Programming Assignment 1: Multithreading
CSC 4103: Operating Systems, Spring 2019

Due Date: February 26th, Tuesday (by 11:59 PM)
Total Points: 10

Instructions: Compile and test-run your code on the classes server. Submit your work as instructed and verify your submission. The verify command will display your submission date/time. Include your name, email, and classes account in all source code files. Late submissions will be penalized at the rate of 10% per day late and no more than 3 calendar days late.

Objective
To learn the use of POSIX Pthreads or Java Threads

Background
Modern operating systems provide features for a process to use multiple threads to speed up accesses. In the class, we have learned the concepts associated with multithreaded computer systems, such as multithreading models. There are various thread libraries. POSIX Pthreads, and Java Threads are widely used for creating and manipulating threads. The textbook (Threads, Chapter 4) gives examples of multithreaded programs using these libraries.

Programming Task
In this assignment, you need to implement the **Project – Matrix Multiplication** given in the textbook (Chapter 4, 8th edition), which is also attached in this project description. Your program will calculate each element $C_{i,j}$ of the matrix product C of two matrices A and B using an individual/separate thread. So, the number of threads is equal to the number of terms in the matrix product. Use either **POSIX Pthreads or Java threads** to implement this program. The matrices to be multiplied are:

$$A = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix} \quad \text{and} \quad B = \begin{bmatrix} 8 & 7 & 6 \\ 5 & 4 & 3 \end{bmatrix}$$

Programming Language
C/C++ or Java

Submitting Your Work
All files you submit must have a **header** with the following (enclosed in comment lines):

Name:	Your Name (Last, First)
Email:	Your LSU email
Project:	PA-1 (Multithreading)
Instructor:	Feng Chen
Class:	cs4103-sp19
Login ID:	cs4103xx

You need to use the server "**classes.csc.lsu.edu**" to work on the assignment. You can login to your account in the server using SSH. Create a directory **prog1** (by typing **mkdir prog1)** in your home directory in which you create your program or source code.

Make sure that you are in the **prog1** directory while submitting your program. Submit your assignment to the grader by typing the following command:

~cs4103_chf/bin/p_copy 1

This command copies everything in your prog1 directory to the grader's account. Check whether all required files have been submitted successfully:

~cs4103_chf/bin/verify 1

Project 2: Matrix Multiplication Project

Given two matrices, A and B, where matrix A contains M rows and K columns and matrix B contains K rows and N columns, the matrix product of A and B is matrix C, where C contains M rows and N columns. The entry in matrix C for row i, column j ($C_{i,j}$) is the sum of the products of the elements for row i in matrix A and column j in matrix B. That is,

$$C_{i,j} = \sum_{n=1}^{K} A_{i,n} \times B_{n,j}$$

For example, if A is a 3-by-2 matrix and B is a 2-by-3 matrix, element $C_{3,1}$ is the sum of $A_{3,1} \times B_{1,1}$ and $A_{3,2} \times B_{2,1}$.

For this project, calculate each element $C_{i,j}$ in a separate *worker* thread. This will involve creating $M \times N$ worker threads. The main—or parent—thread will initialize the matrices A and B and allocate sufficient memory for matrix C, which will hold the product of matrices A and B. These matrices will be declared as global data so that each worker thread has access to A, B, and C.

Matrices A and B can be initialized statically, as shown below:

```
#define M 3
#define K 2
#define N 3

int A [M][K] = { {1,4}, {2,5}, {3,6} };
int B [K][N] = { {8,7,6}, {5,4,3} };
int C [M] [N];
```

Alternatively, they can be populated by reading in values from a file.

Passing Parameters to Each Thread

The parent thread will create $M \times N$ worker threads, passing each worker the values of row i and column j that it is to use in calculating the matrix product. This requires passing two parameters to each thread. The easiest approach with Pthreads and Win32 is to create a data structure using a `struct`. The members of this structure are i and j, and the structure appears as follows:

```
/* structure for passing data to threads */
struct v
{
   int i; /* row */
   int j; /* column */
};
```

Both the Pthreads and Win32 programs will create the worker threads using a strategy similar to that shown below:

```
/* We have to create M * N worker threads */
for (i = 0; i < M, i++)
   for (j = 0; j < N; j++ ) {
      struct v *data = (struct v *) malloc(sizeof(struct v));
      data->i = i;
      data->j = j;
      /* Now create the thread passing it data as a parameter */
   }
}
```

```
public class WorkerThread implements Runnable
{
    private int row;
    private int col;
    private int[][] A;
    private int[][] B;
    private int[][] C;

   public WorkerThread(int row, int col, int[][] A,
    int[][] B, int[][] C) {
      this.row = row;
      this.col = col;
      this.A = A;
      this.B = B;
      this.C = C;
   }
   public void run() {
      /* calculate the matrix product in C[row] [col] */
   }
}
```

Figure 4.15 Worker thread in Java.

The `data` pointer will be passed to either the `pthread_create()` (Pthreads) function or the `CreateThread()` (Win32) function, which in turn will pass it as a parameter to the function that is to run as a separate thread.

Sharing of data between Java threads is different from sharing between threads in Pthreads or Win32. One approach is for the main thread to create and initialize the matrices A, B, and C. This main thread will then create the worker threads, passing the three matrices—along with row i and column j—to the constructor for each worker. Thus, the outline of a worker thread appears in Figure 4.15.

Waiting for Threads to Complete

Once all worker threads have completed, the main thread will output the product contained in matrix C. This requires the main thread to wait for all worker threads to finish before it can output the value of the matrix product. Several different strategies can be used to enable a thread to wait for other threads to finish. Section 4.3 describes how to wait for a child thread to complete using the Win32, Pthreads, and Java thread libraries. Win32 provides the `WaitForSingleObject()` function, whereas Pthreads and Java use `pthread_join()` and `join()`, respectively. However, in these programming examples, the parent thread waits for a single child thread to finish; completing this exercise will require waiting for multiple threads.

In Section 4.3.2, we describe the `WaitForSingleObject()` function, which is used to wait for a single thread to finish. However, the Win32 API also provides the `WaitForMultipleObjects()` function, which is used when waiting for multiple threads to complete. `WaitForMultipleObjects()` is passed four parameters:

```
#define NUM_THREADS 10

/* an array of threads to be joined upon */
pthread_t workers[NUM_THREADS];

for (int i = 0; i < NUM_THREADS; i++)
   pthread_join(workers[i], NULL);
```

Figure 4.16 Pthread code for joining ten threads.

1. The number of objects to wait for
2. A pointer to the array of objects
3. A flag indicating if all objects have been signaled
4. A timeout duration (or `INFINITE`)

For example, if `THandles` is an array of thread `HANDLE` objects of size `N`, the parent thread can wait for all its child threads to complete with the statement:

```
WaitForMultipleObjects(N, THandles, TRUE, INFINITE);
```

A simple strategy for waiting on several threads using the Pthreads `pthread_join()` or Java's `join()` is to enclose the join operation within a simple `for` loop. For example, you could join on ten threads using the Pthread code depicted in Figure 4.16. The equivalent code using Java threads is shown in Figure 4.17.

```
final static int NUM_THREADS = 10;

/* an array of threads to be joined upon */
Thread[] workers = new Thread[NUM_THREADS];

for (int i = 0; i < NUM_THREADS; i++) {
   try {
     workers[i].join();
    } catch (InterruptedException ie) { }
}
```

Figure 4.17 Java code for joining ten threads.