Московский Авиационный Институт

(Национальный Исследовательский Университет)

Институт №8 “Компьютерные науки и прикладная математика”

Кафедра №806 “Вычислительная математика и программирование”

Лабораторная работа №2 по курсу

«Операционные системы»

Группа: М8О-211БВ-24

Студент: Рыбин В.В.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 04.10.25

Москва, 2025

Постановка задачи

Вариант 16.

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

Общий метод и алгоритм решения

Использованные системные вызовы:

- int pthread_create(pthread_t *thread, const pthread_attr_t *attr, void *(*start_routine)(void*), void *arg); – Создает поток с заданными атрибутами, который начинает выполнение функции start_routine
- int pthread_join(pthread_t thread, void **retval); – ожидает завершения указанного потока.
- int pthread_mutex_lock(pthread_mutex_t *mutex); – блокирует мьютекс. Предотвращает состояние гонки при одновременном доступе из нескольких потоков
- int pthread_mutex_unlock(pthread_mutex_t *mutex); – разблокирует мьютекс.\
- int clock_gettime(clockid_t clk_id, struct timespec *tp); – получает текущее монотоное время системы

  struct timespec {
  time_t tv_sec; - секунды
  long tv_nsec; - наносекунды
  };

Я реализовал программу, которая использует многопоточность для вычисления площади окружности с использованием метода Монте-Карло. Сначала нужно вписать окружность в квадрат длиной ее радиуса. Далее для этого метода необходимо брать случайные точки и проверять, попали ли они в квадрат или в окружность. Затем площадь окружности вычисляется как (колво точек в окружности/колво точек всего) * площадь квадрата.

Я зафиксировал количество точек в main.c. Количество потоков подается как ключ к моей программе. Затем я отдаю на каждый поток какое то колво точек, чтобы они посчитали, что куда попало. Я делю это равно, то есть общее колво точек/колво потоков.

Также я использую mutex, чтобы предотвратить гонку.

Код программы

main.c

```c
#include <stdint.h>
#include <stdbool.h>

#include <stdlib.h>
#include <stdio.h>
#include <string.h>

#include <unistd.h>
#include <pthread.h>
```

```c
#include <errno.h>

#include <time.h>

typedef struct {
    size_t number;
    double R;
    int count_iterations;
    unsigned int seed;
} ThreadArgs;

static volatile int32_t count_in_circle = 0;
static volatile int32_t count_total = 0;

static pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;

static void *work(void *_args) {
    ThreadArgs *args = (ThreadArgs *)_args;
    int n = args->count_iterations;
    double x, y;
    int local_count_in_circle = 0;

    unsigned int seed = args->seed;

    for (size_t i = 0; i < n; ++i) {
        x = (double)rand_r(&seed) / RAND_MAX * args->R;
        y = (double)rand_r(&seed) / RAND_MAX * args->R;


        if (x * x + y * y <= args->R * args->R) {
            ++local_count_in_circle;
        }
    }

    pthread_mutex_lock(&mutex);
    count_total += n;
    count_in_circle += local_count_in_circle;
    pthread_mutex_unlock(&mutex);

    // printf("Threads #%ld says count_in_circle: %d, count_total: %d\n", args-
>number, count_in_circle, count_total);

    return NULL;
}

void print_usage(const char* program_name) {
    printf("usage: %s --threads <number>\n", program_name);
    printf("       %s -t <number>\n", program_name);
}

int validate_flags(size_t *n_threads, int argc, char* argv[]) {
    if (argc != 3) {
        print_usage(argv[0]);
        return 1;
```

```
    }

    if (strcmp(argv[1], "--threads") == 0 || strcmp(argv[1], "-t") == 0) {
        char* endptr;
        errno = 0;
        *n_threads = strtoll(argv[2], &endptr, 10);

        if (errno == ERANGE) {
            printf("Произошло переполнение\n");
            print_usage(argv[0]);
            return 1;
        }
        else if (endptr == argv[2] || *endptr != '\0') {
            printf("Введите число\n");
            print_usage(argv[0]);
            return 1;
        }
    }
    else {
        print_usage(argv[0]);
        return 1;
    }
}


int main(int argc, char* argv[]) {
    int count = 10000000;

    size_t n_threads = 0;
    int flag = validate_flags(&n_threads, argc, argv);
    if (flag) {
        return 1;
    }

    // --n_threads; // один поток выполняет main

    double R;
    printf("Введите радиус окружности: ");
    scanf("%lf", &R);


    pthread_t *threads = malloc(n_threads * sizeof(pthread_t));
    ThreadArgs *thread_args = malloc(n_threads * sizeof(ThreadArgs));

    clock_t start = clock();


    for (size_t i = 0; i < n_threads; ++i) {
        thread_args[i] = (ThreadArgs) {
            .number = i,
            .R = R,
            .count_iterations = count / n_threads,
            .seed = rand(),
        };
```

```
        pthread_create(&threads[i], NULL, work, &thread_args[i]);
    }


    for (size_t i = 0; i < n_threads; ++i) {
        pthread_join(threads[i], NULL);
    }


    clock_t end = clock();

    double time = (double)(end - start) / CLOCKS_PER_SEC;

    double square = 4 * (count_in_circle/(double)count_total) * R * R;
    printf("***\nSquare circle: %lf\n", square);
    printf("Working time: %lfs\n", time);
    printf("Count threads: %ld\n", n_threads);
    printf("***\n");

    free(thread_args);
    free(threads);

    return 0;
}
```

Протокол работы программы

❯ ./main.out -t 1

Введите радиус окружности: 1

****

Square circle: 3.141585

Working time: 13.408602s

Count threads: 1

❯ ./main.out -t 2

Введите радиус окружности: 1

****

Square circle: 3.141581

Working time: 6.739873s

Count threads: 2

❯ ./main.out -t 8

Введите радиус окружности: 1

****

Square circle: 3.141580

Working time: 1.948613s

Count threads: 8

❯ ./main.out -t 12

Введите радиус окружности: 1

****

Square circle: 3.141585

Working time: 1.587313s

Count threads: 12

❯ ./main.out -t 16

Введите радиус окружности: 1

****

Square circle: 3.141585

Working time: 1.495700s

Count threads: 16

❯ ./main.out -t 1024

Введите радиус окружности: 1

****

Square circle: 3.141578

Working time: 1.612172s

Count threads: 1024

❯ ./main.out -t 8096

Введите радиус окружности: 1

****

Square circle: 3.141578

Working time: 1.807809s

Count threads: 8096

❯ ./main.out -t 16192

Введите радиус окружности: 1

***

Square circle: 3.141559

Working time: 2.102531s

Count threads: 16192

***

Вывод

В ходе данной лабораторной работы я научился работать с потоками. Разобрался с различными проблемами и нюансами при работе с ними (гонка). Также получил следующие выводы, которые я изображу в виде таблицы

Число потоков	Время исполнения (мс)	Ускорение	Эффективность
1	13408.602	1.00	1.00
2	6739.873	1.99	0.99
8	1948.613	6.88	0.86
12	1587.313	8.44	0.71
16	1495.700	8.96	0.56
1024	1612.172	8.31	0.008
8096	1807.809	7.42	0.0009
16192	2102.531	6.38	0.0004

Расчеты:

- **Ускорение: T1/Tn, где T1 – время выполнения с 1 потоком,**
  Tn – время выполнения с n потоками
- **Эффективность: Ускорение/n**

Анализ результатов:

1. Количество потоков МЕНЬШЕ логических ядер процессора (1-8 потоков)

- Наблюдается почти линейное масштабирование (1.99× при 2 потоках)
- Эффективность остается высокой (0.86-0.99)
- Накладные расходы минимальны, потоки работают практически без конкуренции
- Мьютекс не является узким местом из-за малой частоты обращений
- Идеальный случай для CPU-bound задач

Вывод: В этом диапазоне многопоточность дает максимальную отдачу с минимальными накладными расходами.

2. Количество потоков РАВНО логическим ядрам процессора (16 потоков)

- Эффективность резко падает до 0.56
- Начинают проявляться эффекты:
  - Конкуренция за мьютекс становится заметной
  - Накладные расходы на переключение контекста

Вывод: Достигается практический предел эффективного использования CPU, дальнейшее увеличение потоков дает ухудшение результатов

3. Количество потоков БОЛЬШЕ логических ядер процессора (16+ потоков)

- **Катастрофическое падение эффективности** (0.009 при 1024 потоках)
- **Деградация производительности** при дальнейшем увеличении потоков

Критические проблемы:

- Огромные накладные расходы на создание/уничтожение потоков
- Конкуренция за глобальный мьютекс
- Потоки большую часть времени ожидают, а не вычисляют
- Перегрузка планировщика
  - При 16k потоков планировщик ОС должен постоянно принимать решения о том, какой поток выполнять следующим
  - Время на принятие решений становится сопоставимым с временем выполнения

Вывод: Количество потоков, значительно превышающее число логических ядер, приводит к деградации производительности из-за преобладания накладных расходов над полезной работой.