CSCI 4320 (Principles of Operating Systems), Fall 2007
Homework 3

Assigned: September 26, 2007.

Due: October 8, 2007, *at classtime. Not accepted late.*

Credit: 30 points.

1 Reading

Be sure you have read Chapter 2, sections 4 through 7, and Chapter 3.

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. 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.
   - Priority scheduling.

2. (5 points) Recall that some proposed solutions to the mutual-exclusion problem (e.g., Peterson's algorithm) involve busy waiting. Do such solutions work if priority scheduling is being used and one of the processes involved has higher priority than the other(s)? Why or why not? How about if round-robin scheduling is being used? Why or why not? Notice that a process can be interrupted while in its critical region; if that happens, it is considered to still be in its critical region, and other processes wanting to be in their critical regions are supposed to busy-wait.

3. (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?

4. (5 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 number and the assignment (e.g., "csci 4320 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 Fedora 7 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 program scheduler.cpp[1] that simulates execution of a scheduler, i.e., generates solutions to problem 1. Currently the program simulates only the FCFS algorithm. Your mission is to make it simulate one or more of the other algorithms mentioned in problem 1. You will get full credit for simulating one algorithm, extra points for simulating additional algorithms.

   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 standard input — i.e., no GUIs, Web-based programs, etc. The latter requirement is to make it easier for me to test your code, at least partially automatically. 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.

[1] `http://www.cs.trinity.edu/~bmassing/Classes/CS4320_2007fall/Homeworks/HW03/Problems/scheduler.cpp`