CS 4540 – Operating Systems

Instructor: Lilien, Western Michigan University

Justin Lanyon

**SOFTWARE LIFE CYCLE REPORT – FOR ASSIGNMENT 05**

**PHASE 1: SPECIFICATION (“What do we build?”)**

Write a multithreaded version of the above algorithm that creates ten threads, each of which: (i) generates a predefined number of random points, (ii) determines if the points fall within the circle, and (iii) counts the number of points that fall within the circle.

Each thread will have to update the shared global count circleCount of all points that fall within the circle. Protect against race conditions on updates to circleCount by using mutex locks.

When all thread have exited, the parent thread will calculate and output the estimated value of π.

**PHASE 2: DESIGN**

**2.1 Modules and Their Basic Structure**

Main: creates all 10 threads and joins them. Then it calculates the estimate for pi based on how many points where in the circle and how many points overall.

threadFunction: generates random numbers based on hard-coded value and increments the circleCount when one is within the circle.

**Phase 3: RISK ANALYSIS**

There are no risks (to cost, human health, timetable) are identifiable.

**PHASE 4: VERIFICATION**

This algorithm has a very standard path that has been verified for its correctness by me through analyzing its steps in comparison to the specifications.

**PHASE 5: CODING**

**5a) Code Refinement #1(class structure with pseudocode only;)**

/\* INCLUDES\*/

/\*THREAD FUNCTION\*/

/\*MAIN METHOD\*/

**5b) Code Refinement #2**

/\* INCLUDES\*/

/\*GLOBAL VARIABLES\*/

/\*THREAD FUNCTION\*/

/\*Random x y coordinates and increment circleCount\*/

/\*MAIN METHOD\*/

/\*CREATE THREADS\*/

**5c) Code Refinement #3**

/\* INCLUDES\*/

/\*GLOBAL VARIABLES\*/

/\*THREAD FUNCTION\*/

/\*Loop till numPoints\*/

/\*RANDOM X Y coordinate\*/

/\*EXTRA CREDIT - CALCULATE THREAD ESTIMATE\*/

/\*MAIN METHOD\*/

/\*Create mutex lock\*/

/\*Loop to create all threads\*/

/\*Loop to join all threads\*/

/\*Destroy Mutex\*/

/\*Calculate PI estimate\*/

/\*Print Pi estimate\*/

**5d) Code Refinement #4 (complete program--with complete fields/properties, code for constructor/methods)**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Course: CS 4540 – Fall 2015

\* Assignment 5

\* Name: Justin Lanyon

\* E-mail: justin.j.lanyon@wmich.edu

\* Submitted: 12/11/2015

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* INCLUDES\*/

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <math.h>

/\*GLOBAL VARIABLES\*/

int NR\_THREADS = 10;

pthread\_mutex\_t count\_mutex;

double circleCount=0;

int NR\_PTS = 1000000;

/\*Thread Function\*/

void \* threadFunction(){

double x =0, y=0, threadPi =0, inCircle =0;

int counter =0;

pthread\_t thrID = pthread\_self();

unsigned seed = getpid() \* time(NULL) \* (thrID + 1);

/\*Loop till numPoints\*/

while(counter<(NR\_PTS/NR\_THREADS)){

/\*RANDOM X Y coordinate\*/

x =(rand\_r(&seed)/(RAND\_MAX +2.0));

y =(rand\_r(&seed)/(RAND\_MAX +2.0));

if((pow(x,2.0))+(pow(y,2.0)) <=1){

inCircle++;

pthread\_mutex\_lock(&count\_mutex);

circleCount++;

pthread\_mutex\_unlock(&count\_mutex);

}/\*END IF\*/

counter++;

}/\*END WHILE LOOP\*/

/\*EXTRA CREDIT - CALCULATE THREAD ESTIMATE\*/

threadPi = 4 \*inCircle/(NR\_PTS/NR\_THREADS);

printf("Thread estimate for PI is : %f\n",threadPi);

}/\*END THREAD FUNCTION\*/

int main(){

int i =0;

double piEst=0;

int err;

pthread\_t tid[NR\_THREADS];

/\*Create mutex lock\*/

if (pthread\_mutex\_init(&count\_mutex, NULL) != 0)

{

printf("\n mutex init failed\n");

return 1;

}

/\*Loop to create all threads\*/

while(i<NR\_THREADS){

err = pthread\_create(&(tid[i]),NULL,&threadFunction,NULL);

if(err !=0){

printf("\ncan't create thread : %d\n",err);

}

i++;

}

i =0;

/\*Loop to join all threads\*/

while(i<NR\_THREADS){

pthread\_join(tid[i], NULL);

i++;

}

/\*Destroy Mutex\*/

pthread\_mutex\_destroy(&count\_mutex);

/\*Calculate PI estimate\*/

piEst = 4 \*circleCount / (NR\_PTS);

/\*Print Pi estimate\*/

printf("The estimate for PI after combining all threads is: %f\n", piEst);

printf("%d points were used in this estimation\n",NR\_PTS);

return 0;

}/\*END MAIN\*/

**PHASE 6: TESTING**

One can test it by changing the numPoints to make the estimates more or less accurate by increasing or decreasing respectively.

**PHASE 7: REFINING THE PROGRAM**

No refinements are needed. In this program, I have already included all required features!

**PHASE 8: PRODUCTION**

I prepared a copy of the entire program for Lab TA’s evaluation, as specified by the TA.! Then, I sent

electronically the copy to the Lab TA.

**PHASE 9: MAINTENANCE**

To fully benefit from the program evaluation feedback received from the Lab TA, I will perform program

maintenance. This means that I should use all TAs feedback to improve my program.