item 8 in **`doc/threads.ps`**.
Again, we will work in the **`threads`** directory. As you have noticed in the previous assignment, the given Nachos scheduler implements a straight-forward FIFO 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/threads.ps`** to implement the priority scheduler. You will need to modify the code in the files **`thread.cc`**, **`thread.h`**, **`scheduler.cc`**, and **`scheduler.h`**.

Hints: There is not much new code to write. Note that there is a method **`List::SortedInsert()`** available that will be very handy.

Deliverable 1: (10 points)

Submit the extended source files **`thread.cc`**, **`thread.h`**, **`scheduler.cc`**, and **`scheduler.h`** (with proper comments!).

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

See item 2 in **`doc/threads.ps`**. You may also want to review the description of a bounded buffer in chapter 6.6.1 in the textbook.

The bounded buffer should be implemented as a new class **`BoundedBuffer`** (for simplicity, you may want to put the code into **`threadtest.cc`**). The buffer size (maximum queue length) should be set at the time of instantiation (as a parameter to the constructor). The class **`BoundedBuffer`** should provide two public methods named **`Load`** and **`Store`** which take an integer out of the buffer, or place an integer into the buffer, respectively.
Make sure to properly synchronize the **`Load`** and **`Store`** methods using your locks and condition variables implemented in the previous assignment! The needed locks and/or condition variables should be instantiated as members inside the **`BoundedBuffer`** class, and should be properly called by the **`Load`** and **`Store`** methods (such that a user of the buffer does not need to worry about any synchronization).

Deliverable 2: (20 points)

Submit your bounded buffer implementation (i.e. source file **`threadtest.cc`**) with proper comments.

Task 3: Test your implementation using a producer-consumer example

Modify/extend the code in file **`threadtest.cc`** such that it creates 2 producer and 2 consumer threads which communicate via one (shared!) bounded buffer.

The producer and consumer threads should run as two functions named **`Producer`** and **`Consumer`**, respectively. A producer thread should store a sequence of 10 integers into the buffer, whereas a consumer thread should load a sequence of 10 integers from the buffer and print them to the screen. To distinguish the numbers sent by the two producers, producer 1 should send the values 10, 11, 12, 13, … 19, whereas producer 2 sends 20, 21, 22, 23, … 29. Also, let’s limit the size of the bounded buffer to 5 integers.

To test your priority-based scheduler, assign priorities in such a way that the numbers printed on the screen by the consumers are in sequence (i.e. 10, 11, 12, … 29). As a second test case, assign priorities such that the sequence of numbers appears alternating as 10, 20, 11, 21, 12, 22, …, 19, 29.

Test your code thoroughly! Make use of the **`–rs <seed>`** option to test context switches at random times. Your program should produce the same output and run flawlessly in any case.

Hint: Add comments and DEBUG statements to your code in order to make your coding and debugging (and my grading) easier!

Deliverable 3: (20 points)

Submit your producer-consumer implementation (i.e. source file **`threadtest.cc`**) with proper comments. Also provide a brief description of your experiments along with a script of the running program (just cut/paste from your shell window) that shows that your implementation correctly produces the two number sequences listed above.

Submission instructions:

To submit your homework, send an email with subject "EECS 211 HW 3" to the course instructor at doemer@uci.edu. Please submit the deliverables listed above as attachments.

To ensure proper credit, be sure to send your email before the deadline:
February 8, 2007, 11:59pm (before midnight).

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