University of Windsor

School of Computer Science

60-330 Operating Systems Fundamentals - Winter 2018
Assignment 3

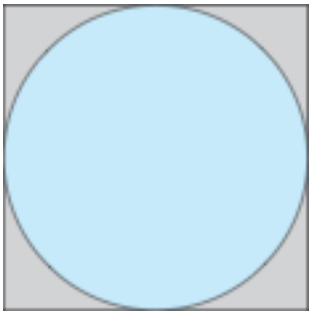
Deadline: **Friday February 16 2018 at 11:59pm**

Rules: This assignment must be submitted and will be evaluated individually.

Objectives: The aim of this assignment is to help students understand: (i) how to write multi-threaded programs and consolidate the concepts learned in class; and (ii) the main concepts of multi-threaded programming with synchronization and thread scheduling. Students will obtain hands-on experience in working with POSIX PThreads (or, Windows Threads or Java Threads) and with OpenMP API (or, any implicit threading API of your choice). Use a thread API of your choice, though I prefer you use POSIX Pthreads API.

Tasks:

1. Program-1. An interesting way of calculating π is to use a technique known as *Monte Carlo*, which involves randomization. This technique works as follows. Suppose you have a circle inscribed within a square, as shown in the figure (assume that the radius of this circle is 1; thus we have a square of size 2x2).



First, generate s series of points as simple (*x*, *y*) coordinates. These points must fall within the Cartesian coordinates that bound the square. Of the total number of random points that are generated, some will occur within the circle. Next, estimate π by performing the following calculation:

$$\pi = 4 \times (\textit{number of points in circle}) / (\textit{total number of points})$$

Write a multi-threaded version of this algorithm that creates a separate thread (the *slave-thread*) to generate a number of random points. The slave-thread will count the number of points that occur within the circle (the `hit_count`) and store that result in the global variable `circle_count`. When the slave-thread has exited, the parent thread (the *master-thread*) will calculate and output the estimated value of π. It is worth experimenting with the number of random points generated. As a general rules, the greater the number of random points, the closer the approximation of π.

Below, are codes for generating random numbers, as well as codes for determining if the random (*x*, *y*) point occurs within the circle.

```
/* Generates a double precision random number */
double random_double()
{
     return random() / ((double)RAND_MAX + 1);
}

/* seed the random number generator */
     srandom((unsigned)time(NULL));

/*generate random numbers between -1.0 and +1.0 (exclusive)*/
/* to obtain a random (x, y) point*/
     x = random_double() * 2.0 - 1.0;
     y = random_double() * 2.0 - 1.0;

     /* is (x, y) point within the circle ? */
     if ( sqrt(x*x + y*y) < 1.0 )
          ++hit_count; /* yes, (x, y) point is in circle */
```

Note: Program-1 contains only 2 threads; a master-thread and its single slave-thread.

2. Program-2: Repeat Program-1 but instead of using a separate thread to generate random points, use OpenMP to parallelize the generation of points. Be careful not to place the calculation of π in the parallel region, since you want to calculate π only once.

3. Program-3. Program-1 asked you to design a multi-threaded program that estimated π using the *Monte Carlo* technique. In Program-1, you were asked to create a single slave-thread that generated random points, storing the results in a global variable `circle_count`. Once that slave-thread exited, the master-thread performed the calculation that estimated the value of π. Modify Program-1 so that you create several slave-threads, each of which generates random points and determines if the points fall within the circle. Each slave-thread will have to update the global count of all points that fall within the circle. Protect against race conditions on updates to the shared global variable by using mutex locks.

   Note: each slave-thread must generate `points_per_thread` random points, which is the ratio of the total number of random points to the number of slave-threads.

4. Program-4. Program-2 asked you to design a program using OpenMP that estimated π using the *Monte Carlo* technique. Examine your solution to Program-2 looking for any possible race conditions. If you identify a race condition, protect against it using the strategy outline in Section 5.10.2 of the textbook.

Let the constant `NUMBER_OF_POINTS` be the total number of random (*x, y*) points and the constant `NUMBER_OF_SLAVES` be the number of slave-threads.

1. Run Program-1 and Program-2 with `NUMBER_OF_POINTS = 100, 1000, 10000, 100000, 1000000`, and return for each run both their execution times and their estimated π values. Write a short report in which you compare the running times and the estimated π values, and provide meaningful comments (one or two paragraphs) on your results. The comments should address the running times obtained and relate them to the

concepts learned in class: (i) your multi-threaded Program-1 and Program-2 should run faster than a non-threaded program; (ii) Program-2 which uses OpenMP should run faster than Program-1; and (iii) a larger value of `NUMBER_OF_POINTS` should give a more accurate estimation of π than a smaller value.

2. Run Program-3 and Program-4 with `NUMBER_OF_POINTS = 1000000` and `NUMBER_OF_SLAVES = 2, 20, 40, 80, 100`, and return for each run both their execution times and their estimated π values. Write a short report in which you compare the running times and the estimated π values, and provide meaningful comments (one or two paragraphs) on your results. The comments should address the running times obtained and relate them to the concepts learned in class: (i) your multi-threaded Program-3 and Program-4 should run faster than Program-1 and Program-2; (ii) Program-4 which uses OpenMP should run faster than Program-3; (iii) a larger value of `NUMBER_OF_SLAVES` should give a faster running time than a smaller value; and (iv) what can you say about the estimated π values as `NUMBER_OF_SLAVES` changes from 2 to 100?

Submission:

1. You must submit: (i) a short report (in PDF or word), which contains the details of the implementation, the runs, the input/output of these runs, and the comments on the results of Program-1 to Program-4, (ii) the source code of your program. You should upload all these files to the Blackboard, as a **single** zip file.
2. In your report, you must provide details about the implementation, show how you run the two programs and the comments that address relationship between the running time and the concepts learned in class. Marks will be deducted if explanations/comments are missing.
3. Add the following note at the beginning of your report: "*I confirm that I will keep the content of this assignment confidential. I confirm that I have not received any unauthorized assistance in preparing for or writing this assignment. I acknowledge that a mark of 0 may be assigned for copied work.*" + Name + SID
4. Any submission after the deadline will receive a penalty of 10% for the first 24 hrs, and so on, for up to seven days. After seven days, the mark will be *zero*.
5. Unlimited resubmissions are allowed. But keep in mind that we will consider/mark the last submission. This means that if you resubmit after the deadline, a penalty will be applied, even if you submitted an earlier version before the deadline.