CSCI 3323 (Principles of Operating Systems), Fall 2012
Homework 3

Credit: 30 points.

1 Reading

Be sure you have read Chapter 2, sections 2.4 through 2.5.

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 my mailbox in the department office.

1. (5 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.

2. (5 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. (5 points) Solve the dining philosophers problem with monitors rather than semaphores. (Do this yourself, though, rather than looking for a solution online or in another book!)

4. (5 points) Restrooms are usually designated as men-only or women-only, but this requires having two restrooms if everyone is to be accommodated. A less expensive approach consistent with cultural norms in the U.S. would be to have one restroom with a sign on the door that indicates its current state — empty, in use by at least one woman, or in use by at least one man. If it is empty, either a man or a women may enter; if it is occupied, a person of the same sex may enter, but a person of the opposite sex must wait until it is empty. Write pseudocode for four functions to implement this approach: `woman_enter`, `man_enter`, `woman_leave`, and `man_leave`, to be used by the following pseudocode:

```
/* woman process */
while (TRUE) {
    woman_enter();
    use_restroom();
    woman_leave();
    do_other_stuff();
}
/* man process */
while (TRUE) {
    man_enter();
    use_restroom();
    man_leave();
    do_other_stuff();
}
```

   You can use any of the synchronization mechanisms we have talked about (shared variables, semaphores, monitors, or even message passing).

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 number and the assignment (e.g., "csci 3323 homework 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. (10 points) The starting point for this problem is a C++ program scheduler.cpp[1] 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.

   - Sample input[2].

[1] `http://www.cs.trinity.edu/~bmassing/Classes/CS3323_2012fall/Homeworks/HW03/Problems/scheduler/scheduler.cpp`

[2] `http://www.cs.trinity.edu/~bmassing/Classes/CS3323_2012fall/Homeworks/HW03/Problems/scheduler/sample-in.txt`

- Output for sample input[3].

I chose C++ for the starter code because in theory all of you have had at least some exposure to C++, and this might be a good opportunity for you to dust off that skill. The starter code also makes use of some library classes (`string` and `vector`) that you may not have worked with before. `string` is functionally pretty similar to strings in languages such as Java; `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[4] 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
```

However, feel free to rewrite anything about this program, including starting over in a language of your choice. Just remember that the program has to run on one of the department Linux machines, and it needs to accept input from command-line arguments and files — i.e., no GUIs, Web-based programs, etc. The latter requirement is to make it possible for me to automate testing your code. If you make changes to the format of the input — and I prefer that you don't — change the comments so they describe the changed requirements.

2. (Optional — up to 5 extra-credit points) Write a program to test your solution to either problem 3 or problem 4. If you want to do this using C and POSIX threads, you could start with the code for the programming problem in Homework 2. Or you could write in Java and use its monitor-based synchronization (synchronized methods/blocks plus `wait`, `notify`, and `notifyAll`) and/or features of the `java.util.concurrent` library package (which has, among many other things, a `Semaphore` library class). You can find some simple examples of multithreaded Java programs on the "Sample programs" page for my parallel programming class: http://www.cs.trinity.edu/ bmassing/CS3366/SamplePrograms/[5]. The bounded buffer example may be useful if you want to use monitor-based synchronization.

[3]`http://www.cs.trinity.edu/~bmassing/Classes/CS3323_2012fall/Homeworks/HW03/Problems/scheduler/sample-out.txt`

[4]`http://www.cs.trinity.edu/~bmassing/Classes/CS3323_2012fall/Homeworks/HW03/Problems/scheduler/vector-example.cpp`

[5]`http://www.cs.trinity.edu/~bmassing/CS3366/SamplePrograms/`