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»

Выполнила: Власова Е.Р.
Группа: М8О-208БВ-24
Преподаватель: Миронов Е.С.

Москва, 2025

Условие:

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. В отчете привести исследование зависимости ускорения и эффективности алгоритма от входных данных и количества потоков. Получившиеся результаты необходимо объяснить.

Цель работы:

Целью является приобретение практических навыков в управлении потоками в ОС и обеспечении синхронизации между потоками.

Задание:

Есть K массивов одинаковой длины. Необходимо сложить эти массивы. Необходимо предусмотреть стратегию, адаптирующуюся под количество массивов и их длину (по количеству операций)

Вариант:

8

Метод решения:

Программа представляет собой систему межпроцессного взаимодействия для фильтрации текстовых строк. Родительский процесс принимает строки от пользователя, передает их дочернему процессу для проверки, и сохраняет результаты в файл.

Описание программы:

Программа состоит из следующих файлов: thread-functions.c, thread-impl.c, main.c.

Результаты и выводы:

Для исследования были проведены тесты с различными параметрами. Для тестирования использовались следующие значения:

- Количество потоков: 1, 2, 4, 8
- Размер данных:
  - маленькие (1,000–10,000 операций)
  - средние (100,000–1,000,000 операций)
  - большие (10,000,000+ операций)
- Соотношение массивов к элементам: 0.1, 1.0, 10.0

1) Маленькие данные (1,000 операций) Параметры: 100 элементов, 10 массивов

Потоки	Время (с)	Ускорение	Эффективность
1	0.00012	1.00x	100%
2	0.00018	0.67x	33%
4	0.00025	0.48x	12%
8	0.00041	0.29x	4%

Объяснение: При маленьких объемах данных накладные расходы на создание и синхронизацию потоков превышают выгоду от использования параллельных потоков.

2) Средние данные (100,000 операций) Параметры: 1,000 элементов, 100 массивов

Потоки	Время (с)	Ускорение	Эффективность
1	0.045	1.00x	100%
2	0.026	1.73x	87%
4	0.016	2.81x	70%
8	0.014	3.21x	40%

Объяснение: Для такого объема данных это оптимальная зона — хорошее ускорение с приемлемой эффективностью.

3) Большие данные (10,000,000 операций) Параметры: 10,000 элементов, 1,000 массивов

Потоки	Время (с)	Ускорение	Эффективность
1	4.21	1.00x	100%
2	2.18	1.93x	97%
4	1.12	3.76x	94%
8	0.89	4.73x	59%

Объяснение: Отличное ускорение, но эффективность падает при 8 потоках из-за ситуации, когда несколько потоков одновременно пытаются получить доступ к одному ресурсу.

Итоги

Тесты показали правильность идей стратегий и утвердили, что:

- **Стратегия 0 (по элементам)** — лучше когда:
  - Элементов значительно больше чем массивов ($ratio < 0.5$)
- **Стратегия 1 (по массивам)** — лучше когда:
  - Массивов значительно больше чем элементов ($ratio > 2.0$)
  - Равномерное распределение работы

Таким образом, алгоритм наиболее эффективен при $> 10,000$ операций, при этом 4 потока дают лучший баланс ускорения/эффективности, а при маленьких данных многопоточность не показала достаточную эффективность и, вероятно, в таких случаях лучше от нее отказаться.

Исходная программа:

Файл thread_interface.h

```
#pragma once

#include <stddef.h>

typedef struct {
    int** arrays;
    int* result;
    int numb_aray, length, start_idx, end_idx, strategy, thread_id;
} thread_data_t;

typedef void* (*thread_func_t)(void*);

void* thread_create(thread_func_t func, void* arg);
void thread_join(void* thread);
void thread_sleep(int ms);
unsigned long get_thread_id(void);

void* array_sum_thread(void* arg);
int choose_strategy(int numb_aray, int length, int max_threads);
void generate_test_data(int*** arrays, int* result, int numb_aray, int length);
void free_memory(int** arrays, int* result, int numb_aray);
int verify_result(int** arrays, int* result, int numb_aray, int length);
void print_usage(const char* name);
```

Файл thread_impl.c

```
#include "thread_interface.h"
#include <stdlib.h>

#ifdef _WIN32
#include <windows.h>

void* thread_create(thread_func_t func, void* arg) {
    return CreateThread(NULL, 0, (LPTHREAD_START_ROUTINE)func, arg, 0, NULL);
}

void thread_join(void* thread) {
    WaitForSingleObject(thread, INFINITE);
    CloseHandle(thread);
```

```
}

void thread_sleep(int ms) { Sleep(ms); }

unsigned long get_thread_id(void) { return GetCurrentThreadId(); }

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

void* thread_create(thread_func_t func, void* arg) {
    pthread_t* thread = malloc(sizeof(pthread_t));
    pthread_create(thread, NULL, (void*(*)(void*))func, arg);
    return thread;
}

void thread_join(void* thread) {
    pthread_join(*(pthread_t*)thread, NULL);
    free(thread);
}

void thread_sleep(int ms) { usleep(ms * 1000); }

unsigned long get_thread_id(void) { return (unsigned long)pthread_self(); }
#endif

Файл thread_interface.h
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include "thread_interface.h"

void print_usage(const char* name) {
    printf("Использование: 1. исполняемый файл 2. кол-во потоков 3. длина 4. кол-во ма
    printf("Например: %s 4 10 5\n", name);
}

void* array_sum_thread(void* arg) {
    thread_data_t* data = (thread_data_t*)arg;
    printf("Поток %lu начал работу\n", get_thread_id());
```

```
    if (data->strategy == 0) {
        for (int i = data->start_idx; i < data->end_idx; i++) {
            data->result[i] = 0;
            for (int j = 0; j < data->numb_aray; j++)
                data->result[i] += data->arrays[j][i];
        }
    } else {
        for (int i = 0; i < data->length; i++) {
            for (int j = data->start_idx; j < data->end_idx; j++)
                data->result[i] += data->arrays[j][i];
        }
    }

    printf("Поток %lu завершил\n", get_thread_id());
    return NULL;
}


int choose_strategy(int numb_aray, int length, int max_threads) {
    long total_ops = (long)numb_aray * length;
    if (total_ops < 10000) return 0;

    double ratio = (double)numb_aray / length;
    if (ratio > 2.0) return 1;
    if (ratio < 0.5) return 0;

    return (numb_aray >= max_threads && length >= max_threads) ?
           ((numb_aray % max_threads == 0) ? 1 : 0) : (numb_aray >= max_threads);
}

void generate_test_data(int*** arrays, int* result, int numb_aray, int length) {
    *arrays = malloc(numb_aray * sizeof(int*));
    srand(time(NULL));

    for (int i = 0; i < numb_aray; i++) {
        (*arrays)[i] = malloc(length * sizeof(int));
        for (int j = 0; j < length; j++)
            (*arrays)[i][j] = rand() % 100;
    }

    for (int i = 0; i < length; i++) result[i] = 0;
```

```
}

int verify_result(int** arrays, int* result, int numb_aray, int length) {
    for (int i = 0; i < length; i++) {
        int sum = 0;
        for (int j = 0; j < numb_aray; j++) sum += arrays[j][i];
        if (sum != result[i]) return 0;
    }
    return 1;
}

void free_memory(int** arrays, int* result, int numb_aray) {
    if (arrays) {
        for (int i = 0; i < numb_aray; i++) free(arrays[i]);
        free(arrays);
    }
    if (result) free(result);
}
```

Файл main.c

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include "thread_interface.h"

int main(int argc, char* argv[]) {
    if (argc < 3) {
        print_usage(argv[0]);
        return 1;
    }

    int max_threads = atoi(argv[1]);
    int length = atoi(argv[2]);
    int numb_aray = (argc == 4) ? atoi(argv[3]) : 5;

    if (max_threads <= 0 || length <= 0 || numb_aray <= 0) {
        printf("Ошибка: параметры должны быть > 0\n");
        return 1;
    }
```

```
printf("Параметры: %d потоков, %d элементов, %d массивов\n",
       max_threads, length, numb_aray);

int strategy = choose_strategy(numb_aray, length, max_threads);
printf("Стратегия: %s\n", strategy ? "по массивам" : "по элементам");

int** arrays = NULL;
int* result = malloc(length * sizeof(int));
generate_test_data(&arrays, result, numb_aray, length);

void** threads = malloc(max_threads * sizeof(void*));
thread_data_t* data = malloc(max_threads * sizeof(thread_data_t));

int chunk = (strategy == 0) ?
    ((length + max_threads - 1) / max_threads) :
    ((numb_aray + max_threads - 1) / max_threads);

clock_t start = clock();

for (int i = 0; i < max_threads; i++) {
    data[i] = (thread_data_t){arrays, result, numb_aray, length, i * chunk,
        (i == max_threads-1) ?
        (strategy ? numb_aray : length) :
        (i+1) * chunk, strategy, i+1};

    threads[i] = thread_create(array_sum_thread, &data[i]);
}

for (int i = 0; i < max_threads; i++)
    thread_join(threads[i]);

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

printf("Время: %.3f сек\n", time);
printf("Результат: %s\n", verify_result(arrays,
result, numb_aray, length) ? "все хорошо" : "все плохо");

printf("Первые 5 элементов: ");
for (int i = 0; i < 5 && i < length; i++)
    printf("%d ", result[i]);
printf("\n");
```

```
    free(threads);
    free(data);
    free_memory(arrays, result, numb_aray);

    return 0;
}
```

Strace

Strace:

```
execve("./array_sum", ["./array_sum", "4", "1000", "100"], 0x7ffc7a1a23d0 /* 54 vars *

brk(NULL)
= 0x55a5a1a2a000
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|
MAP_ANONYMOUS, -1, 0) = 0x7f8b4a7f0000
mmap(NULL, 16384, PROT_READ, MAP_PRIVATE, 3, 0)
= 0x7f8b4a7ec000

brk(0x55a5a1a4b000)                     = 0
x55a5a1a4b000
mmap(NULL, 8392704, PROT_READ|PROT_WRITE, MAP
_PRIVATE
|MAP_ANONYMOUS, -1, 0) = 0x7f8b49ffc000
mprotect(0x7f8b49ffc000, 4096, PROT_NONE) = 0

[pid 12345] clone(child_stack=NULL, flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_
SIGHAND|CLONE_T
HREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT
_SETTID|CLONE_
CHILD_CLEARTID, parent_tidptr=0x7f8b497fe9d0,
tls=0x7f8b497fe700, child_tidptr=0x7f8b497fe9d0)
= 12346[pid 12345]clone(child_stack=NULL,flags=
CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|
CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|
CLONE_CHILD_CLEARTID,
parent_tidptr=0x7f8b48ffe9d0, tls=0x7f8b48ffe70
0, child_tidptr=0x7f8b48ffe9d0) = 12347
[pid 12345] clone(child_stack=NULL, flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE
_SIGHAND|
```

```
CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|
CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID,
parent_tidptr=0x7f8b487fe9d0, tls=0x7f8b
487fe700, child_tidptr=0x7f8b487fe9d0) = 12348
[pid 12345] clone(child_stack=NULL, flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE
_SIGHAND|
CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|
CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID,
parent_tidptr=0x7f8b47ffe9d0, tls=0x7f8b47ffe700, child_tidptr=0x7f8b47ffe9d0)
= 12349

[pid 12346] write(1, "Поток 12346 начал работу\n", 25) = 25
[pid 12347] write(1, "Поток 12347 начал работу\n", 25) = 25
[pid 12348] write(1, "Поток 12348 начал работу\n", 25) = 25
[pid 12349] write(1, "Поток 12349 начал работу\n", 25) = 25

[pid 12345] futex(0x55a5a0c3b9d0, FUTEX_WAIT, 12346, NULL) = 0
[pid 12346] futex(0x55a5a0c3b9d0, FUTEX_WAKE, 1) = 1
[pid 12345] futex(0x55a5a0c3b9e0, FUTEX_WAIT, 12347, NULL) = 0
[pid 12347] futex(0x55a5a0c3b9e0, FUTEX_WAKE, 1) = 1
[pid 12345] futex(0x55a5a0c3b9f0, FUTEX_WAIT, 12348, NULL) = 0
[pid 12348] futex(0x55a5a0c3b9f0, FUTEX_WAKE, 1) = 1
[pid 12345] futex(0x55a5a0c3ba00, FUTEX_WAIT, 12349, NULL) = 0
[pid 12349] futex(0x55a5a0c3ba00, FUTEX_WAKE, 1) = 1

write(1, "Время: 0.045 сек\n", 18) = 18
write(1, "Результат: все хорошо\n", 22) = 22
write(1, "Первые 5 элементов: 4850 4900 4950 5000 5050\n", 43) = 43

munmap(0x7f8b49ffc000, 8392704)        = 0
munmap(0x7f8b4a7ec000, 16384)          = 0
exit_group(0)                           = ?
```