CSCI 3323 (Principles of Operating Systems), Fall 2016

Homework 3

Credit: 55 points.

1 Reading

Be sure you have read Chapter 2, sections 2.4 through 2.7, and skimmed Chapter 6.

2 Honor Code Statement

Please include with each part of the assignment the Honor Code pledge or just the word "pledged", plus one or more of the following about collaboration and help (as many as apply).[1] Text *in italics* is explanatory or something for you to fill in. For written assignments, it should go right after your name and the assignment number; for programming assignments, it should go in comments at the start of your program.

- This assignment is entirely my own work.
- This assignment is entirely my own work, except for portions I got from the assignment itself *(some programming assignments include "starter code")* or sample programs for the course *(from which you can borrow freely — that's what they're for).*
- I worked with *names of other students* on this assignment.
- I got help with this assignment from *source of help — ACM tutoring, another student in the course, the instructor, etc.*
- I got significant help from *outside source — a book other than the textbook (give title and author), a Web site (give its URL), etc.. ("Significant" here means more than just a little assistance with tools — you don't need to tell me that you looked up an error message on the Web, but if you found an algorithm or a code sketch, tell me about that.)*
- I provided significant help to *names of students* on this assignment. *("Significant" here means more than just a little assistance with tools — you don't need to tell me about helping other students decipher compiler error messages, but beyond that, do tell me.)*

3 Problems

Answer the following questions. You may write out your answers by hand or using a word processor or other program, but please submit hard copy, either in class or in one of my mailboxes (outside my office or in the ASO).

[1] Credit where credit is due: I based the wording of this list on a posting to a SIGCSE mailing list. SIGCSE is the ACM's Special Interest Group on CS Education.

1. (10 points) Solve the dining philosophers problem with monitors rather than semaphores. (Despite what I said in class about reading the literature, for this assignment do *not* look for a solution online or in another book; this is a problem you can and should try to solve just based on what we've done in this class.)

2. (10 points) Five batch jobs (call them A through E) arrive at a computer center at almost the same time, in the order shown below. Their estimated running times (in minutes) and priorities are as follows, with 5 indicating the *highest* priority:

job	*running time*	*priority*
A	10	3
B	6	5
C	2	2
D	4	1
E	8	4

   For each of the following scheduling algorithms, determine the turnaround time for each job and the average turnaround time. Assume that all jobs are completely CPU-bound (i.e., they do not block). (Before doing this by hand, decide how much of programming problem 1 you want to do.)

   - First-come, first-served (run them in alphabetic order by job name).
   - Shortest job first.
   - Round robin, using a time quantum of 1 minute.
   - Round robin, using a time quantum of 2 minutes.
   - Preemptive priority scheduling.

3. (10 points) Suppose that a scheduling algorithm favors processes that have used the least amount of processor time in the recent past. Why will this algorithm favor I/O-bound processes yet not permanently starve CPU-bound processes, even if there is always an I/O-bound process ready to run?

4. (10 points) Suppose you are designing an electronic funds transfer system, in which there will be many identical processes that work as follows: Each process accepts as input an amount of money to transfer, the account to be credited, and the account to be debited. It then locks both accounts (one at a time), transfers the money, and releases the locks when done. Many of these processes could be running at the same time. Clearly a design goal for this system is that two transfers that affect the same account should not take place at the same time, since that might lead to race conditions. However, no problems should arise from doing a transfer from, say, account A to account B at the same time as a transfer from account C to account D, so another design goal is for this to be possible. The available locking mechanism is fairly primitive: It acquires locks one at a time, and there is no provision for testing a lock to find out whether it is available (you must simply attempt to acquire it, and wait if it's not available). A friend proposes a simple scheme for locking the accounts: First lock the account to be credited; then lock the account to be debited. Can this scheme lead to deadlock? If you think it cannot, briefly explain why not. If you think it can, first give an example of a possible deadlock situation, and then design a scheme that avoids deadlocks, meets the stated design goals, *and uses only the locking mechanism just described.*

4 Programming Problems

Do the following programming problems. You will end up with at least one code file per problem. Submit your program source (and any other needed files) by sending mail to `bmassing@cs.trinity.edu` with each file as an attachment. Please use a subject line that mentions the course and the assignment (e.g., "csci 3323 hw 3" or "O/S hw 3"). You can develop your programs on any system that provides the needed functionality, but I will test them on one of the department's Linux machines, so you should probably make sure they work in that environment before turning them in.

1. (15 points)

   The starting point for this problem is a C++ program scheduler.cpp[2] that simulates execution of a scheduler, i.e., generates solutions to problems such as the one in the written part of this assignment. Comments describe input and desired output. Currently the program simulates only the FCFS algorithm. Your mission is to make it simulate one or more of the other algorithms mentioned in the written problem (FCFS, SJF, round robin using time quantums of 1 minute and 2 minutes, and preemptive priority scheduling). You will get full credit for simulating one algorithm, extra points for simulating additional algorithms. *Hint:* Keep in mind that arrival times may be nonzero, and it doesn't make sense to schedule a job that hasn't arrived yet.

   - Simple sample input[3].
   - Output for sample input[4].

   The starter code also makes use of some library classes (`string` and `vector`) that I think most of you have used but some of you may not have. `string` is functionally pretty similar to strings in languages such as Java and Scala; `vector` represents a templated expandable array (i.e., one with a type parameter that lets you specify the type of elements in the array). I'm cautiously optimistic that between the starter code, this toy example[5] of using `vector`, and what you can find on the Web about these classes (the Wikipedia articles seem okay), you will be able to use them to implement your choice of scheduling algorithm(s).

   If you don't remember, or didn't learn, how to compile C++ from the command line in Linux:

   ```
   g++ -Wall -pedantic scheduler.cpp
   ```

   (`-pedantic` is optional but does flag any nonstandard usage. `-Wall` is optional too but so potentially useful that I strongly recommend its use.)

2. (Optional — up to 10 extra-credit points) Write a program to test your solution to problem 1. See the writeup for the optional programming problem for Homework 2 for some suggestions about suitable platforms.

[2] `http://www.cs.trinity.edu/~bmassing/Classes/CS3323_2016fall/Homeworks/HW03/Problems/scheduler.cpp`

[3] `http://www.cs.trinity.edu/~bmassing/Classes/CS3323_2016fall/Homeworks/HW03/Problems/sample-in.txt`

[4] `http://www.cs.trinity.edu/~bmassing/Classes/CS3323_2016fall/Homeworks/HW03/Problems/sample-out.txt`

[5] `http://www.cs.trinity.edu/~bmassing/Classes/CS3323_2016fall/Homeworks/HW03/Problems/vector-example.cpp`