Lab 3: Threads

Sejong Yoon, Ph.D.

References:

- Silberschatz, et al. *Operating System Concepts* (10e), 2018
- Materials from OS courses offered at TCNJ (Dr. Jikai Li), Princeton, Rutgers, Columbia (Dr. Junfeng Yang), Stanford, MIT, UWisc, VT

Agenda

- Multi-threaded programming

Exercise 4.21

- Write a multithreaded program that calculates various statistical values for a list of numbers. This program will be passed a series of numbers on the command line and will then create three separate worker threads. One thread will determine the average of the numbers, the second will determine the maximum value, and the third will determine the minimum value. For example, suppose your program is passed the integers:

```
90 81 78 95 79 72 85
```

The program will report:

```
The average value is 82
The minimum value is 72
The maximum value is 95
```

Exercise 4.21 (cont.)

- The variables representing the average, minimum, and maximum values will be stored globally. The worker threads will set these values, and the parent thread will output the values once the workers have exited. A simpler solution (*with fixed length input*) for minimum case is provided in the next page for head-start.

- When compile, don't forget to put **-lpthread** to link Pthread library.

- Try to expand this program by creating additional threads that determine other statistical values, such as median and standard deviation

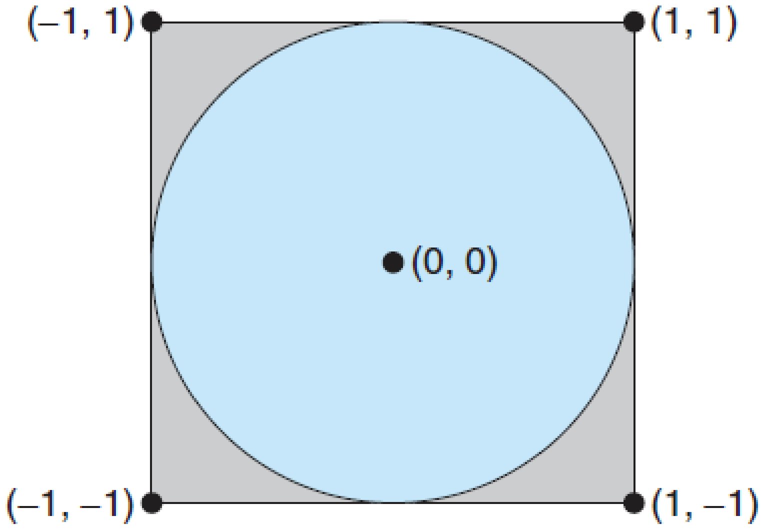
Exercise 4.21 (cont.)

```
// You need pthread.h, stdio.h, and stdlib.h
int len = 7;
int nums[7] = {90, 81, 78, 95, 79, 72, 85};
int r_min;
void *myMin (void *param)
{
        int i;

        r_min = nums[0];
        for ( i = 0; i < len; ++i ) {
                if ( r_min > nums[i] ) r_min = nums[i];
        }

        pthread_exit(0);
}
```

```
int main (int argc, char** argv)
{
        pthread_t tid;

        pthread_create ( &tid, 0, myMin, NULL );

        pthread_join ( tid, NULL );

        printf ( "Min = %d\n", r_min );

        return 0;
}
```

Exercise 4.22

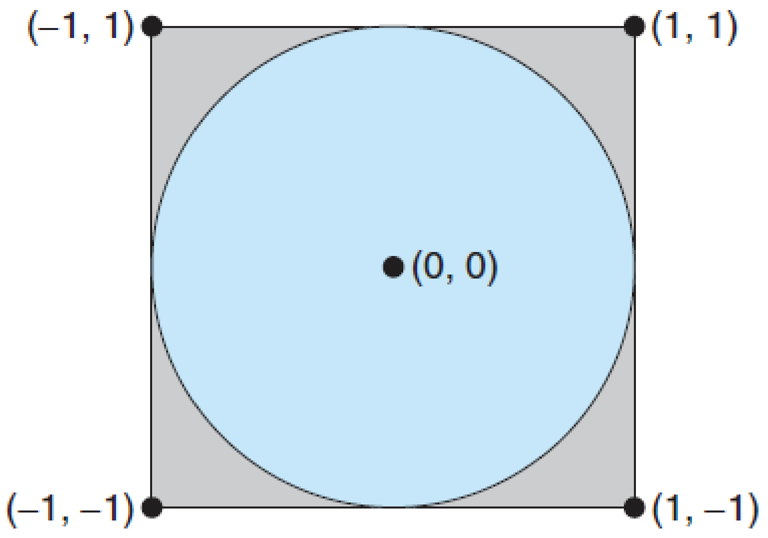
- An interesting way of calculating pi is to use a technique known as Monte Carlo, which involves randomization. This technique works as follows: Suppose you have a circle inscribed within a square, as shown in figure below: (Assume that the radius of this circle is 1.)



Exercise 4.22 (cont.)

- First, generate a series of random 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 pi by performing the following calculation:

  pi = 4× (number of points in circle) / (total number of points)



Exercise 4.22 (cont.)

- Write a multi-threaded version of this algorithm that creates a separate thread to generate a number of random points. The thread will count the number of points that occur within the circle and store that result in a global variable. When this thread has exited, the parent thread will calculate and output the estimated value of pi.
- It is worth experimenting with the number of random points generated. As a general rule, the greater the number of points, the closer the approximation to pi (about 3.14159265359).
- Readers interested in the details of the Monte Carlo method for estimating pi should consult the bibliography at the end of chapter 4.
- Next week, we modify this exercise using relevant material from chapter 5.

Lab 3 assignment

- Add a comment to the beginning of your source code containing your name, the name of the course, and the title of the assignment:

```
/** John Smith
CSC345-01
Lab 3 Exercise 1 */
```

- Rename your source file into lab03_ex1.c, lab03_ex2.c
- Prepare Makefile that compiles your source codes into object code lab03_ex1, lab03_ex2
- Zip your source file into lab03.zip
- Submit your zip file via Canvas