EE 361C/382C: Multicore Computing : Assignment 1: Spring 2020

Instructor: Professor 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)

Deadline: Feb 6, 2020

This assignment has a programming component. The source code (Java files) of the programming part must be submitted to Canvas before the end of the due date (i.e., 11:59pm **Feb 6**). Please zip and name the source code as EID1-EID2.zip. For each of the programming questions, you need to submit the source files. The non-programming part should be written or typed on a paper and submitted at the beginning of the class. The assignment should be done in teams of two.

0. **(1 point)**

   Create a TACC UserID and add it to googledoc on Canvas.
   Link: https://portal.tacc.utexas.edu/

1. **(4 points)** You are one of P recently arrested prisoners. The warden, a deranged computer engineer, makes the following announcement:

   - You may meet together today and plan a strategy, but after today you will be in isolated cells and have no communication with one another.
   - I have set up a "switch room" which contains a light switch, which is either *on* or *off*. The switch is not connected to anything.
   - Every now and then, I will select one prisoner at random to enter the "switch room." This prisoner may throw the switch (from *on* to *off*, or vice-versa), or may leave the switch unchanged. Nobody else will ever enter this room.
   - Each prisoner will visit the switch room arbitrarily often. More precisely, for any N, eventually each of you will visit the switch room at least N times.

- At any time, any of you may declare: "we have all visited the switch room at least once." If the claim is correct, I will set you free. If the claim is incorrect, I will feed all of you to the crocodiles. Choose wisely!

(a) Devise a winning strategy when you know that the initial state of the switch is *off*.

(b) Devise a winning strategy when you do not know whether the initial state of the switch is *on* or *off*.

2. **(20 points)** Use SPIN to create counter-examples to show that any of the following modifications to Peterson's algorithm makes it incorrect:

   a) A process in Peterson's algorithm sets the *turn* variable to itself instead of setting it to the other process.

   b) A process sets the *turn* variable before setting the *wantCS* variable.

   Submit the trail generated by the SPIN program.

3. **(10 points)** Show that the **Filter** lock allows some threads to overtake others an arbitrary number of times. In other words, if a thread is inside the filter and does not execute the filter program for an interval of time, then some other threads can enter and exit the critical section multiple times during that interval. (Hint: Consider executions with at least three threads.)

4. **(15 points)** The l-exclusion problem is a variant of the starvation-free mutual exclusion problem. We make two changes: as many as l threads may be in the critical section at the same time, and fewer than l threads might fail (by halting) in the critical section. An implementation must satisfy the following conditions:

   l-Exclusion: At any time, at most l threads are in the critical section.

   l-Starvation-Freedom: As long as fewer than l threads are in the critical section then some thread that wants to enter the critical section will eventually succeed (even if some threads in the critical section have halted).

   Modify the n-process **Filter** mutual exclusion algorithm to turn it into an l-exclusion algorithm.

5. **(Programming, 20 points)** Write a Java class that computes frequency of an item in an array of integers. It provides the following `static` method:

```
public class Frequency {
  public static int parallelFreq(int x, int[] A, int numThreads) {
    // your implementation goes here.
  }
}
```

This method creates as many threads as specified by `numThreads`, divides the array `A` into that many parts, and gives each thread a part of the array. Each thread computes the frequency of `x` in its own subarray in parallel. The method should return the overall sum of the frequencies. Use *Callable* interface for your implementation.

6. **(Programming, 30 points)** Write a program that uses n threads, where $n = 1..8$. These threads increment a shared variable c. The total number of increment operations are $m = 1,200,000$. Each thread reads the value of c and increments it m/n times. For the case that m/n is not an integer, divide the work as even as possible and make sure the total count is m. Implement the following methods and compare the the total time taken for each of the following methods for $n = 1..8$.

```
public class PIncrement {
    public static int parallelIncrement(int c, int numThreads) {
        // your implementation goes here
    }
}
```

Submit the plot as part of the assignment.
(a) Lamport's Bakery Algorithm.
(b) Java's AtomicInteger (with `compareAndSet` method).
(c) Java's synchronized construct
(d) Java's Reentrant Lock
Hint for part (a): use `AtomicBoolean` or `AtomicInteger` for shared variables to guarantee atomicity. You can only use `get()` and `set()` methods in this part.