COSC 3750
Linux Programming
Homework 11, Matrix Multiply with PThreads

1 Intro

This is the final project for the course, read "final exam". You will write a program that does a simple matrix multiply. Depending on the arguments it will do one of two things:

1. create some number of worker threads which will perform the actual matrix multiply. When they are done the result matrix will placed in a file.
2. create NO threads and perform the matrix multiplication by itself. Again, the result will be written to a file.

This does not sound too difficult. But, you must write a program that will do this with **any** size matrix AND it will do everything that I say, the way that I say to do it.

I will supply program source code that will create a matrix in the correct format. You don't need anything special to create the executable. If you use a fourth argument (an integer), that will be the value of EVERY element in the array. So you can

```
make_mat matrix1 4 5 1
```

and get a 4x5 matrix of doubles and all those 20 values are 1.

2 Your job

2.1 Main Program:

The **main** program will do the following:

1. If you feel the need for additional mutexes, that is fine, but make SURE that you really need them. If you have too many, you will just get lost.
2. Create two integer variables that the threads will use to choose the element of the result matrix that they will produce. You must "protect" these with a mutex.

3. Create and initialize a mutex for those row/column variables. Pick some reasonable name that is probably unique. You should only have one mutex that will control access to both variables.

4. Create and initialize a mutex for writing to the output file. Pick some reasonable name that is probably unique. This will be used in conjunction with a condition variable (CV).

5. Create and initialize a CV for use in writing the output. Pick some reasonable name that is probably unique.

6. Avoid global variables. That means figure out a way to pass information to the threads via the argument to the thread function.

7. Process the command line arguments. The program will determine how many (if any) threads are to be created. Check the arguments. If there are 3 (argc = 4) then there are NO threads and the arguments are two input file names and the output file name, in that order. If there are 4 arguments, then the first argument is the number of threads and the file names as before.

8. Open the two input files and the output file.

9. Check the input files. If the matrices contained in them cannot be multiplied, print an informative error message and exit. Remember this means the number of columns of the first one have to match the number of rows of the second one. And the numbers of rows or columns must be positive.

10. Create the threads, if needed.

11. If there are no threads, the main program will perform the matrix multiply and write the results to the output file. In this case only, the mutexes and condition variable will not be used.

12. If there are threads, only they, NOT the main program, will compute the result. The "main" thread will wait on the condition variable (having locked that mutex first). When some thread has completed **some** amount of work, it will signal the mutex, the main thread will be allowed to run and that main thread will write the completed data out. Then it will wait again on the mutex.

13. The data in these matrix files will be "raw" data. That is, NOT ASCII. There are no delimiters. The first 4 bytes will be an integer number of rows, the second 4 will be an integer number of columns, the remainder of the file will be a row-by-column number of **doubles** that are the values.

14. Regardless of whether there are threads or not, the main thread will time the actual matrix multiply. This will NOT include the times to create/join the threads, open or close the files.

15. The main thread **will** do a *pthread_join()* on all (if any) of the workers. It will ensure that all files are properly closed. When it is ready to end, it will print a SIMPLE summary, on the order of

```
$> ./mmult 10 f1 f2 f3

Matrix sizes:
    M: 25
    N: 10
    P: 20

Worker threads: 10

Total time: 34 seconds.

$>
```

2.2 Worker:

Each **worker** thread will:

1. Lock the mutex that protects the row/column to be computed.
2. Retrieve its assignment and update the row/column as discussed in class.
3. Unlock the mutex and proceed to compute its element of the matrix. This result will be stored in some buffer accessible to worker and "main".
4. At a point determined by you, after some thread has computed the last of some set of elements, it will signal the CV that the main thread is waiting on. This is when the main thread writes data to the file. Somehow, you must ensure that this all works without a hitch. Might need more mutexes or CVs.
5. Repeat until the matrix multiplication is complete.
6. Once complete, each thread will call *pthread_exit()*.

3 What to turn in

You will need to upload 1 file, **hw11.tar** or **hw11.tgz**. It will contain at a minimum **mmult.c** and **Makefile**. If you decide you need to create some additional header or source code files that is fine. Just tar them together and upload that.