

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-216БВ-24

Студент: Лукиянчук А.О.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 23.12.25

Москва, 2025

Постановка задачи

Вариант 16

Составить программу на языке Си, обрабатывающую данные в многопоточном режиме. При обработке 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.

Также необходимо уметь продемонстрировать количество потоков, используемое вашей программой с помощью стандартных средств операционной системы. В отчёте привести исследование зависимости ускорения и эффективности алгоритма от входных данных и количества потоков. Получившиеся результаты необходимо объяснить.

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

Общий метод и алгоритм решения

Использованные системные вызовы и функции:

- `int pthread_create(pthread_t* thread, const pthread_attr_t* attr, void* (*start_routine) (void*), void* arg)` – создает новый поток выполнения.
- `int pthread_join(pthread_t thread, void** thread_return)` – ожидает завершения указанного потока и получает его возвращаемое значение.
- `int gettimeofday(struct timeval* tv, struct timezone* tz)` – получает текущее время с микросекундной точностью.
- `void exit(int status)` – завершает выполнение процесса с указанным статусом.
- `ssize_t write(int fd, const void* buf, size_t count)` – записывает данные из буфера в файловый дескриптор.

Алгоритм метода Монте-Карло для вычисления площади окружности:

1. Известно, что окружность радиуса R вписана в квадрат со стороной $2R$.
2. Площадь квадрата: $S_{\text{кв}} = (2R)^2 = 4R^2$.
3. Генерируем N случайных точек равномерно распределённых внутри квадрата.
4. Подсчитываем количество точек M , попавших внутрь окружности (удовлетворяющих условию $x^2 + y^2 \leq R^2$).
5. Площадь окружности: $S = \pi R^2$

Параллельная реализация:

- Программа принимает три аргумента: количество потоков, общее количество точек и радиус окружности.
- Общее количество точек равномерно распределяется между потоками.
- Каждый поток использует свой линейный конгруэнтный генератор случайных чисел с уникальным seed для обеспечения независимости генерации.
- Каждый поток подсчитывает количество точек, попавших в окружность, на своей порции данных.
- Главный поток суммирует результаты всех потоков и вычисляет итоговую оценку площади.
- Измеряется время выполнения программы для анализа ускорения.

Анализ ускорения и эффективности:

Число потоков	Время исполнения (мс)	Ускорение	Эффективность	Погрешность
1	4836.350	1.00	1.00	0.000014
2	4226.966	1.14	0.57	0.000043
8	2570.958	1.88	0.235	0.000037
16	2321.788	2.08	0.130	0.000014
22	1382.939	3.50	0.159	-0.000029
64	1367.058	3.54	0.055	0.000011
128	1621.683	2.98	0.023	-0.000012
1024	721.318	6.71	0.0065	-0.000056

Ускорение показывает во сколько раз применение параллельного алгоритма уменьшает время решения задачи по сравнению с применением последовательного алгоритма. Ускорение определяется величиной $SN = T1 / TN$, где $T1$ – время выполнения на одном потоке, TN – время на N потоках.

Эффективность – величина $EN = SN / N$, где SN – ускорение, N – количество используемых потоков.



Анализ результатов:

Графики демонстрируют закон Амдала, который иллюстрирует ограничение роста производительности вычислительной системы с увеличением числа вычислителей. Наблюдаются следующие эффекты:

1. При увеличении числа потоков от 1 до 22 наблюдается значительный рост ускорения (до 3.5 раз).
2. При дальнейшем увеличении числа потоков до 64 ускорение продолжает расти, но медленнее.
3. При переходе к 128 потокам наблюдается снижение ускорения из-за накладных расходов на переключение контекстов между потоками.
4. При 1024 потоках ускорение резко возрастает до 6.71, что может объясняться особенностями планировщика ОС и использованием гиперпоточности процессора.

Эффективность использования потоков закономерно падает с увеличением их количества, что соответствует теоретическим ожиданиям. Наилучший баланс между ускорением и эффективностью наблюдается при 8-22 потоках.

Код программы

```
[main.c]
#define _POSIX_C_SOURCE 200112L
#include <stdio.h>
#include <pthread.h>
#include <unistd.h>
#include <string.h>
#include <stdlib.h>
#include <sys/time.h>
#include "montecarlo.h"

#define PI 3.14159265358979323846

int main(int argc, char** argv) {
    if (argc != 4) {
        fprintf(stderr, "Usage: %s <max_threads> <total_points> <radius>\n", argv[0]);
        return 1;
    }

    struct timeval start, end;
    gettimeofday(&start, NULL);

    uint64_t max_threads = strtoull(argv[1], NULL, 10);
    uint64_t total_points = strtoull(argv[2], NULL, 10);
    double radius = strtod(argv[3], NULL);

    if (max_threads == 0 || total_points == 0 || radius <= 0) {
        fprintf(stderr, "Error: invalid parameters\n");
        return 1;
    }

    uint64_t points_per_thread = total_points / max_threads;
    uint64_t extra_points = total_points % max_threads;

    pthread_t* threads = malloc(sizeof(pthread_t) * max_threads);
    ThreadData* thread_data = malloc(sizeof(ThreadData) * max_threads);

    if (!threads || !thread_data) {
        fprintf(stderr, "Error: memory allocation failed\n");
        free(threads);
        free(thread_data);
        return 1;
    }
```

```

}

uint32_t base_seed = time(NULL);

for (uint64_t i = 0; i < max_threads; ++i) {
    thread_data[i].radius = radius;
    thread_data[i].points_per_thread = points_per_thread + (i < extra_points ? 1 :
0);
    thread_data[i].seed = base_seed + i;
    thread_data[i].hits = 0;
}

for (uint64_t i = 0; i < max_threads; ++i) {
0) {    if (pthread_create(&threads[i], NULL, monte_carlo_thread, &thread_data[i]) !=
        fprintf(stderr, "Error creating thread %lu\n", i);
        for (uint64_t j = 0; j < i; ++j) {
            pthread_join(threads[j], NULL);
        }
        free(threads);
        free(thread_data);
        return 1;
    }
}

for (uint64_t i = 0; i < max_threads; ++i) {
    pthread_join(threads[i], NULL);
}

uint64_t total_hits = 0;
for (uint64_t i = 0; i < max_threads; ++i) {
    total_hits += thread_data[i].hits;
}

double estimated_area = 4.0 * radius * radius * (double)total_hits / total_points;
double exact_area = PI * radius * radius;

printf("Radius: %.6f\n", radius);
printf("Total points: %lu\n", total_points);
printf("Points inside circle: %lu\n", total_hits);
printf("Estimated area: %.6f\n", estimated_area);
printf("Exact area: %.6f\n", exact_area);
printf("Error: %.6f\n", estimated_area - exact_area);
printf("Threads used: %lu\n", max_threads);

gettimeofday(&end, NULL);
long seconds = end.tv_sec - start.tv_sec;
long microseconds = end.tv_usec - start.tv_usec;
double elapsed_ms = seconds * 1000.0 + microseconds / 1000.0;

```

```

    printf("Execution time: %.3f ms\n", elapsed_ms);

    free(threads);
    free(thread_data);

    return 0;
}
[src/montecarlo.c]
#include "../include/montecarlo.h"
#include <stdint.h>

static uint32_t lcg(uint32_t* seed) {
    *seed = *seed * 1103515245 + 12345;
    return *seed;
}

void* monte_carlo_thread(void* arg) {
    ThreadData* data = (ThreadData*)arg;
    data->hits = 0;

    uint32_t seed = data->seed;

    for (uint64_t i = 0; i < data->points_per_thread; ++i) {
        double x = (double)lcg(&seed) / (1ULL << 32) * 2 * data->radius - data->radius;
        double y = (double)lcg(&seed) / (1ULL << 32) * 2 * data->radius - data->radius;

        if (x*x + y*y <= data->radius * data->radius) {
            data->hits++;
        }
    }

    return NULL;
}
[include/montecarlo.h]
#pragma once

#include <stdint.h>
#include <pthread.h>

typedef struct {
    uint64_t points_per_thread;
    double radius;
    uint32_t seed;
    uint64_t hits;
} ThreadData;

void* monte_carlo_thread(void* arg);
[Makefile]
CC = gcc

```

```

CFLAGS = -I./include -Wall -Wextra -O2 -pthread -D_POSIX_C_SOURCE=200112L
LDFLAGS = -pthread -lm

TARGET = main
SRCS = main.c src/montecarlo.c
OBJS = $(SRCS:.c=.o)

all: $(TARGET)

$(TARGET): $(OBJS)
    $(CC) -o $@ $^ $(LDFLAGS)

%.o: %.c
    $(CC) $(CFLAGS) -c $< -o $@

clean:
    rm -f $(OBJS) $(TARGET)

run: all
    ./$(TARGET) 4 1000000 1.0

.PHONY: all clean run

```

Протокол работы программы

Тестирование:

```

→ lab2 ./montecarlo 1 1000000000 1.0
[18:12:17]
Radius: 1.000000
Total points: 1000000000
Points inside circle: 785401633
Estimated area: 3.141607
Exact area: 3.141593
Error: 0.000014
Threads used: 1
Execution time: 4836.350 ms

→ lab2 ./montecarlo 22 1000000000 1.0
[18:14:26]
Radius: 1.000000
Total points: 1000000000
Points inside circle: 785390915
Estimated area: 3.141564
Exact area: 3.141593
Error: -0.000029
Threads used: 22
Execution time: 1382.939 ms

```

→ lab2 ./montecarlo 1024 1000000000 1.0
[18:16:28]

Radius: 1.000000
Total points: 1000000000
Points inside circle: 785384097
Estimated area: 3.141536
Exact area: 3.141593
Error: -0.000056
Threads used: 1024
Execution time: 721.318 ms

→ lab2 strace -f -e trace=clone,futex,brk,mmap,munmap,gettimeofday ./montecarlo 4 1000000
5.0

[14:29:46]

```
brk(NULL) = 0x55a91abee000
mmap(NULL, 166763, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f92b1c7e000
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f92b1c7c000
mmap(NULL, 1102152, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f92b1b6e000
mmap(0x7f92b1b7d000, 569344, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x0) = 0x7f92b1b7d000
mmap(0x7f92b1c08000, 466944, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x9a000) = 0x7f92b1c08000
mmap(0x7f92b1c7a000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x10b000) = 0x7f92b1c7a000
mmap(NULL, 2169904, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f92b1800000
mmap(0x7f92b1824000, 1511424, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x24000) = 0x7f92b1824000
mmap(0x7f92b1995000, 454656, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x195000) = 0x7f92b1995000
mmap(0x7f92b1a04000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x203000) = 0x7f92b1a04000
mmap(0x7f92b1a0a000, 31792, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f92b1a0a000
mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f92b1b6b000
munmap(0x7f92b1c7e000, 166763) = 0
brk(NULL) = 0x55a91abee000
brk(0x55a91ac0f000) = 0x55a91ac0f000
mmap(NULL, 8392704, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) = 0x7f92b0fff000
mmap(NULL, 8392704, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) = 0x7f92b07fe000
strace: Process 65593 attached
strace: Process 65592 attached
[pid 65591] mmap(NULL, 8392704, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) = 0x7f92afffd000
strace: Process 65594 attached
[pid 65591] mmap(NULL, 8392704, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) = 0x7f92af7fc000
strace: Process 65595 attached
[pid 65591] futex(0x7f92b17ff990, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 65592, NULL, FUTEX_BITSET_MATCH_ANY <unfinished ...>
```



```
[pid 65593] +++ exited with 0 +++
[pid 65591] <... futex resumed>          = 0
[pid 65592] +++ exited with 0 +++
[pid 65591] futex(0x7f92b07fd990, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 65594, NULL,
FUTEX_BITSET_MATCH_ANY) = 0
[pid 65594] +++ exited with 0 +++
[pid 65591] futex(0x7f92afffc990, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 65595, NULL,
FUTEX_BITSET_MATCH_ANY) = 0
[pid 65595] +++ exited with 0 +++
Radius: 5.000000
Total points: 1000000
Points inside circle: 784802
Estimated area: 78.480200
Exact area: 78.539816
Error: -0.059616
Threads used: 4
Execution time: 5.250 ms
+++ exited with 0 +++
```

Вывод

В ходе выполнения лабораторной работы были изучены и применены основные средства работы с потоками в ОС Linux с использованием библиотеки pthreads. Была создана программа, реализующая параллельный алгоритм метода Монте-Карло для вычисления площади окружности.

В процессе работы были исследованы зависимости ускорения и эффективности алгоритма от количества потоков. Результаты подтвердили действие закона Амдала: с увеличением числа потоков ускорение растёт, но эффективность использования каждого дополнительного потока снижается. Наилучшие показатели ускорения были достигнуты при использовании 22-64 потоков, однако эффективность при этом существенно падала.

Возникшие сложности включали необходимость обеспечения независимости генераторов случайных чисел в разных потоках и корректного распределения нагрузки. Эти проблемы были успешно решены путём использования уникальных seed для каждого потока и равномерного распределения точек.

Работа позволила получить практические навыки создания многопоточных приложений, измерения их производительности и анализа полученных результатов, что является важным для разработки эффективных параллельных алгоритмов.