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-210Б-23

Студент: Юрченко А.Н.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 12.12.24

Москва, 2024

Постановка задачи

Вариант 16.

Целью является приобретение практических навыков в:

• Управление потоками в ОС

• Обеспечение синхронизации между потоками

Задание

Составить программу на языке Си, обрабатывающую данные в многопоточном режиме. При обработки использовать стандартные средства создания потоков операционной системы (Windows/Unix). Ограничение максимального количества потоков, работающих в один момент времени, должно быть задано ключом запуска вашей программы.

Так же необходимо уметь продемонстрировать количество потоков, используемое вашей программой с помощью стандартных средств операционной системы.

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

Общий метод и алгоритм решения

Использованные системные вызовы:

- int pthread_mutex_lock (pthread_mutex_t *mutex) - для перевода мьютекса из разблокированного в заблокированное состояние
- int pthread_mutex_unlock(pthread_mutex_t *mutex) - для перевода мьютекса из заблокированного в разблокированное состояние
- int pthread_mutex_init(pthread_mutex_t *restrict mutex, const pthread_mutexattr_t *restrict attr) - нужен для инициализации мьютекса
- int pthread_mutex_destroy(pthread_mutex_t *mutex) - нужен для уничножения мьютекса
- int pthread_create(*ptherad_t, const pthread_attr_t *attr, void* (*start_routine)(void*), void *arg) - создает новый поток
- int pthread_join(pthread_t thread, void **value_ptr) - откладывает выполнение вызывающего (эту функцию) потока, до тех пор, пока не будет выполнен поток thread
- ssize_t write(int fd, const void *buf, size_t count) - производит запись в описатель файла

Алгоритм работы программы:

1. Подаются на вход три аргумента: радиус окружности, количество потоков, количество точек, необходимое для работы метода Монте-Карло.
2. Находится число точек, которое будет обрабатывать каждый поток (количество точек, подаваемое 3 аргументом деленное на количество потоков) и создается переменная, которая будет хранить число точек, попавшее в заданную окружность (total_inside_circle).
3. Вызывается pthread_create, чтобы создать потоки, которые будут выполнять функцию monte_carlo.

4. В функции monte_carlo происходит случайным образом получение координаты точки и проверяется попала ли она в окружность, если попала, то локальный счетчик переменных, попавших в окружность для данного потока, увеличивается на 1. Делаем это пока не закончатся все точки с которыми нужно провести данный экперимент. Блокируем мьютекс, чтобы прибавить полученное значение локальным счетчиком к глобальному счетчику. Разблокируем мьютекс.
5. Вычисляем пи на основе полученного количества точек внутри круга и общего количества точек. И наконец рассчитываем площадь по формуле пи * r * r.

Построение графиков:

Пусть радиус равен 10, а число точек, подаваемых на вход, равно 1000000

Расчет ускорения: Ускорение можно рассчитать как отношение времени выполнения программы с одним потоком к времени выполнения программы с n потоками.

Расчет эффективности: Эффективность можно рассчитать как отношение ускорения к количеству потоков.

Число потоков	Время выполнения, сек	Ускорение	Эффективность
1	0.120227	1	1
2	0.057596	2.09	1.05
3	0.054611	2.20	0.73
4	0.053328	2.25	0.56
6	0.051822	2.32	0.39
8	0.048285	2.49	0.31
10	0.044956	2.67	0.27
15	0.039868	3.01	0.20
25	0.034953	3.44	0.14
30	0.037524	3.21	0.11

Код программы

main.c

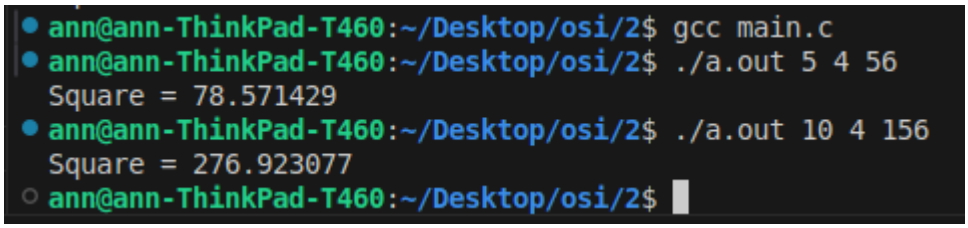
```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <time.h>
#include <string.h>
#include <unistd.h>

#define MAX_THREADS 4

typedef struct {
    int num_points;
    int inside_circle;
    pthread_mutex_t* mutex;
    int* total_inside_circle;
    double radius;
} ThreadData;

void* monte_carlo(void* arg) {
    ThreadData* data = (ThreadData*)arg;
    int inside = 0;

    srand(time(NULL));
    double radius = data->radius;

    for (int i = 0; i < data->num_points; i++) {
        double x = ((double)rand() / RAND_MAX) * 2 * radius - radius;
        double y = ((double)rand() / RAND_MAX) * 2 * radius - radius;
        if (x * x + y * y <= radius * radius) {
            inside++;
        }
    }

    pthread_mutex_lock(data->mutex);
```

```
    *(data->total_inside_circle) += inside;
    pthread_mutex_unlock(data->mutex);

    data->inside_circle = inside;
    return NULL;
}
int main(int argc, char* argv[]) {
    if (argc != 4) {
        char err[] = "Usage: <radius> <num_threads> <num_points>\n";
        write(STDERR_FILENO, err, sizeof(err));
        return 1;
    }

    double radius = atof(argv[1]);
    int num_threads = atoi(argv[2]);
    int num_points = atoi(argv[3]);

    if (radius < 0 || num_threads > MAX_THREADS || num_points <= 0) {
        char err[] = "Invalid INPUT\n";
        write(STDERR_FILENO, err, sizeof(err));
        return 1;
    }

    int points_per_thread = num_points / num_threads;
    int total_inside_circle = 0;

    pthread_t threads[MAX_THREADS];
    ThreadData thread_data[MAX_THREADS];
    pthread_mutex_t mutex;

    if (pthread_mutex_init(&mutex, NULL) != 0) {
        char err[] = "Problems with init mutex\n";
        write(STDERR_FILENO, err, sizeof(err));
        return 1;
```

```
    }

    for (int i = 0; i < num_threads; i++) {
        thread_data[i].num_points = points_per_thread;
        thread_data[i].mutex = &mutex;
        thread_data[i].total_inside_circle = &total_inside_circle;
        thread_data[i].radius = radius;
        pthread_create(&threads[i], NULL, monte_carlo, &thread_data[i]);
    }
    for (int i = 0; i < num_threads; i++) {
        pthread_join(threads[i], NULL);
    }
    pthread_mutex_destroy(&mutex);

    double estimated_pi = (4.0 * total_inside_circle) / num_points;
    double area = estimated_pi * radius * radius;

    char* result = (char*)malloc(50 * sizeof(char));
    if (result == NULL) {
        char err[] = "Memory allocation error\n";
        write(STDERR_FILENO, err, strlen(err));
        return 1;
    }
    if (sprintf(result, "Square = %f\n", area) < 0) {
        char err[] = "Overflow\n";
        write(STDERR_FILENO, err, sizeof(err));
        free(result);
        return 1;
    }
    write(STDOUT_FILENO, result, strlen(result));
    free(result);
    return 0;
}
```

Протокол работы программы

Тестирование:



Strace:

ann@ann-ThinkPad-T460:~/Desktop/osi/2$ strace ./a.out 10 4 156

execve("./a.out", ["./a.out", "10", "4", "156"], 0x7ffcdec2e138 /* 71 vars */) = 0

brk(NULL) = 0x5c8802f29000

arch_prctl(0x3001 /* ARCH_??? */, 0x7ffd32eccd60) = -1 EINVAL (Недопустимый аргумент)

mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x760d2e660000

access("/etc/ld.so.preload", R_OK) = -1 ENOENT (Нет такого файла или каталога)

openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3

newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=66559, ...}, AT_EMPTY_PATH) = 0

mmap(NULL, 66559, PROT_READ, MAP_PRIVATE, 3, 0) = 0x760d2e64f000

close(3) = 0

openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3

read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0P\237\2\0\0\0\0"..., 832) = 832

pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784

pread64(3, "\4\0\0\0 \0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0\0"..., 48, 848) = 48

pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0I\17\357\204\3$\f\221\2039x\324\224\323\236S"..., 68, 896) = 68

newfstatat(3, "", {st_mode=S_IFREG|0755, st_size=2220400, ...}, AT_EMPTY_PATH) = 0

pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784

mmap(NULL, 2264656, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x760d2e400000

mprotect(0x760d2e428000, 2023424, PROT_NONE) = 0

mmap(0x760d2e428000, 1658880, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x760d2e428000

mmap(0x760d2e5bd000, 360448, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1bd000) = 0x760d2e5bd000

mmap(0x760d2e616000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x215000) = 0x760d2e616000

mmap(0x760d2e61c000, 52816, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x760d2e61c000

close(3) = 0

mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x760d2e64c000

arch_prctl(ARCH_SET_FS, 0x760d2e64c740) = 0

set_tid_address(0x760d2e64ca10) = 37430

set_robust_list(0x760d2e64ca20, 24) = 0

rseq(0x760d2e64d0e0, 0x20, 0, 0x53053053) = 0

mprotect(0x760d2e616000, 16384, PROT_READ) = 0

mprotect(0x5c880237e000, 4096, PROT_READ) = 0

mprotect(0x760d2e69a000, 8192, PROT_READ) = 0

prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0

munmap(0x760d2e64f000, 66559) = 0

rt_sigaction(SIGRT_1, {sa_handler=0x760d2e491870, sa_mask=[], sa_flags=SA_RESTORER|SA_ONSTACK|SA_RESTART|SA_SIGINFO, sa_restorer=0x760d2e442520}, NULL, 8) = 0

rt_sigprocmask(SIG_UNBLOCK, [RTMIN RT_1], NULL, 8) = 0

mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) = 0x760d2da00000

mprotect(0x760d2da01000, 8388608, PROT_READ|PROT_WRITE) = 0

getrandom("\xf8\xd4\xbf\xc7\x94\x12\x11\x11", 8, GRND_NONBLOCK) = 8

brk(NULL) = 0x5c8802f29000

brk(0x5c8802f4a000) = 0x5c8802f4a000

rt_sigprocmask(SIG_BLOCK, ~[], [], 8) = 0

clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, child_tid=0x760d2e200910, parent_tid=0x760d2e200910, exit_signal=0, stack=0x760d2da00000, stack_size=0x7fff00, tls=0x760d2e200640} => {parent_tid=[37431]}, 88) = 37431

rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0

mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) = 0x760d2d000000

mprotect(0x760d2d001000, 8388608, PROT_READ|PROT_WRITE) = 0

rt_sigprocmask(SIG_BLOCK, ~[], [], 8) = 0

clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, child_tid=0x760d2d800910, parent_tid=0x760d2d800910, exit_signal=0, stack=0x760d2d000000, stack_size=0x7fff00, tls=0x760d2d800640} => {parent_tid=[0]}, 88) = 37432

rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0

mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) = 0x760d2c600000

mprotect(0x760d2c601000, 8388608, PROT_READ|PROT_WRITE) = 0

rt_sigprocmask(SIG_BLOCK, ~[], [], 8) = 0

clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, child_tid=0x760d2ce00910, parent_tid=0x760d2ce00910, exit_signal=0, stack=0x760d2c600000, stack_size=0x7fff00, tls=0x760d2ce00640} => {parent_tid=[0]}, 88) = 37433

rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0

mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) = 0x760d2bc00000

mprotect(0x760d2bc01000, 8388608, PROT_READ|PROT_WRITE) = 0

rt_sigprocmask(SIG_BLOCK, ~[], [], 8) = 0

clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, child_tid=0x760d2c400910, parent_tid=0x760d2c400910, exit_signal=0, stack=0x760d2bc00000, stack_size=0x7fff00, tls=0x760d2c400640} => {parent_tid=[37434]}, 88) = 37434

rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0

write(1, "Square = 297.435897\n", 20Square = 297.435897

) = 20

exit_group(0)   = ?

+++ exited with 0 +++

Вывод

В ходе лабораторной работы мной были приобретены навыки в управлении потоками в ОС и обеспечении синхронизации между потоками. Так же было продемонстрировано количество потоков, используемое написанной программой с помощью стандартных средств операционной системы и проведено исследование зависимости ускорения и эффективности алгоритма от входных данных и количества потоков.