Programming Assignment 2 [30 pts]

COMP4270/6270 Operating Systems – Spring'20

Due 4/8/2020

Instructor: Myounggyu Won

1. [5 pts] Design a multithreaded C program that creates 5 threads. Each thread generates a random point (x, y) where $0 < x, y < 1$. The output of this program should be 5 random points generated by the 5 threads. For example,

(0.1, 0.4)
(0.2, 0.1)
(0.001, 0.0038)
(0.33, 0.23)
(0.31, 0.8)

2. [5 pts] Extend the program you created for Problem 1 by checking if generated points are inside a quarter of a circle of radius 1 with origin at (0, 0) as shown in Figure 1.

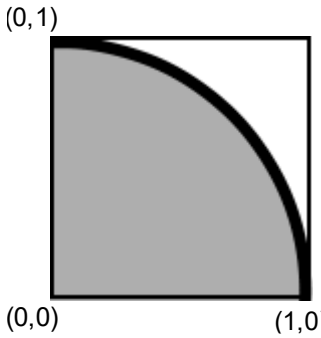

Figure 1. The area of a quarter of a circle of radius 1.

The output of this program should be 5 random points plus 'Yes' or 'No' for each point where 'Yes' indicates the point is inside the circle, and 'No' for otherwise. For example,

(0.1, 0.4) Yes
(0.2, 0.1) Yes
(0.001, 0.0038) Yes
(0.33, 0.23) Yes
(0.9, 0.9) No

3. [5 pts] We learned about the theory on how to use mutex locks to solve the critical section problem. Read the following web page to learn about how to "code" with mutex locks, i.e., how to guard a critical section with mutex locks.

https://www.geeksforgeeks.org/mutex-lock-for-linux-thread-synchronization/

4. [15 pts] Extend the program you created for Problem 2 such that the program now calculates π. The idea for calculating π is as follows.

The area of the whole square is one, while the area of the part inside the circle is π/4 (See Figure 1). If we choose a point in the square at random, the probability that it is inside the circle is π/4. If we choose `N` points in the square at random, and if `C` of them are inside the circle, we expect the fraction `C/N` of points that fall inside the circle to be about π/4. That is, we expect `4*C/N` to be close to π. If `N` is large, we can expect `4*C/N` to be a good estimate for π, and as `N` gets larger and larger, the estimate is likely to improve [1].

Specifically, declare a global variable C which is initialized to 0. Note that this global variable is shared by the 5 threads. Each thread generates 100,000 random points, and if a generated point falls inside the circle, then the global variable C is incremented by 1. Once all threads finish generating 100,000 random points, the variable C will contain the total number of points out of 500,000 that are inside the circle. After the 5 threads finish running, the main thread calculates and prints out the value of π.

Hint. Use the pthread_join() function within the main function to wait for all the 5 threads to terminate.

A key challenge of this problem is to guard the critical section where the variable C is accessed and modified using a mutex lock.

Hint1. Create and initialize a one global mutex lock (refer to Problem 3) and use it to guard the critical section.
Hint2. The same start routine can be used for all 5 threads, because these threads do the same thing, i.e., generating 100,000 random points, checking if those points are inside the circle, and modifying the variable C accordingly.

The correct output of this program should be 3.14, or a value close to this.

Files to submit:

- Firstname_p1.c
- Firstname_p2.c
- Firstname_p4.c

References

[1] http://math.hws.edu/javanotes/c12/exercises.html