CSc 422/522 — Homework 4

due Tuesday, April 20

The first two problems are worth 10 points each. The last two problems are worth 20 points each. Graduate students are to solve all problems (60 points). Undergraduates are to solve any combination that adds to 40 points.

As usual, the work you turn in must be your own. Please explain your answers and programs.

1. The last assignment introduced the Roller Coaster and Savings Account problems. Solve the following problems. You may write programs, but paper solutions are fine.

(a) Develop a solution to the roller coaster problem using asynchronous message passing. Show the code for both the passengers and the car.

(b) Develop a solution to the roller coaster problem using the rendezvous primitives in SR (see Section 8.3 of the text). Show the code for both the passengers and the car.

(c) Develop a solution to the savings account problem using asynchronous message passing. Represent the bank as a process. Give the code for the bank and the code a customer would execute to do a deposit or a withdrawal.

(c) Develop a solution to the savings account problem using the rendezvous primitives in SR. Again represent the bank as a process. Give the code for the bank and the code a customer would execute to do a deposit or a withdrawal.

2. Consider the file server program in Figure 7.10. In the program, all the file servers receive from channel **`open`**. In addition, each open file is handled by a different file server. Answer the following as independent questions. You may need to introduce additional processes.

(a) Suppose each channel can have only one receiver. Modify the file server program to provide the same functionality with this restriction.

(b) Suppose that there is only *one* file server, period. It can handle up to **`n`** open files. Modify the code so that all clients share the same server. Identify all additional information you would need to have.

(c) Suppose that each *different* open file is handled by a different file server, but that if the same file is opened by two clients, they both interact with the same server. Modify the code to implement this design. Identify all additional information you would need to have.

3. Section 7.4 of the text describes three different ways to have **`n`** processes exchange values. In this problem you will implement and determine the performance of similar programs. You may write your programs either in SR or in C plus the MPI message-passing library. If you use MPI, you may use the group communication routines if you wish. *The processes in your program may not share any variables.*

The problem you are to solve is computing the sum of the local values in each process (this is called a plus reduction). Let the processes be numbered from **`1`** to **`n`**. Assume the initial value in process **`i`** is set to **`i`**. After every round described below, a process increments its local value. The first sum will thus be **`n*(n+1)/2`**, the second sum will be **`n`** more than that, and so on.

(a) Implement three programs for solving this problem. Use the centralized, symmetric, and circular ring algorithms given in Figure 7.11, 7.12, and 7.13. Each process should execute **R** rounds of exchanges, where the value of **R** is a command-line argument. The output of your program should be the final sum and the execution time of your program, from just before you fork the processes until just after they complete **R** rounds.

(b) A much more realistic situation would be a program in which there is significant computation before exchanging information. Modify your programs to include this. In particular, add a command-line argument **T** and add a loop before each exchange that simply increments a dummy variable **T** times. Again output the final sum and the elapsed execution time.

(c) Run your programs on Par for **n** equals 4 and **n** equals 8. (The latter program will have context switching, but so be it.) For the programs in part (a), use a value of **R** that is big enough that you can see if there is any difference between the programs. For the programs in part (b) use a value of **T** that is large enough so at least half of the execution time is spent looping rather than exchanging values, and use a value of **R** that is big enough that you can see if there is any difference between the programs.

(d) Submit you programs for part (b), the output of your tests for part (c), a table that summarizes the output, and a *brief* description of what your results show and why. You do not need to write a report.

4. Section 9.1 of the text describes the manager/workers technique for implementing a distributed bag of tasks. In this problem you are to write a manager/workers program to solve *either* of the problems below. Parameterize your program by the number of workers and test your program on Lec or Par or both at the same time. It is up to you to define what constitutes a task.

You may write your program in SR, in C plus the MPI message-passing library, or in Java plus the network or remote method invocation packages. *The processes in your program may not share any variables except as specified below.*

Problem 1. Determine the number of words in **`/usr/dict/words`** that have unique letters, namely the number of words in which no letter appears more than once. Print the number of such words. Also print all those that are the longest. You may read the dictionary file into variables that are shared by the workers. (It contains 25143 words.)

Problem 2. Consider the following game, which is played on a square grid of points. There are two players, red and blue. A *move* is drawing a horizontal or vertical line between a pair of points that are not already connected. Red draws red lines, and blue draws blue lines. Red plays first and makes two moves. Blue then makes one move. Red then makes two moves. Blue makes one more move. And so on.

The object of the game is for red to enclose a region of the grid with red lines. If blue is a perfect player, what is the largest area that red can enclose? Parameterize your program by the size of the grid. Solve the problem for the largest grid that you can handle in a reasonable amount of time? (The number of moves grows exponentially with the size of the grid.)