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-212БВ-24

Студент: Сучков В.С.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 29.11.25

Москва, 2025

Постановка задачи

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.

Вариант 18.

Найти образец в строке наивным алгоритмом.

Общий метод и алгоритм решения

Использованные функции из библиотеки “pthread.h”:

- pthread_create() - запускает новый поток с указанной функцией.
- pthread_join() - приостанавливает основной поток до завершения указанного рабочего потока.

Использованные формулы для расчётов:

Ускорение: S = Ts / Tp, где
Ts – время последовательной реализации,
Tp – время параллельной реализации,
1 <= S <= p,
p – количество ядер
Показывает, во сколько раз параллельная версия быстрее последовательной.

Эффективность: X = S / p, где X < 1
Показывает, насколько эффективно каждое ядро используется.

Алгоритм решения:

1. Инициализация входных данных.
2. Генерация тестовых данных.
3. Последовательный поиск.
4. Распределение работы для потоков.
5. Создание и запуск потоков.
6. Работа потоков (параллельно).
7. Ожидание завершения и сбор результатов.
8. Расчет метрик производительности.
9. Вывод результатов.

Код программы

main.c

```
#include <stdlib.h>

#include <unistd.h>

#include <pthread.h>
```

```
#include <string.h>
#include <stdio.h>
#include <time.h>
#include <sys/time.h>

typedef struct {
    int id;
    char *text;
    char *pattern;
    int patternLength;
    int startIndex;
    int endIndex;
    long long *localCount;
} ThreadArgs;

double getRealTime() {
    struct timeval tv;
    gettimeofday(&tv, NULL);
    return tv.tv_sec * 1000.0 + tv.tv_usec / 1000.0;
}

long long countPattern(const char *text, const char *pattern, int patternLen, int start, int end) {
    long long count = 0;

    for (int i = start; i <= end - patternLen; i++) {
        int j;
        for (j = 0; j < patternLen; j++) {
            if (text[i + j] != pattern[j]) {
                break;
            }
```

```
        }
        if (j == patternLen) {
            count++;
        }
    }
    return count;
}

// Функция потока для параллельной версии
void *parallelWork(void *_args) {
    ThreadArgs *args = (ThreadArgs *)_args;

    args->localCount[args->id] = countPattern(
        args->text, args->pattern, args->patternLength,
        args->startIndex, args->endIndex
    );

    return NULL;
}

long long sequentialSearch(char *text, char *pattern, int patternLength, int textLength) {
    return countPattern(text, pattern, patternLength, 0, textLength);
}

long long parallelSearch(char *text, char *pattern, int patternLength, int textLength, int threadsCount) {
    pthread_t *threads = (pthread_t *)malloc(threadsCount * sizeof(pthread_t));
    ThreadArgs *threadArgs = (ThreadArgs *)malloc(threadsCount * sizeof(ThreadArgs));
    long long *localCounts = (long long *)calloc(threadsCount, sizeof(long long));

    if (!threads || !threadArgs || !localCounts) {
```

```
    printf("Ошибка выделения памяти\n");
    free(threads);
    free(threadArgs);
    free(localCounts);
    return -1;
}

int range = textLength / threadsCount;
long long totalCount = 0;

for (int i = 0; i < threadsCount; i++) {
    threadArgs[i].id = i;
    threadArgs[i].text = text;
    threadArgs[i].pattern = pattern;
    threadArgs[i].patternLength = patternLength;
    threadArgs[i].localCount = localCounts;
    threadArgs[i].startIndex = i * range;
    if (i == threadsCount - 1) {
        threadArgs[i].endIndex = textLength;
    } else {
        threadArgs[i].endIndex = (i + 1) * range;
    }

    if (threadArgs[i].endIndex > textLength - patternLength) {
        threadArgs[i].endIndex = textLength - patternLength + 1;
    }
    if (threadArgs[i].startIndex > threadArgs[i].endIndex) {
        threadArgs[i].startIndex = threadArgs[i].endIndex;
    }
```

```
        if (pthread_create(&threads[i], NULL, parallelWork, &threadArgs[i]) != 0) {
            printf("Ошибка создания потока %d\n", i);
            threadsCount = i;
            break;
        }
    }

    for (int i = 0; i < threadsCount; i++) {
        pthread_join(threads[i], NULL);
        totalCount += localCounts[i];
    }

    free(threads);
    free(threadArgs);
    free(localCounts);

    return totalCount;
}

void generateTestData(char *text, char *pattern, int textLength, int patternLength) {
    const char charset[] =
"abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789 ";
    int charsetLen = strlen(charset);

    for (int i = 0; i < textLength; i++) {
        text[i] = charset[rand() % charsetLen];
    }
    text[textLength] = '\0';

    for (int i = 0; i < patternLength; i++) {
```

```
        pattern[i] = charset[rand() % charsetLen];
    }
    pattern[patternLength] = '\0';

    int oneToAdd = (textLength / 10000) + 1;
    for (int i = 0; i < oneToAdd; i++) {
        int guarantee = rand() % (textLength - patternLength);
        memcpy(&text[guarantee], pattern, patternLength);
    }
}

int main(int argc, char **argv) {
    if (argc != 4) {
        printf("Ошибка некорректного ввода\n");
        printf("Использование: %s <длина текста> <длина шаблона> <количество потоков>\n",
argv[0]);
        exit(EXIT_FAILURE);
    }

    int textLength = atoi(argv[1]);
    int patternLength = atoi(argv[2]);
    int threadsCount = atoi(argv[3]);

    if (patternLength > textLength || threadsCount <= 0 || textLength <= 0) {
        printf("Ошибка некорректных параметров\n");
        return 1;
    }

    char *text = (char *)malloc((textLength + 1) * sizeof(char));
    char *pattern = (char *)malloc((patternLength + 1) * sizeof(char));
```

```
if (!text || !pattern) {
    printf("Ошибка выделения памяти\n");
    return 1;
}

srand(time(NULL));

generateTestData(text, pattern, textLength, patternLength);

double startTime, endTime;

// Последовательная версия
startTime = getRealTime();
long long seqResult = sequentialSearch(text, pattern, patternLength, textLength);
endTime = getRealTime();
double timeResult1 = endTime - startTime;

// Параллельная версия
startTime = getRealTime();
long long parResult = parallelSearch(text, pattern, patternLength, textLength, threadsCount);
endTime = getRealTime();
double timeResult2 = endTime - startTime;

printf("Последовательное время: %.5f мс\n", timeResult1);
printf("Параллельное время: %.5f мс\n", timeResult2);

if (timeResult2 > 0) {
    double speedup = timeResult1 / timeResult2;
    double efficiency = (speedup / threadsCount) * 100;
```

```
        printf("Ускорение: %.2f\n", speedup);
        printf("Эффективность: %.2f%%\n", efficiency);
    }

    printf("Найдено вхождений (посл): %lld\n", seqResult);
    printf("Найдено вхождений (пар): %lld\n", parResult);

    free(text);
    free(pattern);

    return 0;
}
```

Протокол работы программы

```
goida@zov:~/OS/lab2$ gcc -o res main.c -pthread
goida@zov:~/OS/lab2$ ./res 1000000 100 2
Последовательное время: 3.46802 мс
Параллельное время: 2.68286 мс
Ускорение: 1.29
Эффективность: 64.63%
Найдено вхождений (посл): 99
Найдено вхождений (пар): 99
goida@zov:~/OS/lab2$ ./res 1000000 100 4
Последовательное время: 3.45093 мс
Параллельное время: 1.91089 мс
Ускорение: 1.81
Эффективность: 45.15%
Найдено вхождений (посл): 101
Найдено вхождений (пар): 101
goida@zov:~/OS/lab2$ ./res 1000000 100 6
Последовательное время: 4.30298 мс
Параллельное время: 1.42700 мс
Ускорение: 3.02
Эффективность: 50.26%
Найдено вхождений (посл): 99
Найдено вхождений (пар): 99
goida@zov:~/OS/lab2$ ./res 1000000 100 7
Последовательное время: 3.53589 мс
Параллельное время: 1.32910 мс
Ускорение: 2.66
Эффективность: 38.01%
Найдено вхождений (посл): 101
Найдено вхождений (пар): 101
goida@zov:~/OS/lab2$ ./res 1000000 100 8
Последовательное время: 3.46802 мс
Параллельное время: 1.27612 мс
Ускорение: 2.72
Эффективность: 33.97%
Найдено вхождений (посл): 99
Найдено вхождений (пар): 99
goida@zov:~/OS/lab2$ ./res 1000000 100 16
Последовательное время: 3.44800 мс
Параллельное время: 1.46606 мс
Ускорение: 2.35
Эффективность: 14.70%
Найдено вхождений (посл): 99
Найдено вхождений (пар): 99
goida@zov:~/OS/lab2$ ./res 1000000 100 128
Последовательное время: 3.45703 мс
Параллельное время: 7.11987 мс
Ускорение: 0.49
Эффективность: 0.38%
Найдено вхождений (посл): 101
Найдено вхождений (пар): 101
goida@zov:~/OS/lab2$ ./res 1000000 100 1024
Последовательное время: 4.01196 мс
Параллельное время: 56.71313 мс
Ускорение: 0.07
Эффективность: 0.01%
```

Таблица результатов:

Число потоков	Время выполнения (мс)	Ускорение	Эффективность
2	2.68	1.29	64.63%
4	1.91	1.81	45.15%
6	1.43	3.02	50.26%
7	1.33	2.66	38.01%
8	1.28	2.72	33.97%
16	1.47	2.35	14.70%
128	7.12	0.49	0.38%
1024	56.71	0.07	0.01%

Вывод

В процессе выполнения лабораторной работы я составил программу на языке Си, обрабатывающую данные в многопоточном режиме. В процессе изучения результатов я на практике убедился, что увеличение числа потоков уменьшает время выполнения программы, поскольку мы используем сразу несколько ядер процессора для вычислений. Однако, при увеличении числа потока выше количества физических ядер выполнение замедляется, так как операционная система вынуждена выполнять множество операций управления ресурсами, что негативно отражается на эффективности решения основной задачи.