CSc 422 — Homework 1

Due Tuesday, February 5, 2002

This assignment is worth 40 points. The first four problems are worth 5 points each; the two programs are worth 10 points each. Turn in written answers to the first four problems. Also turn in nicely-commented listings of your programs *as well as* answers to the questions I ask about each program. In addition, submit your programs electronically. See the end of this assignment for information on programming style and electronic turnin.

You may discuss the meanings of questions with classmates, but the answers and programs you turn in must be yours alone. For the exercises from the book, explain your answers clearly and succinctly.

1. MPD book, Exercise 2.5.

2. MPD book, Exercise 2.12.

3. MPD book, Exercise 2.18.

4. MPD book, Exercise 2.25.

5. **Atomic vs. Nonatomic Execution.** The purpose of this problem is to let you see the effects of not protecting critical sections of code. Write your programs either in C with the Pthreads library or in the MPD language. Compile your programs on Lectura but run your tests on Parallel so that threads execute concurrently. Parallel (`par`) has 6 processors; details are on the class Web page.

Assume that `x`, `y`, and `z` are *shared* integer variables, and that all are initially zero. Consider the following three statements:

```
S1:  x = x + 1;
S2:  y = y + 1;
S3:  z = z + x - y;
```

(a) Write a program that has `numWorkers` processes (threads). Each process executes the above three statements `numIters` times. Both `numWorkers` and `numIters` should be command-line arguments, in that order. At the end of the program, write out the final values of the three variables. Execute your program for one, two, and three workers, and for 1000, 2000, and 5000 iterations. What do you observe? Why?

(b) Modify your program to use two atomic actions, as follows:

```
⟨S1; S2;⟩
⟨S3;⟩
```

Use a semaphore or a mutex lock to protect the two critical sections in each process. (They are not disjoint, so you need to protect them with the same semaphore or lock to make them appear to be atomic.) Repeat the same set of experiments you used for part (a) and answer the same questions.

6. **Unix Tee Command**. Write Pthreads or MPD programs to solve the problem described in Exercise 2.3 of the text. Do just parts (a) and (c); namely, write a sequential program, and then write a concurrent program that uses the "while inside co" style. Your concurrent program will have three processes: one for reading standard input, one for writing to standard output, and one for writing to a file. As specified in the exercise, you are to use two buffers.

Use busy waiting (flag variables) to synchronize use of the buffers. Be careful programming the synchronization, because both output processes need to write from a buffer before the input process can refill it. Do not use semaphores for this assignment. (If you already know how to use semaphores, you might want to use them for initial debugging, and then replace them by flags.)

The buffer size should be a command-line argument. It can be either the number of bytes or number of lines, whichever you prefer. Again develop your programs on Lectura, then try running them on Parallel. Test your program with both small and large files and with a few different buffer sizes. What do you observe? Why? (You will not get any performance improvement for this problem; most likely your programs will run quite a bit slower because of synchronization overhead.)

Programming Style. Your programs should be easy to read and self-contained. At a minimum you should:

(1) Include a descriptive header comment with your name, a brief description of the program, and the command-line arguments.

(2) Give short but descriptive comments for each variable, process, procedure, and significant block of code.

(3) Use a *reasonable* level of indentation but not too much. I think 3 or 4 spaces is plenty; 8 is usually too many. Your printed listing should not have huge amounts of white space and should not have long lines that get wrapped around when printed.

Be sure to include your name in the header comment.

Electronic Turnin. Use the `turnin` program on Lectura to submit electronic versions of your program so that we can run some tests. See the man page for `turnin` for details.

For problem 5, the assignment name is `hw1.prob5`. The file names should be `nolocks.c` and `locks.c` (or `nolocks.mpd` and `locks.mpd`) for parts (a) and (b), respectively.

For problem 6, the assignment name is `hw1.prob6`. The file names should be `tee.sequential.c` (or `.mpd`) and `tee.concurrent.c` (or `.mpd`).