CSCI 3323 (Principles of Operating Systems), Fall 2018

Homework 3

Credit: 65 points.

1 Reading

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

2 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. (15 points) Solve the dining philosophers problem with monitors rather than semaphores. (I'm looking for pseudocode here, similar to what I showed in class for the bounded-buffer problem.) (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) 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?

3. (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.*

3 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. (30 points)

   The starting point for this problem is a C++ program http://www.cs.trinity.edu/~bmassing/Classes/CS that simulates execution of a scheduler, as we did for a simple example in class for the FCFS and SJF algorithms. Comments describe input and desired output. Currently the program simulates only the FCFS algorithm. Your mission is to make it simulate additional algorithms:

   - SJF without preemption.
   - Priority scheduling with preemption (and assuming that larger numbers mean higher priorities).
   - Round robin (with time quantum as a parameter).

   Sample inputs and outputs:

   - http://www.cs.trinity.edu/~bmassing/Classes/CS3323_2018fall/Homeworks/HW03/Problems/sa
     http://www.cs.trinity.edu/~bmassing/Classes/CS3323_2018fall/Homeworks/HW03/Problems/sa
   - http://www.cs.trinity.edu/~bmassing/Classes/CS3323_2018fall/Homeworks/HW03/Problems/sa
     http://www.cs.trinity.edu/~bmassing/Classes/CS3323_2018fall/Homeworks/HW03/Problems/sa

   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). If you don't know about these classes, ask me about supplementary reading.

   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.)

   Hint: I recommend that you take the approach I do for FCFS, basically simulating what the algorithm does by keeping a queue of ready processes. To me this seems like the most straightforward way to cope with nonzero arrival times, among other things. But you can do something else as long as it works.

4 Honor Code Statement

Include the Honor Code pledge or just the word "pledged", plus *at least one 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 programming assignments, this should go in the body of the e-mail or in a plain-text file `honor-code.txt` (no word-processor files please).

- This assignment is entirely my own work. *(Here, "entirely my own work" means that it's your own work except for anything you got from the assignment itself — some programming assignments include "starter code", for example — or from the course Web site. In particular, for programming assignments you can copy freely from anything on the "sample programs page".)*
- 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. (Here, "help" means significant help, beyond a little assistance with tools or compiler errors.)*
- I got help from *outside source — a book other than the textbook (give title and author), a Web site (give its URL), etc.. (Here too, you only need to mention significant help — 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 help to *names of students* on this assignment. *(And here too, you only need to tell me about significant help.)*

5 Essay

Include a brief essay (a sentence or two is fine, though you can write as much as you like) telling me what about the assignment you found interesting, difficult, or otherwise noteworthy. For programming assignments, it should go in the body of the e-mail or in a plain-text file `essay.txt` (no word-processor files please).

[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.