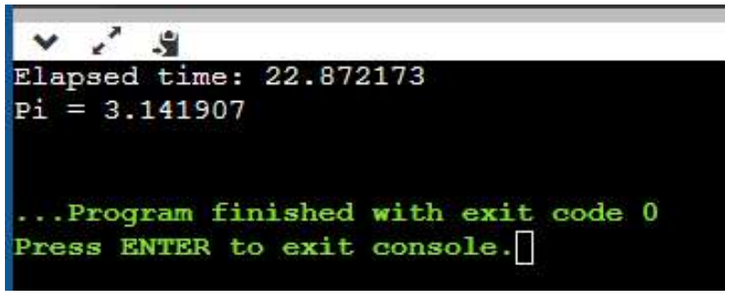
2) PI calculation with threads:

Description:

I'm using 4 threads, since that seems to be the best number in terms of finding the value of pi in the lowest amount of time.
The total darts are divided among the 4 threads, and each thread is executed at the same time.
I'm using mutex to protect the global variable circle_count.
Using mutex is necessary since multiple threads are working at the same time, so we want to avoid 2 threads making updates to the same variable, which may cause race conditions and in turn, corrupt our variable data.
Running the code without mutex can cause corruption in the end value of circle_count, because if 1 thread is updating the value of circle_count and is interrupted and other thread comes in and updates the value, the previous thread will have an incorrect value of circle_count and update incorrectly.
Functions used are:

1. pthread_mutex_init(&mutex, NULL);
    - This is used to initialize the mutex to null.
2. pthread_mutex_lock(&mutex);
    - This is used to lock the mutex.
3. pthread_mutex_unlock(&mutex);
    - This is used to unlock the mutex.
4. pthread_mutex_destroy(&mutex);
    - This is used to destroy the mutex, and free up its space.

Screenshot:



Code:

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include <unistd.h>
#include <time.h>
void *worker(void *param);

#define NUMBER_OF_DARTS 50000000
#define NUM_THREADS 4

// to complete: variables
int circle_count = 0;
double estimated_pi;
int dart_per_thread = NUMBER_OF_DARTS/NUM_THREADS;
pthread_mutex_t mutex;

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

int main (int argc, const char * argv[])
{
      pthread_mutex_init(&mutex, NULL);

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

      // to complete: Threads Creations
      pthread_t tid[NUM_THREADS];

      clock_t start = clock();

      int i = 0;
      for(i=0; i< NUM_THREADS; i++)
            pthread_create(&tid[i], NULL, worker, &dart_per_thread);
```

```
      for(i=0; i< NUM_THREADS; i++)
            pthread_join (tid[i], NULL);

    clock_t end = clock();
    double elapsed_time = (double)(end - start) / CLOCKS_PER_SEC;
    printf("Elapsed time: %f\n", elapsed_time);

      /* estimate Pi */
      estimated_pi = 4.0 * circle_count / NUMBER_OF_DARTS;

      printf("Pi = %f\n",estimated_pi);

      pthread_mutex_destroy(&mutex);
      return 0;
}

void *worker(void *param)
{
      int number_of_darts;
      number_of_darts = *((int *)param);
      int i;
      int hit_count = 0;
      double x,y;

      for (i = 0; i < number_of_darts; i++) {

            /* generate random numbers between -1.0 and +1.0 (exclusive) */
            x = random_double() * 2.0 - 1.0;
            y = random_double() * 2.0 - 1.0;

            if ( sqrt(x*x + y*y) < 1.0 )
                  ++hit_count;
      }

      pthread_mutex_lock(&mutex);
      circle_count += hit_count;
      pthread_mutex_unlock(&mutex);


      pthread_exit(0);
}
```