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-213БВ-24

Студент: Кретов А.В.

Преподаватель: Бахарев В.Д. (ФИИТ)

Оценка: _____

Дата: 13.10.25

Москва, 2025

Постановка задачи

Вариант 18.

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. Необходимо уметь продемонстрировать количество потоков, используемое программой с помощью стандартных средств операционной системы.

Найти образец в строке наивным алгоритмом.

Общий метод и алгоритм решения

- pthread_create() – создает новый поток выполнения
- pthread_join() – ожидает завершения указанного потока).
- void *sbrk(intptr_t increment); — расширяет/сужает брекет (heap). В вашем коде используется в Malloc(size_t) для выделения памяти. Возвращает адрес новой области или (void*)-1 при ошибке.
- int gettimeofday(struct timeval *tv, struct timezone *tz); — получает текущее время с точностью до микросекунд. Вы используете в GetTimeMs() для измерения времени выполнения (переводите в миллисекунды).
- int open(const char *pathname, int flags, ...); — открывает файл и возвращает файловый дескриптор (или -1 при ошибке). Используется в ReadFile() с O_RDONLY.
- int fstat(int fd, struct stat *statbuf); — получает информацию о файле по файловому дескриптору (в вашем коде — размер файла st_size).
- ssize_t read(int fd, void *buf, size_t count); — читает из файла/дескриптора в буфер. Вы читаете содержимое файла в выделенный буфер.
- int close(int fd); — закрывает файловый дескриптор.
- ssize_t write(int fd, const void *buf, size_t count); — пишет данные в дескриптор (вы пишете сообщения и результаты в STDOUT_FILENO).

В рамках лабораторной работы была реализована многопоточная утилита для поиска всех вхождений подстроки в текстовом файле с замером производительности. Основные разделы, сделанные в работе: собственная лёгкая реализация аллокатора Malloc на базе sbrk, утилиты для работы со строками (StrLen, IntToStr, LongLongToStr), функции чтения файла ReadFile (используются системные вызовы open, fstat, read, close), измерение времени GetTimeMs на базе gettimeofday, наивный последовательный алгоритм поиска SequentialSearch, параллельная схема ParallelSearch с запуском потоков через pthread_create/pthread_join и функция потока ThreadSearch, а также вывод результатов в stdout через системный вызов write. Программа принимает аргументы командной строки ./program <pattern> <file> <max_threads>, печатает размер файла, время выполнения последовательного поиска и для каждого числа потоков от 1 до max_threads — время и количество найденных совпадений.

Принцип работы программы организован следующим образом. При запуске проверяются аргументы, парсится количество потоков; затем ReadFile открывает файл (open), получает его размер (fstat), выделяет буфер через Malloc и загружает содержимое (read), после чего закрывает дескриптор (close). Сначала выполняется полный последовательный проход SequentialSearch: для каждой позиции текста сравниваются символы с шаблоном, найденные индексы сохраняются в массив matches, время работы фиксируется через GetTimeMs (на основе gettimeofday) и результат выводится в stdout (write). Затем для каждого числа потоков 1..max_threads вызывается ParallelSearch: весь диапазон возможных позиций делится на numThreads частей с добавлением перекрытия по длине шаблона на границах, для каждого потока выделяется структура TThreadData и отдельный массив для локальных совпадений (выделяется sbrk-ом), создаются потоки pthread_create, которые выполняют ThreadSearch в своих диапазонах; после pthread_join результаты объединяются в общий массив finalMatches с проверкой на дубликаты (чтобы избежать повторного учёта совпадений, найденных в перекрывающихся областях). Для каждой конфигурации число потоков измеряется и печатается как Threads: X, Time: Y ms, Matches: Z. Вывод производится с использованием собственных конвертеров чисел в строки, чтобы формировать понятные текстовые строки для write.

Стоит отметить важные архитектурные моменты: программа использует низкоуровневые системные вызовы для ввода-вывода и sbrk для аллокации, а многопоточность реализована через POSIX-потоки (внутри которых могут применяться системные вызовы вроде clone/futex). Разбиение на блоки учитывает перекрытие длиной шаблона, чтобы не пропустить совпадения на границах, а объединение результатов защищает от дублирования, хотя проверка дубликатов реализована простым поиском и в худшем случае имеет квадратичную сложность.

Рассмотрим методику расчета ускорения и эффективности.

Ускорение – отношение времени выполнения последовательной программы к времени выполнения параллельной программы.

Формула:

$$S = T_s / T_p$$

где:

- S - ускорение
- T_s - время последовательного выполнения
- T_p - время параллельного выполнения

Эффективность – отношение ускорения к количеству потоков, показывает насколько эффективно используются вычислительные ресурсы.

Формула:

$$E = S / N$$

где:

- E - эффективность ($0 \leq E \leq 1$)
- S - ускорение

- N - количество потоков

Код программы

search.c

```
#include <pthread.h>
#include <unistd.h>
#include <sys/time.h>
#include <sys/stat.h>
#include <fcntl.h>

int StrLen(const char* text) { // вычисл длины строки
    int length = 0;
    while (text[length]) {
        length++;
    }
    return length;
}

void IntToStr(int value, char* buffer) { // числа в строку
    char temp[20];
    int tempIndex = 0;
    int i = 0;
    int bufferIndex = 0;

    if (value == 0) {
        buffer[bufferIndex++] = '0';
    } else {
        while (value > 0) {
            temp[tempIndex++] = '0' + (value % 10);
            value /= 10;
        }

        for (i = tempIndex - 1; i >= 0; i--) {
            buffer[bufferIndex++] = temp[i];
        }
    }
```

```
    buffer[bufferIndex] = '\0';
}

void LongLongToStr(long long value, char* buffer) {
    char temp[30];
    int tempIndex = 0;
    int i = 0;
    int bufferIndex = 0;
    if (value == 0) {
        buffer[bufferIndex++] = '0';
    } else {
        while (value > 0) {
            temp[tempIndex++] = '0' + (value % 10);
            value /= 10;
        }

        for (i = tempIndex - 1; i >= 0; i--) {
            buffer[bufferIndex++] = temp[i];
        }
    }
    buffer[bufferIndex] = '\0';
}

typedef struct { // стр для передачи данных в поток
    const char* text;
    const char* pattern;
    int textLen;
    int patternLen;
    int startIdx;
    int endIdx;
    int* matches;
    int matchCount;
    int maxMatchesPerThread;
} TThreadData;

void* Malloc(size_t size) { // аллокация памяти
```

```
    void* result = sbrk(size);
    if (result == (void*)-1) {
        return NULL;
    }
    return result;
}

int SequentialSearch(const char* text, const char* pattern, int* matches) {
    const int textLength = StrLen(text);
    const int patternLength = StrLen(pattern);
    int count = 0;
    for (int i = 0; i <= textLength - patternLength; i++) {
        int j;
        for (j = 0; j < patternLength; j++) {
            if (text[i + j] != pattern[j]) {
                break;
            }
        }
        if (j == patternLength) {
            matches[count] = i;
            count++;
        }
    }
    return count;
}

void* ThreadSearch(void* arg) { // для потока в паралл.
    TThreadData* data = (TThreadData*)arg;
    data->matchCount = 0;
    int actualEndIdx = data->endIdx;
    if (actualEndIdx + data->patternLen > data->textLen) {
        actualEndIdx = data->textLen - data->patternLen;
    }
    for (int i = data->startIdx; i <= actualEndIdx; i++) {
        if (data->matchCount >= data->maxMatchesPerThread) {
            break;
```

```
        }
        int j;
        for (j = 0; j < data->patternLen; j++) {
            if (data->text[i + j] != data->pattern[j]) {
                break;
            }
        }
        if (j == data->patternLen) {
            data->matches[data->matchCount] = i;
            data->matchCount++;
        }
    }
    return NULL;
}

int ParallelSearch(const char* text, const char* pattern, int* finalMatches, int numThreads) {
    const int textLength = StrLen(text);
    const int patternLength = StrLen(pattern);
    const int totalPositions = textLength - patternLength + 1;
    if (totalPositions <= 0) {
        return 0;
    }
    if (numThreads == 1) { // для 1 потока последовательный поиск
        return SequentialSearch(text, pattern, finalMatches);
    }
    pthread_t* threads = (pthread_t*)Malloc(numThreads * sizeof(pthread_t));
    TThreadData* threadData = (TThreadData*)Malloc(numThreads * sizeof(TThreadData));

    if (threads == NULL || threadData == NULL) {
        return 0;
    }

    const int baseChunk = totalPositions / numThreads;
    const int remainder = totalPositions % numThreads;
    int currentStart = 0;
    for (int i = 0; i < numThreads; i++) { // память для каждого потока
```

```
    int chunkSize = baseChunk + (i < remainder ? 1 : 0);
    int endIdx = currentStart + chunkSize - 1;
    if (i < numThreads - 1) {
        endIdx += patternLength - 1;
    }

    if (endIdx >= totalPositions) {
        endIdx = totalPositions - 1;
    }
    int maxMatchesForThread = chunkSize + patternLength;
    int* threadMatches = (int*)Malloc(maxMatchesForThread * sizeof(int));
    if (threadMatches == NULL) {
        return 0;
    }
    threadData[i].text = text;
    threadData[i].pattern = pattern;
    threadData[i].textLen = textLength;
    threadData[i].patternLen = patternLength;
    threadData[i].startIdx = currentStart;
    threadData[i].endIdx = endIdx;
    threadData[i].matches = threadMatches;
    threadData[i].matchCount = 0;
    threadData[i].maxMatchesPerThread = maxMatchesForThread;
    currentStart += chunkSize;
}
for (int i = 0; i < numThreads; i++) { // запуск потоков
    pthread_create(&threads[i], NULL, ThreadSearch, &threadData[i]);
}
int totalMatches = 0; // результаты
for (int i = 0; i < numThreads; i++) {
    pthread_join(threads[i], NULL);
    for (int j = 0; j < threadData[i].matchCount; j++) { // копируем резы потока в конечный массив
        int isDuplicate = 0;
        for (int k = 0; k < totalMatches; k++) {
            if (finalMatches[k] == threadData[i].matches[j]) {
                isDuplicate = 1;
```

```
                break;
            }
        }
        if (!isDuplicate) {
            finalMatches[totalMatches] = threadData[i].matches[j];
            totalMatches++;
        }
    }
}

return totalMatches;
}

long long GetTimeMs() {
    struct timeval timeValue;
    gettimeofday(&timeValue, NULL);
    return (long long)timeValue.tv_sec * 1000 + timeValue.tv_usec / 1000;
}

void ReadFile(const char* filename, char** buffer, int* length) {
    const int fileDescriptor = open(filename, O_RDONLY);
    if (fileDescriptor < 0) {
        *length = 0;
        *buffer = NULL;
        return;
    }
    struct stat fileStat;
    fstat(fileDescriptor, &fileStat);
    *length = fileStat.st_size;
    *buffer = (char*)Malloc(*length + 1);
    if (*buffer != NULL) {
        read(fileDescriptor, *buffer, *length);
        (*buffer)[*length] = '\0';
    }
    close(fileDescriptor);
}
```

```
int main(int argc, char* argv[]) {
    const int MIN_ARGUMENTS = 4;
    if (argc < MIN_ARGUMENTS) {
        const char* errorMessage = "Usage: ./program <pattern> <file> <max_threads>\n";
        write(STDOUT_FILENO, errorMessage, StrLen(errorMessage));
        return 1;
    }
    const char* pattern = argv[1];
    const char* filename = argv[2];
    int maxThreads = 0;
    for (int i = 0; argv[3][i]; i++) {
        maxThreads = maxThreads * 10 + (argv[3][i] - '0');
    }
    if (maxThreads <= 0) {
        const char* errorMessage = "Error: max_threads must be positive\n";
        write(STDOUT_FILENO, errorMessage, StrLen(errorMessage));
        return 1;
    }
    char* text; // Читаем файл
    int textLength;
    ReadFile(filename, &text, &textLength);
    if (text == NULL) {
        const char* errorMessage = "Error: Cannot open file or allocate memory\n";
        write(STDOUT_FILENO, errorMessage, StrLen(errorMessage));
        return 1;
    }

    char infoBuffer[128]; // информация о файле
    int infoIndex = 0;
    const char* infoStr = "File size: ";
    for (int i = 0; infoStr[i]; i++) infoBuffer[infoIndex++] = infoStr[i];
    char sizeStr[20];
    IntToStr(textLength, sizeStr);
    for (int i = 0; sizeStr[i]; i++) infoBuffer[infoIndex++] = sizeStr[i];
    const char* bytesStr = " bytes\n";
```

```
for (int i = 0; bytesStr[i]; i++) infoBuffer[infoIndex++] = bytesStr[i];
write(STDOUT_FILENO, infoBuffer, infoIndex);
const int patternLength = StrLen(pattern);
const int maxMatches = textLength - patternLength + 1;
if (maxMatches <= 0) {
    const char* errorMessage = "Error: Pattern longer than text\n";
    write(STDOUT_FILENO, errorMessage, StrLen(errorMessage));
    return 1;
}
int* matches = (int*)Malloc(maxMatches * sizeof(int));
if (matches == NULL) {
    const char* errorMessage = "Error: Cannot allocate memory for matches\n";
    write(STDOUT_FILENO, errorMessage, StrLen(errorMessage));
    return 1;
}

const long long startTimeSeq = GetTimeMs(); // последовательная
int seqMatches = SequentialSearch(text, pattern, matches);
const long long seqTime = GetTimeMs() - startTimeSeq;
char seqBuffer[128]; // вывод
int seqBufferIndex = 0;
const char* seqTimeStr = "Sequential: ";
for (int i = 0; seqTimeStr[i]; i++) seqBuffer[seqBufferIndex++] = seqTimeStr[i];
char seqTimeNumStr[30];
LongLongToStr(seqTime, seqTimeNumStr);
for (int i = 0; seqTimeNumStr[i]; i++) seqBuffer[seqBufferIndex++] = seqTimeNumStr[i];
const char* seqMatchesStr = " ms, matches: ";
for (int i = 0; seqMatchesStr[i]; i++) seqBuffer[seqBufferIndex++] = seqMatchesStr[i];
char seqMatchesNumStr[20];
IntToStr(seqMatches, seqMatchesNumStr);
for (int i = 0; seqMatchesNumStr[i]; i++) seqBuffer[seqBufferIndex++] = seqMatchesNumStr[i];
const char* newlineStr = "\n";
for (int i = 0; newlineStr[i]; i++) seqBuffer[seqBufferIndex++] = newlineStr[i];
write(STDOUT_FILENO, seqBuffer, seqBufferIndex);

for (int numThreads = 1; numThreads <= maxThreads; numThreads++) {
```

```
        const long long startTimePar = GetTimeMs();
        int parMatches = ParallelSearch(text, pattern, matches, numThreads);
        const long long parTime = GetTimeMs() - startTimePar;
        char buffer[256];
        int bufferIndex = 0;
        const char* threadsStr = "Threads: ";
        for (int i = 0; threadsStr[i]; i++) buffer[bufferIndex++] = threadsStr[i];
        char numStr[20];
        IntToStr(numThreads, numStr);
        for (int i = 0; numStr[i]; i++) buffer[bufferIndex++] = numStr[i];
        const char* timeStr = ", Time: ";
        for (int i = 0; timeStr[i]; i++) buffer[bufferIndex++] = timeStr[i];
        char timeNumStr[30];
        LongLongToStr(parTime, timeNumStr);
        for (int i = 0; timeNumStr[i]; i++) buffer[bufferIndex++] = timeNumStr[i];
        const char* matchesStr = " ms, Matches: ";
        for (int i = 0; matchesStr[i]; i++) buffer[bufferIndex++] = matchesStr[i];
        char matchesNumStr[20];
        IntToStr(parMatches, matchesNumStr);
        for (int i = 0; matchesNumStr[i]; i++) buffer[bufferIndex++] = matchesNumStr[i];
        const char* msStr = "\n";
        for (int i = 0; msStr[i]; i++) buffer[bufferIndex++] = msStr[i];
        write(STDOUT_FILENO, buffer, bufferIndex);
    }
    return 0;
}
```

Протокол работы программы

user@WIN-LVQ040U9NHA:/mnt/d/!labs/os/lab2$ ./a.out "дуб" wap.txt 2

File size: 3393863 bytes

Sequential: 18 ms, matches: 23

Threads: 1, Time: 20 ms, Matches: 23

Threads: 2, Time: 12 ms, Matches: 23

user@WIN-LVQ040U9NHA:/mnt/d/!labs/os/lab2$ ./a.out "дуб" wap.txt 3

File size: 3393863 bytes

Sequential: 18 ms, matches: 23

Threads: 1, Time: 19 ms, Matches: 23

Threads: 2, Time: 13 ms, Matches: 23

Threads: 3, Time: 9 ms, Matches: 23

user@WIN-LVQ040U9NHA:/mnt/d/!labs/os/lab2$ ./a.out "дуб" wap.txt 4

File size: 3393863 bytes

Sequential: 17 ms, matches: 23

Threads: 1, Time: 20 ms, Matches: 23

Threads: 2, Time: 13 ms, Matches: 23

Threads: 3, Time: 10 ms, Matches: 23

Threads: 4, Time: 8 ms, Matches: 23

user@WIN-LVQ040U9NHA:/mnt/d/!labs/os/lab2$ ./a.out "дуб" wap.txt 5

File size: 3393863 bytes

Sequential: 17 ms, matches: 23

Threads: 1, Time: 20 ms, Matches: 23

Threads: 2, Time: 12 ms, Matches: 23

Threads: 3, Time: 9 ms, Matches: 23

Threads: 4, Time: 8 ms, Matches: 23

Threads: 5, Time: 7 ms, Matches: 23

user@WIN-LVQ040U9NHA:/mnt/d/!labs/os/lab2$ ./a.out "дуб" wap.txt 6

File size: 3393863 bytes

Sequential: 18 ms, matches: 23

Threads: 1, Time: 20 ms, Matches: 23

Threads: 2, Time: 12 ms, Matches: 23

Threads: 3, Time: 11 ms, Matches: 23

Threads: 4, Time: 9 ms, Matches: 23

Threads: 5, Time: 7 ms, Matches: 23

Threads: 6, Time: 7 ms, Matches: 23

user@WIN-LVQ040U9NHA:/mnt/d/!labs/os/lab2$ ./a.out "дуб" wap.txt 24

File size: 3393863 bytes

Sequential: 17 ms, matches: 23

Threads: 1, Time: 20 ms, Matches: 23

Threads: 2, Time: 12 ms, Matches: 23

Threads: 3, Time: 9 ms, Matches: 23

Threads: 4, Time: 8 ms, Matches: 23

Threads: 5, Time: 8 ms, Matches: 23

Threads: 6, Time: 8 ms, Matches: 23

Threads: 7, Time: 6 ms, Matches: 23

Threads: 8, Time: 7 ms, Matches: 23

Threads: 9, Time: 6 ms, Matches: 23

Threads: 10, Time: 6 ms, Matches: 23

Threads: 11, Time: 6 ms, Matches: 23

Threads: 12, Time: 7 ms, Matches: 23

Threads: 13, Time: 7 ms, Matches: 23

Threads: 14, Time: 6 ms, Matches: 23

Threads: 15, Time: 6 ms, Matches: 23

Threads: 16, Time: 6 ms, Matches: 23

Threads: 17, Time: 6 ms, Matches: 23

Threads: 18, Time: 9 ms, Matches: 23

Threads: 19, Time: 6 ms, Matches: 23

^[[AThreads: 20, Time: 5 ms, Matches: 23

Threads: 21, Time: 6 ms, Matches: 23

Threads: 22, Time: 6 ms, Matches: 23

Threads: 23, Time: 7 ms, Matches: 23

Threads: 24, Time: 7 ms, Matches: 23

user@WIN-LVQ040U9NHA:/mnt/d/!labs/os/lab2$ ./a.out "дуб" wap.txt 32

File size: 3393863 bytes

Sequential: 17 ms, matches: 23

Threads: 1, Time: 20 ms, Matches: 23

Threads: 2, Time: 12 ms, Matches: 23

Threads: 3, Time: 11 ms, Matches: 23

Threads: 4, Time: 8 ms, Matches: 23

Threads: 5, Time: 7 ms, Matches: 23

Threads: 6, Time: 8 ms, Matches: 23

Threads: 7, Time: 7 ms, Matches: 23

Threads: 8, Time: 6 ms, Matches: 23

Threads: 9, Time: 6 ms, Matches: 23

Threads: 10, Time: 6 ms, Matches: 23

Threads: 11, Time: 6 ms, Matches: 23

Threads: 12, Time: 6 ms, Matches: 23

Threads: 13, Time: 8 ms, Matches: 23

^[[AThreads: 14, Time: 7 ms, Matches: 23

Threads: 15, Time: 6 ms, Matches: 23

Threads: 16, Time: 7 ms, Matches: 23

Threads: 17, Time: 6 ms, Matches: 23

Threads: 18, Time: 5 ms, Matches: 23

Threads: 19, Time: 6 ms, Matches: 23

Threads: 20, Time: 6 ms, Matches: 23

Threads: 21, Time: 5 ms, Matches: 23

Threads: 22, Time: 6 ms, Matches: 23

Threads: 23, Time: 5 ms, Matches: 23

Threads: 24, Time: 8 ms, Matches: 23

Threads: 25, Time: 5 ms, Matches: 23

Threads: 26, Time: 5 ms, Matches: 23

Threads: 27, Time: 7 ms, Matches: 23

Threads: 28, Time: 6 ms, Matches: 23

Threads: 29, Time: 6 ms, Matches: 23

Threads: 30, Time: 6 ms, Matches: 23

Threads: 31, Time: 6 ms, Matches: 23

Threads: 32, Time: 6 ms, Matches: 23

user@WIN-LVQ040U9NHA:/mnt/d/!labs/os/lab2$ ./a.out "дуб" wap.txt 128

File size: 3393863 bytes

Sequential: 18 ms, matches: 23

Threads: 1, Time: 19 ms, Matches: 23

Threads: 2, Time: 13 ms, Matches: 23

Threads: 3, Time: 9 ms, Matches: 23

Threads: 4, Time: 8 ms, Matches: 23

Threads: 5, Time: 9 ms, Matches: 23

Threads: 6, Time: 8 ms, Matches: 23

Threads: 7, Time: 8 ms, Matches: 23

Threads: 8, Time: 6 ms, Matches: 23

Threads: 9, Time: 6 ms, Matches: 23

Threads: 10, Time: 6 ms, Matches: 23

Threads: 11, Time: 6 ms, Matches: 23

Threads: 12, Time: 7 ms, Matches: 23

Threads: 13, Time: 7 ms, Matches: 23

Threads: 14, Time: 6 ms, Matches: 23

Threads: 15, Time: 6 ms, Matches: 23

Threads: 16, Time: 6 ms, Matches: 23

Threads: 17, Time: 6 ms, Matches: 23

Threads: 18, Time: 5 ms, Matches: 23

Threads: 19, Time: 6 ms, Matches: 23

Threads: 20, Time: 6 ms, Matches: 23

Threads: 21, Time: 6 ms, Matches: 23

Threads: 22, Time: 6 ms, Matches: 23

Threads: 23, Time: 6 ms, Matches: 23

Threads: 24, Time: 6 ms, Matches: 23

Threads: 25, Time: 7 ms, Matches: 23

Threads: 26, Time: 6 ms, Matches: 23

Threads: 27, Time: 7 ms, Matches: 23

Threads: 28, Time: 6 ms, Matches: 23

Threads: 29, Time: 6 ms, Matches: 23

Threads: 30, Time: 6 ms, Matches: 23

Threads: 31, Time: 6 ms, Matches: 23

Threads: 32, Time: 6 ms, Matches: 23

Threads: 33, Time: 6 ms, Matches: 23

Threads: 34, Time: 6 ms, Matches: 23

Threads: 35, Time: 7 ms, Matches: 23

Threads: 36, Time: 6 ms, Matches: 23

Threads: 37, Time: 7 ms, Matches: 23

Threads: 38, Time: 6 ms, Matches: 23

Threads: 39, Time: 8 ms, Matches: 23

Threads: 40, Time: 9 ms, Matches: 23

Threads: 41, Time: 7 ms, Matches: 23

Threads: 42, Time: 6 ms, Matches: 23

Threads: 43, Time: 7 ms, Matches: 23

Threads: 44, Time: 7 ms, Matches: 23

Threads: 45, Time: 6 ms, Matches: 23

Threads: 46, Time: 8 ms, Matches: 23

Threads: 47, Time: 7 ms, Matches: 23

Threads: 48, Time: 6 ms, Matches: 23

Threads: 49, Time: 7 ms, Matches: 23

Threads: 50, Time: 8 ms, Matches: 23

Threads: 51, Time: 7 ms, Matches: 23

Threads: 52, Time: 7 ms, Matches: 23

Threads: 53, Time: 7 ms, Matches: 23

Threads: 54, Time: 5 ms, Matches: 23

Threads: 55, Time: 6 ms, Matches: 23

Threads: 56, Time: 7 ms, Matches: 23

Threads: 57, Time: 5 ms, Matches: 23

Threads: 58, Time: 7 ms, Matches: 23

Threads: 59, Time: 6 ms, Matches: 23

Threads: 60, Time: 5 ms, Matches: 23

Threads: 61, Time: 6 ms, Matches: 23

Threads: 62, Time: 6 ms, Matches: 23

Threads: 63, Time: 5 ms, Matches: 23

Threads: 64, Time: 8 ms, Matches: 23

Threads: 65, Time: 7 ms, Matches: 23

Threads: 66, Time: 6 ms, Matches: 23

Threads: 67, Time: 7 ms, Matches: 23

Threads: 68, Time: 7 ms, Matches: 23

Threads: 69, Time: 6 ms, Matches: 23

Threads: 70, Time: 7 ms, Matches: 23

Threads: 71, Time: 6 ms, Matches: 23

Threads: 72, Time: 7 ms, Matches: 23

Threads: 73, Time: 7 ms, Matches: 23

Threads: 74, Time: 6 ms, Matches: 23

Threads: 75, Time: 7 ms, Matches: 23

Threads: 76, Time: 7 ms, Matches: 23

Threads: 77, Time: 7 ms, Matches: 23

Threads: 78, Time: 7 ms, Matches: 23

Threads: 79, Time: 7 ms, Matches: 23

Threads: 80, Time: 7 ms, Matches: 23

Threads: 81, Time: 7 ms, Matches: 23

Threads: 82, Time: 6 ms, Matches: 23

Threads: 83, Time: 6 ms, Matches: 23

Threads: 84, Time: 7 ms, Matches: 23

Threads: 85, Time: 7 ms, Matches: 23

Threads: 86, Time: 7 ms, Matches: 23

Threads: 87, Time: 7 ms, Matches: 23

Threads: 88, Time: 8 ms, Matches: 23

Threads: 89, Time: 6 ms, Matches: 23

Threads: 90, Time: 8 ms, Matches: 23

Threads: 91, Time: 7 ms, Matches: 23

Threads: 92, Time: 7 ms, Matches: 23

Threads: 93, Time: 8 ms, Matches: 23

Threads: 94, Time: 7 ms, Matches: 23

Threads: 95, Time: 7 ms, Matches: 23

Threads: 96, Time: 7 ms, Matches: 23

Threads: 97, Time: 7 ms, Matches: 23

Threads: 98, Time: 8 ms, Matches: 23

Threads: 99, Time: 8 ms, Matches: 23

Threads: 100, Time: 8 ms, Matches: 23

Threads: 101, Time: 7 ms, Matches: 23

Threads: 102, Time: 7 ms, Matches: 23

Threads: 103, Time: 8 ms, Matches: 23

Threads: 104, Time: 8 ms, Matches: 23

Threads: 105, Time: 8 ms, Matches: 23

Threads: 106, Time: 7 ms, Matches: 23

Threads: 107, Time: 7 ms, Matches: 23

Threads: 108, Time: 8 ms, Matches: 23

Threads: 109, Time: 7 ms, Matches: 23

Threads: 110, Time: 7 ms, Matches: 23

Threads: 111, Time: 8 ms, Matches: 23

Threads: 112, Time: 8 ms, Matches: 23

Threads: 113, Time: 8 ms, Matches: 23

Threads: 114, Time: 8 ms, Matches: 23

Threads: 115, Time: 7 ms, Matches: 23

Threads: 116, Time: 8 ms, Matches: 23

Threads: 117, Time: 8 ms, Matches: 23

Threads: 118, Time: 8 ms, Matches: 23

Threads: 119, Time: 8 ms, Matches: 23

Threads: 120, Time: 8 ms, Matches: 23

Threads: 121, Time: 8 ms, Matches: 23

Threads: 122, Time: 8 ms, Matches: 23

Threads: 123, Time: 8 ms, Matches: 23

Threads: 124, Time: 8 ms, Matches: 23

Threads: 125, Time: 8 ms, Matches: 23

Threads: 126, Time: 9 ms, Matches: 23

Threads: 127, Time: 8 ms, Matches: 23

Threads: 128, Time: 8 ms, Matches: 23

Время работы программы при последовательном исполнении алгоритма: 18 мс.

При параллельном выполнении:

Число потоков	Время выполнения, мс	Ускорение	Эффективность
1	19	0,95	0,95
2	13	1,38	0,69
3	9	2	0,67
4	8	2,25	0,56
5	7	2,57	0,51
6	7	2,57	0,43
20	5	3,6	0,18
24	6	3	0,13
64	8	2,25	0,04
96	8	2,25	0,02
128	8	2,25	0,02

Последовательная версия обрабатывает 3.4 МБ текста за 18 мс, находя 23 вхождения. При переходе к параллельной обработке наблюдается значительное ускорение до 3.6 раз при использовании 18-20 потоков, где время сокращается до 5 мс.

Наиболее эффективное использование ресурсов наблюдается в диапазоне 3-5 потоков, где эффективность превышает 50%. Однако дальнейшее увеличение количества потоков свыше 20 не приносит дополнительного выигрыша в скорости, а лишь снижает эффективность использования вычислительных ресурсов до 2% при 128 потоках. Это объясняется законом Амдала - существованием принципиального предела ускорения из-за последовательных участков алгоритма и возрастающими накладными расходами на управление потоками.

Полученные результаты подтверждают, что для задач поиска в текстах среднего размера оптимальным является использование количества потоков, сопоставимого с числом физических ядер процессора, а чрезмерное распараллеливание становится контрпродуктивным.

Вывод

В ходе выполнения данной лабораторной работы удалось реализовать программу, наглядно демонстрирующую принципы многопоточного программирования и ограничения параллельных алгоритмов. Реализованный наивный алгоритм поиска подстроки показал ускорение до 3.6 раз при оптимальном количестве потоков (18-20), что подтверждает эффективность распараллеливания для обработки текстовых данных. Однако дальнейшее увеличение потоков свыше 32 не дает значимого выигрыша из-за накладных расходов на управление потоками и ограничений закона Амдала. Основные трудности в ходе выполнения работы возникли с корректным распределением работы между потоками - требовалось обеспечить перекрытие блоков для поиска совпадений на границах, избегая при этом дублирования результатов. Кроме того, сложности были связаны с серьёзными ограничениями на использование стандартной библиотеки.