Homework 2

Assignment due:

Problems 1, 2, 3: 9:00 pm, Thursday, September 28th, 9:00 pm

Programming Exercises: Thursday, October 5th, 9:00 pm

Problems 1, 2, 3:

Submit typed (not hand-written) answers to the non-programming problems. Prepare a file named **`answers.pdf`** containing your answers, and submit it to the D2L Assignment folder. For these three exercises, explain your answers clearly and succinctly. Problems 1-3 are worth 36 points (12 points each).

Programs:

Submit the program solutions electronically. The programming exercises and associated report are worth 64 points.

A reminder: Homework assignments are individual efforts. The two projects can be done in teams of two, not the homeworks. You may discuss the meanings of questions with classmates, but the answers and programs you turn in are to be yours alone.

For the exercises from the book, explain your answers clearly and succinctly.

Problem 1: Exercise 3.4, page 143

Suppose a computer has an atomic Compare-and-Swap instruction, which has the following effect:

```
CSW(a, b, c):
   < if ( a == c ) {
        c = b;
        return (0);
     }
     else {
        a = c;
        return (1);
     } >
```

Parameters **`a`**, **`b`**, and **`c`** are simple variables, such as integers. Using **`CSW`**, develop a solution to the critical section problem for **`n`** processes. Do not worry about the eventual entry property. Describe clearly how your solution works and why it is correct.

Problem 2: Exercise 4.11, page 194

It is possible to implement a reusable n-process barrier with two semaphores and one counter. These are declared as follows:

```
int count = 0;
sem arrive = 1, go = 0;
```

Develop a solution. (*Hint*: Use the idea of passing the baton.)

Problem 3: Exercise 4.20, page 196

Consider the following writers' preference solution to the readers/writers problem [Courtois et al. 1971]:

```
int nr = 0, nw = 0;
sme m1 = 1, m2 = 1, m3 = 1;  # mutex semaphores
sem read = 1, write = 1;     # read/write semaphores
```

Reader processes:

```
P(m3);
P(read);
P(m1);
nr = nr + 1;
if ( nr == 1 ) {
   P(write);
}
V(m1);
V(read);
V(m3);
```

read the database

```
P(m1);
nr = nr - 1;
if ( nr == 0 ) {
   V(write);
}
V(m1);
```

Writer processes:

```
P(m2);
nw = nw + 1;
if ( nw == 1 ) {
   P(read);
}
V(m2);
P(write);
```

write the database

```
V(write);
P(m2);
nw = nw - 1;
if ( nw == 0 ) {
   V(read);
}
V(m2);
```

The code executed by reader processes is similar to that in Figure 4.10, page 170, but the code for writer processes is much more complex.

Explain the role of each semaphore. Develop assertions that indicate what is true at critical points. In particular, show that the solution ensures that writers have exclusive access to the database and a writer excludes readers. Give a convincing argument that the solution gives writers preference over readers.

Programming Exercises:

Write two solutions to the Jacobi problem: one in C and the other in Java. Use threads and semaphores in both solutions.

Use the same algorithm for both solutions. This will allow you to better compare the execution times of C vs. Java.

Use semaphores in both solutions to implement the barrier.

Implement a Dissemination barrier for both solutions.

You can use Java's Semaphore class. Do ***not*** use the CyclicBarrier class -- write your own Dissemination barrier.

Reading from textbook:

Sections 3.5.3, pages 129 to 131.
Sections 11.1.1 to 11.1.3, pages 534 to 541 (and figure 11.3 on page 542).
If you do not have access to a copy of the textbook :-(, scanned images of these pages are available on D2L->Content->Homework->Homework #2->Hw2-Jacobi-pages.

Command-line arguments:

There are two required:

`N` = the grid will be of size **`N x N`**.

`numProcs` = number of worker processes (threads) to create to solve the problem. Your program should work for all values of **`numProcs`** from **`1`** through **`32`**. (Note: all values of **`numProcs`** from **`1`** to **`32`**, not just the powers of two.)

The remaining command-line arguments are up to you. They should have default values so the user can choose to supply only the required two. As an example, my current solution has 4 optional arguments that are the fixed values for the left, top, right, and bottom edges of the grid, and one additional argument which is the epsilon value used to determine when to stop the computation. With my solution, typing:

```
jacobi 32 4
```

is equivalent to typing:

```
jacobi 32 4 10.0 10.0 800.0 800.0 0.1
```

which gives:

```
N = 32
numProcs = 4
left = 10.0, top = 10.0, right = 800.0, bottom = 800.0
epsilon = 0.1
```

Output:

Print to a file named *JacobiResults.txt* the settings (**`N`**, **`numProcs`**, edge values, **`epsilon`**, anything else that seems useful), and the final grid contents.

Print to standard out **`numProcs`**, the number of iterations and the computation time.

Compilation:
Submit a **`Makefile`** that has a target named **`all`**. The command **`make all`** should produce the executables for both of your solutions. For the Java version, **`make all`** should produce all needed **`.class`** files.

Timing tests:
Run timing tests for both programs. Run timing tests for 1, 2, 3, 4, etc. workers. Some questions that your timing results should be able to answer:

> Do you see speed up as the number of workers increases?
> How do the times for your C solution compare with your times for Java?
> Do the times decrease or increase as you increase the number of workers?

Submit a report (using **`D2L`**) named *timing.pdf*. This will contain the results of your timing runs. We should be able to figure out the answers to the above questions from the timing results that you supply. Charts can be included where useful. Include in this file:

- the command-lines that produced the timing runs reported,
- the machine in the CS department that you used for the timing runs,
- an explanation of the command-line arguments,
- the default values for missing arguments,
- answers to the questions given above.
- anything else relevant to our understanding of your results.

Patrick's executable:
The C version of my solution is available in the /home/cs422/fall17/hw2 directory. There is both an OSX and Linux version.

Turnin:

Note: We are asking for submission of programs via D2L only.

- For the written answers to questions 1, 2, and 3, submit **`answers.pdf`**. Use the **`Homework2-Problems`** Assignment folder.
- For the Jacobi programs, submit your **`timing.pdf`** report and the individual files for your programs (**`.java`**, **`.c`**, **`.h`**, **`Makefile`**). By typing **`make all`**, we should be able to create both the Java and the C executables. Use the **`Homework2-Jacobi`** Assignment folder. For the **`Makefile`**, re-name it to **`Makefile.txt`** so D2L will accept it.
- Logon to D2L. Select CSc 422. Click on the "Add a File" button. A pop-up window will appear. Click on the "Choose File" button. Use the file browser that will appear to select your file(s). Click on the "Upload" button in the lower-right corner of the pop-up window.
- You are now back at "Submit Files ". Click on "Upload" in the lower-right of this window. You should now be at the "File Upload Results" page, and should see the message **`File Submission Successful`**.

D2L will send you a confirmation email after each item is placed in the Assignment folder. This email is sent to your UA email (which is <yourUANetId>@email.arizona.edu). Retain these

emails at least until you have received a grade for the assignment; it is your confirmation that you did submit the program(s).

You may submit the various items more than once. We will grade the last one that you put in the Assignment folder.