EE 361C/382C: Multicore Computing

Instructor: Prof. Vijay Garg (email: garg@ece.utexas.edu)
TA: Aravind Srinivasan (email: aravindsrinivasan@utexas.edu)
TA: Madan Shringeri (email: skmadan@utexas.edu)
TA: Changyong Hu (email: colinhu9@utexas.edu)
TA: Xiong Zheng (email: zhengxiongtym@utexas.edu)

Assignment 2: Spring 2020
Deadline: Feb 20, 2020

This assignment has a programming component. The source code for each of the programming questions must be submitted online. The non-programming part (`question 1`) should also be submitted online. Please zip both the source code and non-programming part together, name it as EID1-EID2.zip and submit it to canvas before the assignment due date (i.e., **11:59pm Feb 20**). The assignment should be done in teams of two.

1. **(10 points)** Consider the following problem called *Renaming*. There are n processes which have indices in range $1..N$ where $N >> n$. We would like these processes to get a new name in the range of $1..M$ where $M = n * (n+1)/2$. Describe a function `int rename(int j)` that returns a unique name to each process in the range $1..M$. Every invocation of `rename()` should terminate and no two processes should obtain the same name. Your function should not use old index to *compute* a new index, i.e., it should satisfy the *index independence property* that if a process whose index is i obtains the new name v, then the process could have obtained the same new name v if its index had been j different from i. (Hint: Use enough splitters to split the input stream of threads).

2. **(30 points)** Write a program that uses n threads, where $n = 1, 2, 4, 8$. These threads increment a shared variable c. The total number of increment operations are $m = 120,000$. Each thread reads the value of c and increments it m/n times. Implement the following methods and compare the the total time taken for each of the following methods. Return the final count as c for each of the algorithm. Submit the plot as part of the assignment.
(a) CLH Queue Lock Algorithm.

(b) Lamport's Fast Mutex Algorithm.
(c) Anderson's Spin Lock Algorithm.

3. **(30 points)** A group of monkeys want to cross a river using an old rope. The rope connects the two sides of river bank and the monkeys can cross the river by climbing the rope hand-over-hand. The rope is too old, so it can only take at most *three* monkeys at any time. Otherwise, the rope breaks and the monkeys fall into the water. If a rightward monkey encounters a leftward monkey on the rope, they will fight each other and fall into the water. Furthermore, the monkeys are ruled by a monkey king named Kong. When Kong wants to cross the river, it waits until the monkeys on the rope leaves the rope. Meanwhile, all the monkeys that have not climbed on the rope have to wait until Kong has crossed the river.

   Implement the Java class `Monkey` that arranges the group of monkeys to cross the river safely using Java ReentrantLocks and Condition Variables.

```
public class Monkey {
    // declare the variables here

    // A monkey calls the method when it arrives at the river bank and
    // wants to climb the rope in the specified direction (0 or 1);
    // Kong's direction is -1.
    // The method blocks a monkey until it is allowed to climb the rope.

    public void ClimbRope(int direction) throws InterruptedException {

    }

    // After crossing the river, every monkey calls this method which
    // allows other monkeys to climb the rope.
    public void LeaveRope() {

    }
    /**
     * Returns the number of monkeys on the rope currently for test purpose.
     *
     * Positive Test Cases:
     * case 1: normal monkey (0 and 1)on the rope, this value should <= 3, >= 0
     * case 2: when Kong is on the rope, this value should be 1
     */
    public int getNumMonkeysOnRope() {

    }
}
```

4. **(30 points)** The purpose of this question is to learn OpenMP. To setup the API, you can install gcc-4.7 compiler on the Linux machine or Visual Studio 2008–2010 C++ on the Windows machine. For more information, please visit: `http://openmp.org/`. Alternatively, you can use TACC account for this question.

   (a) Write a C/C++ program `MatrixMult` that allows parallel multiplication for two matrices of doubles by using OpenMP. The task for your program is described below:

   Your program should accept three arguments. The first two arguments are paths of the input files that encode two matrices that need to be multiplied. The format of an input file is defined as follows: each input file contains one matrix. The first line of an input file contains two positive integers: m and n denoting the number of rows and columns in the matrix. The next m lines in the file provide rows of the matrices with entries separated by space. The third argument to your program, T, indicates the number of threads to be used. Suppose your program is named `run`, and is executed with the following parameters:

   ```
   ./run mfile1 mfile2 T
   ```

   The output of your program should be a matrix with the same format as the input matrices. Assume that matrices are of the proper form and can be multiplied. Submit a plot of time taken to compute the product of matrices of size 100 by 100 when the number of threads are varied from 1 to 8.

   (b) Write a multithreaded C/C++ function `Sieve` in OpenMP that calculates the number of prime numbers between 1 and `N` using the Sieve of Eratosthenes algorithm. The function returns the total number of prime numbers as an integer and its signature is given as follows:

   ```
   int Sieve(int N, int threads)
   ```

   The Sieve of Eratosthenes algorithm creates a list of integers from 1 through `N`. The algorithm starts with the smallest prime number `P` and marks its multiples between $\mathtt{P}^2$ through `N` as non primes. It then finds the next non marked integer greater than `P`, assigns it to P and repeats the previous step. The algorithm terminates if there is no non marked integer between `P` and $\sqrt{\mathtt{N}}$, it then counts the total number of primes in the list. Your function `Sieve` takes in two argument `N`, `threads` and finds all prime numbers less than or equal to `N`. Submit a plot of time taken to compute the number of primes when N is 100000000 and the number of threads are varied from 1 to 8