CSc 422 — Homework 4

Due Tuesday, April 23, 2002

This assignment is again worth 40 points. The first problem is worth 10 points; the programs are worth 15 points each.

Please turn in paper copies of your programs along with your answers. Also submit your programs electronically as described at the end of the assignment.

1. **Replicated Files.** Consider the replicated files example in Figure 8.15 (Section 8.4.2). The `FileServer` module in that figure is programmed using the multiple primitives notation described in Section 8.3; this is essentially the same as the full MPD language.

(a) The code in the `write` procedure uses `call` to update remote copies. Suppose the `call` of `remote_write` were replaced by asynchronous `send`. Would the solution still work? If so, explain why. If not, explain why not.

(b) The code in the file server has one lock for each copy of the file. A reader needs to get only the local read lock; a writer needs to get `n` write locks, one for every copy of the file. Show how to modify the code to use *weighted voting*, which is defined at the end of Section 8.4. Assume `readWeight` and `writeWeight` are the values of the read and write weights.

(c) Suppose you were asked to recode the file server using only remote procedure call, such as Java RMI or the RPC notation defined in Section 8.1. In other words, you cannot use `send` or rendezvous. Show how you would change the code. You may assume you have synchronized methods, monitors, or semaphores for any synchronization you might need within the `FileServer` module.

2. **Odd Person Wins Game**. Suppose you have three people and need a fair way to pick one of the three to win some prize or do some task. A common method is for each of them to flip a coin. If one coin comes up differently that the other two, that person wins. If all coins come up the same, you keep flipping until there is a winner. We can make a game out of this by repeating the contest a number of times and seeing who wins the most times.

Write a distributed program that uses processes and message passing to simulate the above game. You may write your program in Java with sockets, in MPD, in C with the MPI library, or in C with sockets. Test your program on either Lectura or Parallel. I suggest Parallel for MPI, Lectura for Java, and either machine for MPD or C plus sockets.

Use one process for each player, and have the players play several games. Use a random number generator to flip coins. The players *must* interact directly with each other using message passing. Do *not* use an additional coordinator process, and do not share any variables except for constants and the command-line argument. The command-line argument should be `numGames`, the number of games to play. After this many games have been played, print the total number of coin flips won by each player and the total number of ties.

In addition to your program listing, please turn in output from a few tests, including ones where `numGames` is 800 and 1600.

3. **Distributed File Server.** Develop a distributed program that has multiple clients and a file server. The file server manages a single file and allows clients to read *or* write that file. The file can be accessed by at most one client at a time. A client first opens the file for reading or writing. A file reader client reads the entire file and writes its contents to standard output. A file writer client reads lines from standard input up until a line that contains only a single dot "`.`" and writes the lines to the file. After reading or writing the file, a client closes the file; at this time it can be opened by another client (or the same one).

Write your program in Java or MPD. If you use Java, you must use RMI. If you use MPD, you must use virtual machines. Each client and the server should be capable of executing on different machines. You do not actually have to use multiple machines, but it should be easy to modify your program to do so.

The text describes three Java programs that you will find useful; see Sections 5.4, 7.9, and 8.5. If you use Java, I suggest that you also take a look at Exercise 7.23, which describes experiments with the remote file reader program in Section 7.9. You may also wish to take a look at exercises 5.27 and 8.21, which deal with the other Java programs in the text.

The MPD tutorial describes a concurrent grep program that uses virtual machines, and I also handed out a distributed exchange program in class. The programs in Section 8.4 are also very similar to ones in MPD.

Hand in a listing of your program *and* a one or two page explanation of your design decisions, the tests you ran, and the results you observed.

Electronic Turnin. Use the `turnin` program on Lectura to turn in your programs. The assignment names are `hw4.prob2` and `hw4.prob3`. Use whatever file names you wish. Either put a "usage" comment at the top of each program, or turn in README files explaining how to compile and execute the programs.