EECS 211
Advanced System Software
Winter 2005

Assignment 2

Posted: January 22, 2005
Due: January 31, 2005

Topic: Concurrency and Synchronization in Nachos

Instructions:

The goal of this second assignment is to develop, implement and test concurrency and synchronization primitives in the Nachos system. This assignment mostly follows the "Nachos Assignment 1" described in the file **`doc/thread.ps`** of the Nachos installation (see our previous assignment). The instructions below assume that you read **`doc/threads.ps`** in parallel.

Task 1: Understand the given framework
Go into the **`threads`** directory. Run the given program **`nachos`** to test the given code. Trace the execution path by reading through the given sources. Use also the debugger **`gdb`** to run the program step by step. Make sure you understand what is going on in the **`SWITCH`** function. Run the program also with the debug option (**`-d`**), and make yourself familiar with the other options (see the comments in file **`main.cc`**).

Task 2: Implement the missing locks and condition variables
See item 1 in **`doc/threads.ps`**. Complete the code for the classes **`Lock`** and **`Condition`** in file **`synch.cc`**. It may be helpful to look at the code in file **`synchlist.cc`** and **`synchlist.h`** to understand the use of locks (member **`lock`**) and condition variables (member **`listEmpty`**).

Task 3: Implement a producer-consumer example with a bounded buffer
See item 2 in **`doc/threads.ps`**. To implement this, replace/modify the code in file **`threadtest.cc`** such that it creates producer and consumer threads that communicate via a bounded buffer. The bounded buffer should be implemented as a class **`Buffer`** which allows the maximum number of characters in the buffer to be set at the time of instantiation (constructor parameter). Make sure to put sufficient comments and DEBUG statements in the code to make debugging (and grading!) easier.
Test your bounded buffer example with the following 3 cases:

1. 1 producer, 1 consumer, buffer size 10, test message "**`One producer and one consumer communicating!`**"

2. 1 producer, 2 consumers, buffer size 10, test message "**`One producer and two consumers communicating!`**"
3. 2 producers, 2 consumers, buffer size 10, test message "**`Two producers and two consumers communicating!`**"

For switching between these different test cases, use preprocessor directives. Test case 1 should be used if **`TEST1`** is defined, case 2 should be used if **`TEST2`** is defined, etc.
Test your code thoroughly! Make use of the **`-rs <seed>`** option to test context switches at different times. Your program should run flawlessly in any case.

Task 4: Implement a non-preemptive priority-based scheduler
See item 8 in **`doc/threads.ps`**. To implement the new scheduler, replace/modify the code in the files **`scheduler.cc`** and **`scheduler.h`**. The priority should be a non-negative integer value where 0 indicates the highest priority.

Test your scheduling algorithm with the consumer-producer example. Change the priorities of the consumer and producer threads and observe the difference in behavior with respect to the filling status of the buffer.

Deliverables:

1. Synchronization implementation: file **`synch.cc`**
2. Producer-consumer implementation: file **`threadtest.cc`**
3. Priority-based scheduler: files **`scheduler.h`**, **`scheduler.cc`**
4. ASCII-text file explaining your experiments and results: **`assignment2.txt`**

Detailed instructions on how to submit these files are listed on the next page!

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

Submission instructions:

We will use an electronic submission process to submit your deliverables.

To submit your files, go into your **`code`** directory, which is one level above the **`threads`** directory. There, type the command **`turnin`** which interactively allows you to submit the requested files. The **`turnin`** command will list source and text files in your homework directory and lets you turn in the deliverables one by one. An example session looks as follows:

```
doemer@east> setenv PATH /users/faculty/doemer/eecs211/bin:$PATH
doemer@east> cd eecs211/nachos-3.4/code
doemer@east> ls
Makefile          bin/              network/          userprog/
Makefile.common   filesys/          test/             vm/
Makefile.dep      machine/          threads/
doemer@east> turnin
=======================================
EECS 211 Winter 2005: "
Assignment "threads" submission for doemer
Due date: Mon Jan 31 23:59:59 2005
=======================================
Submit synch.h [yes, no]? n
Submit synch.cc [yes, no]? y
  File synch.cc has been submitted
Submit scheduler.cc [yes, no]? y
  File scheduler.cc has been submitted
[...]
=======================================
  Summary:
=======================================
You just submitted file(s):
   synch.cc
   scheduler.cc
You have not submitted file(s):
   synch.h
```

Note that this submission process will only work ***before the deadline*** for the assignment! To avoid missing the deadline, submit early. You may also submit incomplete work early, as you can always re-submit (overwrite!) later, until the deadline has passed.

Finally, you can use the following command to double-check which files have already been submitted by you:

```
doemer@east> ~eecs10/bin/listfiles.py
=======================================
EECS 211 Winter 2005: "threads" listing for doemer
=======================================
Files submitted for assignment "threads":
synch.cc
scheduler.cc
```