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-210БВ-24

Студент: Лабутин М.А.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 14.10.25

Москва, 2025

Постановка задачи

Вариант 18.

Найти образец в строке наивным алгоритмом.

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 новый поток, который начинает выполнение функции поиска в своем диапазоне данных.
- int pthread_join(pthread_t thread, void **retval); — ожидает завершения выполнения потока, чтобы собрать результаты подсчета совпадений.
- int pthread_attr_setstacksize(pthread_attr_t *attr, size_t stacksize); — устанавливает размер стека для потоков (используется для возможности создания большого количества потоков, например 1024 и более).
- int pthread_mutex_unlock(pthread_mutex_t *mutex); – разблокирует мьютекс.
- int clock_gettime(clockid_t clk_id, struct timespec *tp); — получает текущее монотонное время системы для точного замера времени выполнения (T1 и Tn).
- struct timespec { time_t tv_sec; long tv_nsec; }; — структура, хранящая время в секундах и наносекундах.

Я реализовал программу, которая использует многопоточность для поиска количества вхождений заданного образца (pattern) в строке (text) с использованием **наивного алгоритма**. Суть алгоритма заключается в последовательном сравнении образца с подстроками основного текста, начиная с каждой возможной позиции.

Механизм распределения: Количество потоков задается пользователем через аргумент командной строки с флагом -t. Весь массив данных (текст) делится на равные части (чанки). Каждый поток получает свой диапазон индексов start_index и end_index, в пределах которых он выполняет сравнение. Чтобы избежать пропуска совпадений на «границах» разделения, каждый поток проверяет позиции вплоть до конца своего отрезка, учитывая длину паттерна.

Также я использую **mutex**, чтобы предотвратить состояние гонки (race condition) при обновлении общего счетчика найденных вхождений global_match_count из нескольких потоков одновременно.

Код программы

main.c

```c
#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <pthread.h>

#include <time.h>

#include <unistd.h>
```

```c
#include <getopt.h>


#define THREAD_STACK_SIZE (64 * 1024)

long global_match_count = 0;

pthread_mutex_t count_mutex;

typedef struct {

    const char* text;

    const char* pattern;

    int start_index;

    int end_index;

    int pattern_len;

} ThreadData;

double get_time() {

    struct timespec ts;

    clock_gettime(CLOCK_MONOTONIC, &ts);

    return ts.tv_sec + ts.tv_nsec * 1e-9;

}

void* naive_search_thread(void* arg) {

    ThreadData* data = (ThreadData*)arg;

    long local_count = 0;

    for (int i = data->start_index; i <= data->end_index; i++) {

        int j;

        for (j = 0; j < data->pattern_len; j++) {

            if (data->text[i + j] != data->pattern[j]) break;

        }

        if (j == data->pattern_len) {

            local_count++;

        }

    }

    pthread_mutex_lock(&count_mutex);

    global_match_count += local_count;

    pthread_mutex_unlock(&count_mutex);

    return NULL;

}

int main(int argc, char* argv[]) {

    int num_threads = 1;
```

```c
    int opt;
    while ((opt = getopt(argc, argv, "t:")) != -1) {
        if (opt == 't') num_threads = atoi(optarg);
    }
    if (num_threads < 1) num_threads = 1;
    pthread_mutex_init(&count_mutex, NULL);
    size_t text_len = 50 * 1024 * 1024;
    char* text = malloc(text_len + 1);
    memset(text, 'A', text_len);
    const char* pattern = "ABACABA";
    int pat_len = strlen(pattern);
    for (int i = 0; i < 1000; i++) {
        memcpy(text + (rand() % (text_len - pat_len)), pattern, pat_len);
    }
    text[text_len] = '\0';
    global_match_count = 0;
    ThreadData seq_data = {text, pattern, 0, (int)(text_len - pat_len), pat_len};
    double start_t1 = get_time();
    naive_search_thread(&seq_data);
    double end_t1 = get_time();
    double t1 = end_t1 - start_t1;
    long seq_matches = global_match_count;
    global_match_count = 0;
    pthread_t* threads = malloc(num_threads * sizeof(pthread_t));
    ThreadData* t_args = malloc(num_threads * sizeof(ThreadData));

    pthread_attr_t attr;
    pthread_attr_init(&attr);
    pthread_attr_setstacksize(&attr, THREAD_STACK_SIZE);
    int positions = text_len - pat_len + 1;
    int chunk = positions / num_threads;
    double start_tn = get_time();
    for (int i = 0; i < num_threads; i++) {
        t_args[i].text = text;
        t_args[i].pattern = pattern;
        t_args[i].pattern_len = pat_len;
```

```
            t_args[i].start_index = i * chunk;
            t_args[i].end_index = (i == num_threads - 1) ? (positions - 1) : ((i + 1) * chunk - 1);
            pthread_create(&threads[i], &attr, naive_search_thread, &t_args[i]);
        }
        for (int i = 0; i < num_threads; i++) {
            pthread_join(threads[i], NULL);
        }
        double end_tn = get_time();
    double tn = end_tn - start_tn;
        double speedup = t1 / tn;
        double efficiency = speedup / num_threads;
        printf("\nРезультаты эксперимента:\n");
        printf("Потоков:          %d\n", num_threads);
        printf("Время(сек):       %.6f\n", tn);
        printf("Ускорение:        %.2f\n", speedup);
        printf("Эффективность:    %.4f\n", efficiency);
        printf("\nНайдено совпадений: %ld (Проверка: %s)\n",
               global_match_count, global_match_count == seq_matches ? "OK" : "FAIL");
        pthread_mutex_destroy(&count_mutex);
        pthread_attr_destroy(&attr);
        free(text); free(threads); free(t_args);
        return 0;
    }
```

Работа программы

```
mxtv1x@WIN-EMVSNHTNF8D:/mnt/c/Users/User/MAI-LABS/OS-25/lab2$ ./main -t 1

Результаты эксперимента:
Потоков:          1
Время(сек):       0.136094
Ускорение:        1.00
Эффективность:    0.9968
Найдено совпадений: 1000 (Проверка: OK)
```

```
mxtv1x@WIN-EMVSNHTNF8D:/mnt/c/Users/User/MAI-LABS/OS-25/lab2$ ./main -t 2

Результаты эксперимента:
Потоков:          2
Время(сек):       0.067879
Ускорение:        1.97
Эффективность:    0.9871
Найдено совпадений: 1000 (Проверка: OK)
```

```
mxtv1x@WIN-EMVSNHTNF8D:/mnt/c/Users/User/MAI-LABS/OS-25/lab2$ ./main -t 4

Результаты эксперимента:
Потоков:          4
Время(сек):       0.042284
Ускорение:        3.47
Эффективность:    0.8687
Найдено совпадений: 1000 (Проверка: OK)
```

```
mxtv1x@WIN-EMVSNHTNF8D:/mnt/c/Users/User/MAI-LABS/OS-25/lab2$ ./main -t 8

Результаты эксперимента:
Потоков:          8
Время(сек):       0.034519
Ускорение:        3.86
Эффективность:    0.4821
Найдено совпадений: 1000 (Проверка: OK)
```

```
mxtv1x@WIN-EMVSNHTNF8D:/mnt/c/Users/User/MAI-LABS/OS-25/lab2$ ./main -t 12

Результаты эксперимента:
Потоков:          12
Время(сек):       0.029854
Ускорение:        4.44
Эффективность:    0.3702
Найдено совпадений: 1000 (Проверка: OK)
```

```
mxtv1x@WIN-EMVSNHTNF8D:/mnt/c/Users/User/MAI-LABS/OS-25/lab2$ ./main -t 16

Результаты эксперимента:
Потоков:          16
Время(сек):       0.030273
Ускорение:        4.76
Эффективность:    0.2978
Найдено совпадений: 1000 (Проверка: OK)
```

```
mxtv1x@WIN-EMVSNHTNF8D:/mnt/c/Users/User/MAI-LABS/OS-25/lab2$ ./main -t 100

Результаты эксперимента:
Потоков:          100
Время(сек):       0.031467
Ускорение:        4.22
Эффективность:    0.0422
Найдено совпадений: 1000 (Проверка: OK)
```

```
mxtv1x@WIN-EMVSNHTNF8D:/mnt/c/Users/User/MAI-LABS/OS-25/lab2$ ./main -t 1024

Результаты эксперимента:
Потоков:          1024
Время(сек):       0.038982
Ускорение:        3.42
Эффективность:    0.0033
Найдено совпадений: 1000 (Проверка: OK)
```

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	0.135213	1.00	1.00
2	0.067879	1.97	0.9871
4	0.042284	3.47	0.8687
8	0.034519	3.86	0.4821
12	0.029854	4.44	0.3702
16	0.030273	4.76	0.2978
100	0.031467	4.22	0.0422
1024	0.038982	3.42	0.0033

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 (1-4 потоков)

- Наблюдается почти линейное масштабирование (1.97× при 2 потоках)
- Эффективность остается высокой (0.86-0.99)
- Накладные расходы минимальны, потоки работают практически без конкуренции

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

2. Количество потоков РАВНО логическим ядрам процессора (12 потоков)

- Эффективность резко падает до 0.37
- 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

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