Program1 Name: Yu Yang Student ID: 892449550

import java.util.Random;

import java.io.BufferedWriter;

import java.io.BufferedReader;

import java.io.FileReader;

import java.io.FileWriter;

import java.io.FileNotFoundException;

import java.io.IOException;

public class ThreadedMonteCarlo implements Runnable{

private static String SOURCE\_FILE;

private int total = 0;

private int inCircle = 0;

private int num;

BufferedReader reader;

public ThreadedMonteCarlo(String file){

try {

SOURCE\_FILE = file;

reader = new BufferedReader(new FileReader(file));

total = 0;

inCircle = 0;

}catch(FileNotFoundException ex){

ex.printStackTrace();

}

}

private class point{

float x;

float y;

public void setCood(String[] cood){

x = Float.valueOf(cood[0]);

y = Float.valueOf(cood[1]);

}

}

private void creatFile(int num) {

this.num = num;

try {

BufferedWriter os = new BufferedWriter(new FileWriter(SOURCE\_FILE));

os.write(num+"\n");//total number of points

Random r = new Random();

while(num-->0){

float x = -1 + r.nextFloat()\*2;

float y = -1 + r.nextFloat()\*2;

os.write(x+","+y+"\n");

os.flush();

}

os.close();

}catch(IOException ex){

ex.printStackTrace();

}

}

private synchronized void incIcircle(){

++inCircle;

}

private synchronized void incTotal(){

++total;

}

private int getTotal(){

return total;

}

//read points info

private boolean readOneline(BufferedReader rder,point m){

String line = null;

try{

if((line = rder.readLine()) != null) {

m.setCood(line.trim().split(",\\s\*"));

}else{

return false;

}

}catch(IOException ex) {

ex.printStackTrace();

}

incTotal();

return true;

}

private boolean isInCircle(point m){

return (Math.pow(m.x,2)+Math.pow(m.y,2) <= 1);

}

private synchronized boolean proOneLine(point p){

return readOneline(this.reader, p);

}

public void run(){

point p = new point();

while(proOneLine(p)) {

if(isInCircle(p)){

incIcircle();

}

}

//System.out.println(Thread.currentThread().getName()+ " finished"+ " total num is "+ this.getTotal()+" inC is " + this.inCircle);

}

public void passFirstLine(){

String line = null;

try {

if((line = this.reader.readLine()) != null) {

this.num = Integer.parseInt(line);

}

}catch(IOException ex){

ex.printStackTrace();

}

}

public static void main(String [] args){

//constructing object and read fileName

ThreadedMonteCarlo myT = new ThreadedMonteCarlo(args[0]);

//read number of points

myT.passFirstLine();

//create threads

if(Integer.valueOf(args[1])<=0){

System.err.println("invalid process number");

return;

}

Thread th[] = new Thread[Integer.valueOf(args[1])];

for(int i = 0; i < th.length;i++) {

th[i] = new Thread(myT);

//set name, only used for debugging

th[i].setName("th"+i);

}

try {

for(Thread temp:th) {

temp.start();

}

for (Thread temp : th) {

temp.join();

}

}catch(InterruptedException ex){

System.err.println("Interupted!");

}

//check if number of points matches

if(myT.getTotal() != myT.num){

System.err.println("number of points doesn't match");

return;

}

//display result

System.out.print("Calc finished! Total num of points is " + myT.getTotal() + " .");

System.out.println("Num of in-circle points is " + myT.inCircle);

System.out.println("Estimated PI value is " + (4\* (float)myT.inCircle/(float)myT.num));

}

}

//MonteCarlo.c

#include <stdio.h>

#include <unistd.h>

#include <sys/types.h>

#include <string.h>

#include <sys/wait.h>

#include <stdlib.h>

#define READ\_END 0

#define WRITE\_END 1

#define TRUE 1

#define FALSE 0

typedef struct point{

float x;

float y;

} point;

float sq(float x){

return x\*x;

}

void calcInCirPoints(int i,struct point \*pp, int \*cnt) {

for (; --i >= 0;) {

if (sq(pp[i].x) + sq(pp[i].y) <= 1.0)

(\*cnt)++;

}

}

void createMultiPro(int n,int \*\*fd, int \*\*fdc){

pid\_t id;

while(--n>0){

id = fork();

if(id<0){

fprintf(stderr, "Fork failed");

exit(1);

}

if(id>0){

\*fd = ((\*fd) + 4);

\*fdc = ((\*fdc) + 2);

}else if(id == 0){

break;

}

}

}

int createPipe(int numOfProcess,int \*fd, int \*fdC){

while(numOfProcess-->0){

if(pipe(fd) == -1 || pipe(fd+2)==-1 || pipe(fdC) == -1){

return FALSE;

}

fd+=4;

fdC+=2;

}

return TRUE;

}

void parentCloseReadPipeEnd(int \*fd,int numOfProcess,int \*fdc){

while(numOfProcess -- >0) {

close(\*(fd + READ\_END));

close(\*(fd + READ\_END + 2));

close(\*(fdc + WRITE\_END));

fd += 4;

fdc +=2;

}

}

void childCloseWriteEndPipe(int \*fd, int \*fdc) {

close(\*(fd + WRITE\_END));

close(\*(fd + WRITE\_END + 2));

close(\*(fdc + READ\_END));

}

void chldCloseReadEndPipe(int \*fd, int \*fdc) {

close(\*(fd + READ\_END));

close(\*(fd + READ\_END + 2));

close(\*(fdc + WRITE\_END));

}

void parentWritePipe(int numOfProcess,int pointsPerProcess, int pointsMod,int \*fd, struct point \*a,int \*fdc,int \*inCircleTotal) {

int i = 0;

while (i < numOfProcess) {

int tempInC;

if (i == (numOfProcess-1)) {

write(\*(fd + WRITE\_END), &a[i \* pointsPerProcess], sizeof(struct point) \* (pointsMod));

write(\*(fd + WRITE\_END + 2), &pointsMod, sizeof(int));

} else {

write(\*(fd + WRITE\_END), &a[i \* pointsPerProcess], sizeof(struct point) \* pointsPerProcess);

write(\*(fd + WRITE\_END + 2), &pointsPerProcess, sizeof(int));

}

read(\*(fdc+READ\_END),&tempInC,sizeof(int));

\*inCircleTotal +=tempInC;

close(\*(fd + WRITE\_END));

close(\*(fd + WRITE\_END + 2));

close(\*(fdc +READ\_END));

fd += 4;

fdc +=2;

i++;

}

}

float randFloat(){

return -1 + 2\*((float)rand()/RAND\_MAX);

}

void createFile(FILE \*fp,char\* name,int numOfPoints){

fp = fopen(name,"w");

if(fp == NULL) {

perror("fopen");

exit(1);

}

fprintf(fp,"%d\n",numOfPoints);

while(numOfPoints-->0){

float x = randFloat();

float y = randFloat();

fprintf(fp,"%f,%f\n",x,y);

}

if(fclose(fp)!=0){

perror("fclose");

exit(1);

}

}

int readFile(FILE \*fp,char \*name,struct point \*\*a){

char \*line = NULL;

char seperators[] = ", ";

fp = fopen(name,"r");

size\_t len = 0;

ssize\_t sizeR;

int i=0;

int num;

if(fp == NULL) {

perror("fopen");

exit(1);

}

if((sizeR=getline(&line,&len,fp)!=-1)) {

num = (int)atof(strtok(line,"\n"));

}

\*a = malloc(num\* sizeof(struct point));

while((sizeR=getline(&line,&len,fp)!=-1)){

if(i>num){

break;

}

(\*a)[i].x = atof(strtok(line,seperators));

(\*a)[i].y = atof(strtok(NULL,seperators)); i++;

}

if(i != num){

perror("num of points doesn't match\n");

fclose(fp);

free(\*a);

exit(1);

}

free(line);

if(fclose(fp)!=0){

perror("fclose");

exit(1);

}

return num;

}

int main(int argn,char \*arg[])

{

if(argn == 3){

printf("file name is %s\n",arg[1]);

printf("number of processes is %d\n",(int)atof(arg[2]));

}else{

perror("arg number doesn't match, exit");

exit(1);

}

char \*fn = arg[1];

int numOfProcess = (int)atof(arg[2]);

FILE \*fp;

struct point \*a1;

int numOfPoint=readFile(fp,fn,&a1);

int numInCircle=0;

//split points evenly

int pointsPerProcess = numOfPoint/numOfProcess;

int pointsMod = numOfPoint%numOfProcess+pointsPerProcess;

//adjusting num of processes if needed

if(numOfPoint<numOfProcess){

printf("num of points is too small when compared with num of processes,adjusting process number to point number\n");

numOfProcess = numOfPoint;

pointsPerProcess = pointsMod = 1;

}

pid\_t pid;

//each child process has 3 pipes

//1 for pass points

//1 for pass num of points to be processed

//1 for pass back num of in-circle points to parent

int fda[numOfProcess][2][2];

int fdCircle[numOfProcess][2];

int \*fd = &fda[0][0][0];

int \*fdC=&fdCircle[0][0];

/\* create the pipe \*/

if (!createPipe(numOfProcess,fd,fdC)) {

fprintf(stderr,"Pipe failed");

exit(1);

}

/\* now fork several child process \*/

if (pid=fork(), pid < 0) {

fprintf(stderr, "Fork failed");

exit(1);

}

if (pid > 0) { /\* parent process \*/

//parent close unused pipes

parentCloseReadPipeEnd(fd,numOfProcess,fdC);

//papent using pipes to pass info to children

//and get processed info from children

parentWritePipe(numOfProcess,pointsPerProcess,pointsMod,fd,a1,fdC,&numInCircle);

wait(NULL);

printf("number of sample points is %d, number of in-circle points is %d\nEstimated PI value is %f\n",numOfPoint,numInCircle,4\*(float)numInCircle/(float)numOfPoint);

}

else { /\* child process \*/

//create multiple child process

createMultiPro(numOfProcess,&fd,&fdC);

/\* close the unused end of the pipe \*/

childCloseWriteEndPipe(fd,fdC);

int i;//index

//use one pipe to get the num of points to be processed

read(\*(fd+READ\_END+2),&i,sizeof(int));

struct point \*pp;

pp = malloc(i \* sizeof(struct point));

//use another pipe to get points

read(\*(fd+READ\_END), pp, i\*sizeof(struct point));

int cnt=0;//points in circle

//calc in-circle points and pass back

calcInCirPoints(i,pp,&cnt);

write(\*(fdC+WRITE\_END),&cnt,sizeof(int));

//close pipes

chldCloseReadEndPipe(fd,fdC);

free(pp);

wait(NULL);

}

free(a1);

}