Московский Авиационный Институт
(Национальный Исследовательский Университет)
Факультет информационных технологий и прикладной математики
Кафедра вычислительной математики и программирования

Лабораторная работа №3 по курсу
«Операционные системы»

Студент: Ползикова Алина Владимировна
Группа: М8О-208Б-21
Вариант: 16
Преподаватель: Миронов Евгений Сергеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2022

Содержание

Репозиторий

https://github.com/polzzzik/lab_os

Постановка задачи

Цель работы

Изучить управление потоками, обеспечение синхронизации между потоками.

Задание

Вариант 16) Задаётся радиус окружности. Необходимо с помощью метода Монте-Карло рассчитать её площадь

Общие сведения о программе

Программа компилируется из файла main.c. Также используется заголовочные файлы: unistd.h, pthread.h, stdio.h, math.h. В программе используются следующие системные вызовы:

1. pthread_create() - создаёт новый поток, принимает параметры.
2. pthread_join() - ждёт завершения работы потока.

Общий метод и алгоритм решения

Разделим область нашего интегрирования и количество точек между процессами, разделение областей круга будет происходить по оси X, на K, где K – количество потоков, равных частей. (то есть координата x случайных точек будет в пределах выделенной области для каждого потока, а координата У у всех генерироваться в одном и том же диапазоне). Так мы разделяем всю работу на K потоков. Заметим, что такого феномена, как «Race condition» у нас не возникнет, так как каждый поток будет считать только свои точки, а после завершения всех процессов мы сложим полученные ответы.

Исходный код

========== main.c ==========

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <pthread.h>
```

```
#include <unistd.h>

typedef struct TThreadToken {
   double* R;
   double* step;
   double start;
   int Cpoints;
} ThreadToken;

void exit_with_msg(const char* msg, int return_code);
void* integral(void* arg);
int min(int a, int b);

int main(int argc, const char** argv) {
   int Total_points = 10000000;
   double R, Total_Aria_Size;
   int CountThreads;
   if (argc < 2) {
      exit_with_msg("missing arguments", -1);
   }
   for (int i = 0; argv[1][i] > 0; ++i) {
      if (argv[1][i] >= '0' && argv[1][i] <= '9') {
         CountThreads = CountThreads * 10 + argv[1][i] - '0';
      }
   }
   printf("enter Radius: ");
   scanf("%lf", &R);
   Total_Aria_Size = R * 2;
   pthread_t* th = malloc(sizeof (pthread_t) * CountThreads);
   ThreadToken* token = malloc(sizeof(ThreadToken) * CountThreads);
   double start = -R;
   double step = (Total_Aria_Size / (double ) CountThreads);
   int Cpoints = (Total_points + CountThreads - 1)/CountThreads;
   for (int i = 0; i < CountThreads; ++i) {
      token[i].start = start;
      token[i].step = &step;
      token[i].R = &R;
      token[i].Cpoints = min(Cpoints, Total_points - i*Cpoints);
```

```
        start += step;
    }

    for (int i = 0; i < CountThreads; ++i) {
        if (pthread_create(&th[i], NULL, &integral, &token[i]) != 0) {
            exit_with_msg("cannot create thread", 2);
        }
    }
    Cpoints = 0;
    for (int i = 0; i < CountThreads; ++i) {
        if (pthread_join(th[i], NULL) != 0) {
            exit_with_msg("cannot join threads", 3);
        }
        Cpoints += token[i].Cpoints;
    }
    printf("Exact answer is        : %.20lf\n", acos(-1)*R*R);
    printf("Answer is approximately: %.20lf\n",
          Total_Aria_Size*Total_Aria_Size*((double) Cpoints / (Total_points)));
    free(token);
    free(th);
    return 0;
}

int min(int a, int b) {
    if (a < b) return a;
    return b;
}

void exit_with_msg(const char* msg, int return_code) {
    printf("%s\n", msg);
    exit(return_code);
}

int in_circle(double x, double y, double R) {
    return (x*x + y*y <= R*R);
}

void* integral(void* arg) {
```

```
    ThreadToken token = *((ThreadToken*) arg);
    double x, y, R;
    R = *token.R;
    int attempts = token.Cpoints;
    token.Cpoints = 0;
    for (int i = 0; i < attempts; ++i) {
        x = token.start + ((double )rand()/(double )(RAND_MAX)) * (*token.step);
        y = (((double )rand()/(double )(RAND_MAX)) - 0.5) * 2*R;
        if (in_circle(x,y, R)) {
            token.Cpoints++;
        }
    }
    ((ThreadToken*) arg)->Cpoints = token.Cpoints;
    return arg;
}
```

Демонстрация работы программы

```
hplp739@user:~/Desktop/OS/lab_3$ time ./a.out 1
enter Radius: 1
Exact answer is        : 3.14159265358979311600
Answer is approximately: 3.14225600000000016010

real    0m1,150s
user    0m0,580s
sys     0m0,000s
hplp739@user:~/Desktop/OS/lab_3$ time ./a.out 2
enter Radius: 1
Exact answer is        : 3.14159265358979311600
Answer is approximately: 3.14250279999999992953

real    0m1,871s
user    0m1,501s
sys     0m1,278s
hplp739@user:~/Desktop/OS/lab_3$ time ./a.out 3
enter Radius: 1
Exact answer is        : 3.14159265358979311600
Answer is approximately: 3.14203320000000019263

real    0m3,104s
user    0m3,205s
sys     0m3,874s
```

```
hplp739@user:~/Desktop/OS/lab_3$ time ./a.out 5
enter Radius: 1
Exact answer is        : 3.14159265358979311600
Answer is approximately: 3.14103120000000002321

real    0m4,951s
user    0m5,317s
sys     0m10,806s
hplp739@user:~/Desktop/OS/lab_3$ time ./a.out 10
enter Radius: 1
Exact answer is        : 3.14159265358979311600
Answer is approximately: 3.14187040000000017415

real    0m3,850s
user    0m3,733s
sys     0m10,395s
hplp739@user:~/Desktop/OS/lab_3$ time ./a.out 50
enter Radius: 1
Exact answer is        : 3.14159265358979311600
Answer is approximately: 3.14175680000000001613

real    0m2,924s
user    0m2,724s
sys     0m7,714s
hplp739@user:~/Desktop/OS/lab_3$ time ./a.out 100
enter Radius: 1
Exact answer is        : 3.14159265358979311600
Answer is approximately: 3.14182039999999984659

real    0m3,261s
user    0m3,259s
sys     0m8,914s
```

Выводы

В результате данной лабораторной работы, научилась работать с потоками (Threads), составила и отладила программу вычисления площади круга методом Монте-Карла. Во время выполнения работы я поняла, что создать и синхронизировать много потоков может быть более накладно, чем выполнять код на одном ядре. . Из-за этого при создании двух и более потоков программа сильно замедляется (по сравнению с одним потоком). Если задачу в программировании можно выполнять по частям, независимо друг от друга, то многопоточный подход ускорит работу программы (если синхронизации не обойдётся дороже!) в несколько раз. Так в играх вычисления (например, перемножение матриц) производятся на большом количестве графических ядер параллельно.

Как видно из демонстрации работы программы, самый быстрый способ выполнить поставленную задачу – это рассчитать всё одним потоком без использования распараллеливания. Это связано с издержками, которые мы несём при создании и выполнении потоков. Однако дальше видна тенденция к ускорения работы, при увеличении количества потоков, это уже говорит о том, что метод параллельных вычислений действительно даёт выигрыш в скорости, хотя и конкретно в данной задаче не покрывает расходов на создание потоков.