CSE451, Winter 2005
Homework #3

Out: Monday January 24th, 2005
Due: Monday January 31st, 2005

1. Suppose that the following processes arrive for execution at the times indicated. Each process will run with a single burst of CPU activity (i.e., no I/O) which lasts for the listed amount of time.

process	arrival time	CPU burst time	priority
p1	0ms	18ms	2
p2	1ms	12ms	1
p3	20ms	16ms	3
p4	31ms	14ms	4

a. What is the job throughput, average waiting time and average turnaround time for these processes with non-preemptive, FCFS scheduling?

b. With preemptive RR (quantum = 10ms) scheduling? (Different strategies might be used to add a newly submitted process to the ready queue. Explain what strategy you're using.)

c. With preemptive priority scheduling (given the above priorities, assuming higher numbers mean higher priority)?

2. Consider the Sleeping-Barber Problem (p233, question 6.11 in the textbook,) with the modification that there are k barbers and k barber chairs in the barber room, instead of just one. Write a program to coordinate the barbers and the customers using Java, C, or pseudo-code. You can use either semaphores or monitors.

*(Hint: there are many solutions that will work, some of which that will perform better than others. Don't worry about performance, just correctness; of course, deadlock performs poorly, and is also **incorrect** behavior. My only performance requirement is that barbers must be able to work in parallel, so it's probably a bad idea for a barber to be in a mutex while cutting hair).*

3. “Spot the bugs.” Consider the C source code we have made available at:

http://www.cs.washington.edu/education/courses/cse451/CurrentQtr/homework/buggycode.c

The code consists of a producer thread and a consumer thread running concurrently. Both threads have access to a shared buffer and a few shared variables. The code makes use of pthread functions to create threads, create and exercise locks, and to wait for threads to terminate – you can find out more information about the functions used by reading the appropriate man pages.

You can compile and run this code: on linux, save the code in a file called “buggycode.c”, and compile and run it as follows:

```
bash$   gcc –o buggycode buggycode.c –l pthread
bash$   ./buggycode
```

a. Find all of the bugs that you can in this code. For each bug, explain (1) why it is a bug, (2) what could go wrong because of the bug, and (3) whether this bug will deterministically cause erroneous behavior or not.

b. For each bug that you find, explain (briefly, in prose) what you would do to fix it.

c. The code as shown uses two locks. Could a solution be written that uses only one lock? What are the advantages and disadvantages of using two vs. one lock?

d. (harder) Is there a solution that is race-free but uses no locks (or other synchronization primitives) at all?

4. Use pseudo-code to implement:

- monitors using semaphores
- semaphores using monitors

Your solution may *not* busy-wait.

5. (purely optional suggested exercise problems) Multithreaded programming and synchronization is tricky. The only way to get good at it is to practice and learn from mistakes. I strongly recommend that you get practice by doing many of the textbook exercises. The following in particular will help you prepare for potential midterm questions well:

- 6.11, 6.13, 6.16, 6.22, 6.27, 7.1, 7.2

The following will also help with the fundamental concepts:

- 6.3, 6.4, 6.9

You do not need to turn in answers for any of these – this list is just a suggestion for the purposes of your own study.