Правительство Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»
(НИУ ВШЭ)

Московский институт электроники и математики им. А.Н. Тихонова

ОТЧЕТ
О ПРАКТИЧЕСКОЙ РАБОТЕ № 6
по дисциплине «Системное программирование»

# ПОТОКИ

Студент гр. БИБ 211
_____ В.А. Санников
«___» _____ 2024 г.

Руководитель
Старший преподаватель
_____ В.И. Морозов
«___» _____ 2024 г.

Москва

**СОДЕРЖАНИЕ**

## 1 Задание на практическую работу

Целью работы является изучение языка С, а также написание двух программных реализаций для ОС Windows и ОС Linux, которые удовлетворяют следующему заданию: “В файле записана строка. Программа должна считать имя файла из первого аргумента командной строки и посчитать, сколько строчных (не заглавных) букв содержится в строке. Для расчёта программа должна создать N дочерних потоков (N передаётся вторым аргументом командной строки) и передать каждому из них часть строки. Каждый из дочерних потоков должен рассчитать количество строчных (не заглавных) букв в своей части строки и прибавить его к значению разделяемой переменной. Родительский поток должен получить значение разделяемой переменной и вывести его в консоль. Если исходный файл не существует, или в нём записано менее 2 символов, следует вывести соответствующее сообщение для пользователя и завершить работу программы.”

Для выполнения работы необходимо:

С помощью языка программирования Си или C++ необходимо написать две программы с идентичным функционалом, решающих задачу из варианта, – одну для ОС Windows и одну для ОС Linux. Для выполнения задания может использоваться любая среда разработки. При выполнении необходимо учитывать следующие требования:

а) Задача считается решённой, если необходимое по варианту действие выполняется для всех допустимых входных данных.

б) Для работы с потоками (запуска, ожидания, отсоединения, завершения) следует использовать только предназначенные для этого функции из API соответствующих ОС. Внимание! Не используйте return для завершения потоков в данной работе. Однако для файлового и консольного ввода-вывода можно воспользоваться любыми доступными средствами.

в) Программа должна осуществлять параллельное выполнение расчётов, т.е. в определённый момент времени работы программы в ней должны одновременно работать несколько потоков, каждый из которых выполняет свою часть задачи.

г) Доступ ко всем (не только к тем, которые заданы искусственно для выполнения работы) разделяемым ресурсам должен быть синхронизирован с помощью соответствующих средств синхронизации, при этом для каждого ресурса должно

использоваться отдельный экземпляр (например, по одному мьютексу на каждый ресурс).

д) Программа должна корректно завершаться, не вызывая аварийный останов.

е) Программа должна брать входные данные из аргументов, переданных при запуске в консоли. В случае, если количество переданных аргументов не равно ожидаемому, программа должна вывести подсказку для пользователя, поясняющую правила её (программы) использования.

ж) Возвращаемые значения всех вызываемых функций должны проверяться на предмет возникновения ошибок. В случае возникновения ошибки необходимо вывести сообщение, оповещающее пользователя о произошедшем, содержащее в обязательном порядке код ошибки и её текстовое описание (предоставленное ОС). В случае, если в результате возникшей ошибки программа должна быть завершена, перед завершением необходимо освободить все занятые ресурсы (очистить выделенную память, закрыть открытые файлы).

з) Программа должна работать с любым количеством данных без необходимости внесения изменений в код и перекомпиляции. Если количество потоков, которые необходимо породить, больше количества входных данных, необходимо запустить кол-во потоков, вдвое меньшее кол-ва входных данных и вывести соответствующее предупреждение.

Если количество символов строки входного файла не делится нацело на количество потоков, числа должны распределяться между потоками следующим образом:

Пусть кол-во входных данных равно M, кол-во потоков равно N, тогда первым N - 1 потокам должны быть назначены фрагменты входных данных длины M/N с округлением вниз, а последний поток должен обработать M-(N-1)*(M/N с округлением вниз) символов. При этом если N больше чем M/2 то нужно уменьшить количество потоков до M/2 с округлением вниз и вывести соответствующее уведомление для пользователя. То есть обобщая программа должна делить информацию из входного текстового файла на сегменты по правилам описанными ранее и выдать каждому потоку его сегмент информации.

## 2 Ход работы

Для выполнения данной лабораторной работы необходимо выбрать задание по варианту. Мой номер в группе = 19, всего дано 15 вариантов, соответственно, рассмотрим вариант под номером 4:

“В файле записана строка. Программа должна считать имя файла из первого аргумента командной строки и посчитать, сколько строчных (не заглавных) букв содержится в строке. Для расчёта программа должна создать N дочерних потоков (N передаётся вторым аргументом командной строки) и передать каждому из них часть строки. Каждый из дочерних потоков должен рассчитать количество строчных (не заглавных) букв в своей части строки и прибавить его к значению разделяемой переменной. Родительский поток должен получить значение разделяемой переменной и вывести его в консоль. Если исходный файл не существует, или в нём записано менее 2 символов, следует вывести соответствующее сообщение для пользователя и завершить работу программы.”

### 2.1 Программная реализация на языке C для ОС Windows

```
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>
#include <string.h>
#include <stdbool.h>
CRITICAL_SECTION cs;
int sharedCounter = 0;
typedef struct {
    char* data;
    int start;
    int end;
} ThreadData;
void errorExit(const char* message) {
    LPSTR errorText = NULL;
    FormatMessageA(FORMAT_MESSAGE_ALLOCATE_BUFFER |
FORMAT_MESSAGE_FROM_SYSTEM | FORMAT_MESSAGE_IGNORE_INSERTS,
```

```
            NULL, GetLastError(), MAKELANGID(LANG_NEUTRAL,
SUBLANG_DEFAULT), (LPSTR)&errorText, 0, NULL);
    fprintf(stderr, "%s: %s\n", message, errorText);
    LocalFree(errorText);
    ExitProcess(1);
}
DWORD WINAPI countLowerCase(void* param) {
    ThreadData* tData = (ThreadData*)param;
    int localCounter = 0;
    for (int i = tData->start; i < tData->end; i++) {
        if (islower(tData->data[i])) {
            localCounter++;
        }
    }
    EnterCriticalSection(&cs);
    sharedCounter += localCounter;
    LeaveCriticalSection(&cs);
    printf("Thread %d processed: %.*s\n", GetCurrentThreadId(), tData->end - tData->start,
tData->data + tData->start);
    printf("Thread %d counted lowercase letters: %d\n", GetCurrentThreadId(), localCounter);
    Sleep(30000); // Пауза на 30 секунд
    ExitThread(0);
}
int main(int argc, char* argv[]) {
    if (argc != 3) {
        fprintf(stderr, "Usage: %s <input_file> <number_of_threads>\n", argv[0]);
        ExitProcess(1);
    }
    InitializeCriticalSection(&cs);
    char* filename = argv[1];
    int numThreads = atoi(argv[2]);
    FILE* file = fopen(filename, "r");
    if (file == NULL) {
        errorExit("Error opening file");
```

```
}
fseek(file, 0, SEEK_END);
long fileSize = ftell(file);
rewind(file);
if (fileSize < 2) {
    fprintf(stderr, "File should contain at least 2 characters\n");
    fclose(file);
    ExitProcess(1);
}
int maxThreads = fileSize / 2;
if (numThreads > maxThreads) {
    numThreads = maxThreads;
    printf("Number of threads reduced to %d to meet the requirement.\n", numThreads);
}
char* fileContent = (char*)malloc(fileSize + 1);
if (fileContent == NULL) {
    errorExit("Error allocating memory for file content");
}
if (fread(fileContent, 1, fileSize, file) != fileSize) {
    free(fileContent);
    fclose(file);
    errorExit("Error reading file content");
}
fileContent[fileSize] = '\0';
ThreadData* threadsData = (ThreadData*)malloc(numThreads * sizeof(ThreadData));
if (threadsData == NULL) {
    free(fileContent);
    fclose(file);
    errorExit("Error allocating memory for threads data");
}
HANDLE* threadHandles = (HANDLE*)malloc(numThreads * sizeof(HANDLE));
if (threadHandles == NULL) {
    free(fileContent);
    free(threadsData);
```

```
        fclose(file);
        errorExit("Error allocating memory for thread handles");
    }
    int segmentSize = fileSize / numThreads;
    int remainingChars = fileSize % numThreads;
    int currentStart = 0;
    for (int i = 0; i < numThreads; i++) {
        threadsData[i].data = fileContent;
        threadsData[i].start = currentStart;
        threadsData[i].end = currentStart + segmentSize + (i < remainingChars ? 1 : 0);
        currentStart = threadsData[i].end;
    }
    for (int i = 0; i < numThreads; i++) {
        threadHandles[i] = CreateThread(NULL, 0, countLowerCase, &threadsData[i], 0,
NULL);
        if (threadHandles[i] == NULL) {
            free(fileContent);
            free(threadsData);
            fclose(file);
            errorExit("Error creating thread");
        }
    }
    WaitForMultipleObjects(numThreads, threadHandles, TRUE, INFINITE);
    fclose(file);
    free(fileContent);
    free(threadsData);
    free(threadHandles);
    DeleteCriticalSection(&cs);
    printf("Total lowercase letters in the file: %d\n", sharedCounter);
    ExitProcess(0);
}
```

Листинг 1 – Программная реализация для ОС Windows.

Давай разберем код по частям.

Объявление библиотек и глобальных переменных:

- В начале программы подключаются заголовочные файлы для работы с вводом/выводом, динамической памятью, многопоточностью (windows.h), строками и логическими значениями (stdbool.h).
- CRITICAL_SECTION — это механизм синхронизации в Windows для защиты общих данных при доступе из нескольких потоков. В данном случае, CriticalSection используется для безопасного доступа к sharedCounter, который подсчитывает количество строчных букв в тексте.
- sharedCounter хранит общее количество строчных букв в тексте, которое увеличивается каждым потоком при обработке.

Структура ThreadData и функция errorExit

- ThreadData — это структура для передачи данных в поток. Она содержит указатель на данные, начало и конец сегмента файла, который поток должен обработать.
- errorExit — это функция для вывода сообщения об ошибке, включая описание системной ошибки, выход из программы с кодом 1.

Функция обработки в потоках countLowerCase:

- countLowerCase — функция, которая обрабатывает сегмент файла, подсчитывая количество строчных букв в заданном диапазоне символов.
- islower используется для проверки, является ли символ строчной буквой.
- После подсчёта локального количества строчных букв, поток защищает доступ к sharedCounter с помощью EnterCriticalSection и LeaveCriticalSection.
- Функция выводит информацию о выполненной обработке и о количестве найденных строчных букв.
- Приостановка потока на 30 секунд реализована с помощью Sleep.
- По завершении работы поток вызывает ExitThread.

Функция main и обработка файла:

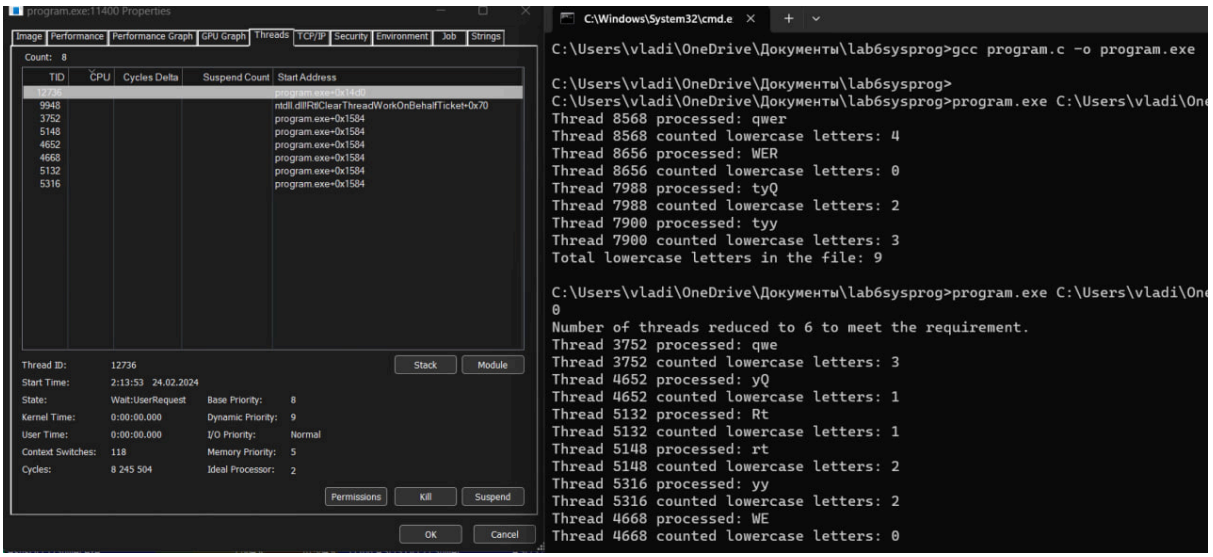
- В main сначала проверяются аргументы командной строки: необходимо передать путь к файлу и количество потоков.
- Затем инициализируется критическая секция.
- Далее начинается обработка файла: чтение файла, подготовка данных для каждого потока, создание потоков и ожидание завершения их работы.
- По завершении работы потоков закрываются файл, освобождаются ресурсы памяти и обработчики потоков.

- Удаляется критическая секция.
- На экран выводится общее количество найденных строчных букв в файле.

Особенности:

- Программа распределяет обработку файла между потоками для эффективности.
- Используется механизм критической секции для безопасного обновления общего счётчика sharedCounter.
- В коде есть обработка ошибок, включая вывод сообщения об ошибке с описанием системной ошибки.
- После обработки каждый поток ожидает 30 секунд перед выходом.

Этот код позволяет эффективно подсчитывать количество строчных букв в больших файлах, используя многопоточность для ускорения процесса.




Скриншот 1 – Сборка программ в исполняемые файлы и тестирование программы в ОС Windows.

### 2.2 Программная реализация на языке C для ОС Linux

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <pthread.h>
#include <unistd.h> // For usleep
#include <stdbool.h>
#include <errno.h> // Include errno.h for errno variable
#include <ctype.h> // Include ctype.h for islower
pthread_mutex_t mutex;
int sharedCounter = 0;
typedef struct {
    char* data;
    int start;
    int end;
} ThreadData;
void errorExit(const char* message) {
    fprintf(stderr, "%s error: %s\n", message, strerror(errno));
    exit(EXIT_FAILURE);
}
void* countLowerCase(void* param) {
    ThreadData* tData = (ThreadData*) param;
    int localCounter = 0;

    for (int i = tData->start; i < tData->end; i++) {
        if (islower(tData->data[i])) {
            localCounter++;
        }
    }
    if (pthread_mutex_lock(&mutex) != 0) {
        errorExit("Mutex lock");
    }
    sharedCounter += localCounter;
    if (pthread_mutex_unlock(&mutex) != 0) {
        errorExit("Mutex unlock");
    }
    printf("Thread %ld processed: %.*s\n", pthread_self(), tData->end - tData->start,
tData->data + tData->start);
    printf("Thread %ld counted lowercase letters: %d\n", pthread_self(), localCounter);
    pthread_exit(NULL);
}
int main(int argc, char* argv[]) {
```

```
if (argc != 3) {
    fprintf(stderr, "Usage: %s <input_file> <number_of_threads>\n", argv[0]);
    exit(1);
}
if (pthread_mutex_init(&mutex, NULL) != 0) {
    errorExit("Mutex initialization");
}
char* filename = argv[1];
int numThreads = atoi(argv[2]);
FILE* file = fopen(filename, "r");
if (file == NULL) {
    errorExit("File opening");
}
fseek(file, 0, SEEK_END);
long fileSize = ftell(file);
rewind(file);
if (fileSize < 2) {
    errorExit("File should contain at least 2 characters");
}
int maxThreads = fileSize / 2;
if (numThreads > maxThreads) {
    numThreads = maxThreads;
    printf("Number of threads reduced to %d to meet the requirement.\n", numThreads);
}
char* fileContent = (char*)malloc(fileSize + 1);
if (fileContent == NULL) {
    errorExit("Memory allocation");
}
if (fread(fileContent, 1, fileSize, file) != fileSize) {
    free(fileContent);
    fclose(file);
    errorExit("File reading");
}
fileContent[fileSize] = '\0';
ThreadData* threadsData = (ThreadData*)malloc(numThreads * sizeof(ThreadData));
if (threadsData == NULL) {
    free(fileContent);
    fclose(file);
    errorExit("Memory allocation for threads data");
}
pthread_t* threadHandles = (pthread_t*)malloc(numThreads * sizeof(pthread_t));
if (threadHandles == NULL) {
    free(fileContent);
    free(threadsData);
```

```
        fclose(file);
        errorExit("Memory allocation for thread handles");
    }
    int segmentSize = fileSize / numThreads;
    int remainingChars = fileSize % numThreads;
    int currentStart = 0;
    for (int i = 0; i < numThreads; i++) {
        threadsData[i].data = fileContent;
        threadsData[i].start = currentStart;
        threadsData[i].end = currentStart + segmentSize + (i < remainingChars ? 1 : 0);
        currentStart = threadsData[i].end;
    }
    for (int i = 0; i < numThreads; i++) {
        if (pthread_create(&threadHandles[i], NULL, countLowerCase, &threadsData[i]) != 0)
{
            free(fileContent);
            free(threadsData);
            fclose(file);
            errorExit("Thread creation");
        }
    }
    for (int i = 0; i < numThreads; i++) {
        if (pthread_join(threadHandles[i], NULL) != 0) {
            free(fileContent);
            free(threadsData);
            fclose(file);
            errorExit("Thread join");
        }
    }
    free(fileContent);
    free(threadsData);
    free(threadHandles);

    if (pthread_mutex_destroy(&mutex) != 0) {
        errorExit("Mutex destruction");
    }
    fclose(file);
    printf("Total lowercase letters in the file: %d\n", sharedCounter);
    exit(0);
}
```

Листинг 3 – Программная реализация для ОС Linux

Давайте разберем этот код по частям:

Подключаемые библиотеки:

1. stdio.h, stdlib.h, string.h, pthread.h, unistd.h, stdbool.h, errno.h, ctype.h: Все эти библиотеки предоставляют функциональность, необходимую для работы с файлами, строками, многопоточностью и другими стандартными операциями в языке C.

Глобальные переменные и структуры:

1. pthread_mutex_t mutex: Мьютекс для синхронизации доступа к общей переменной sharedCounter.
2. int sharedCounter: Переменная, в которой хранится общее количество строчных букв, найденных всеми потоками.
3. ThreadData struct: Структура для передачи данных в каждый поток. Включает указатель на данные, начальный и конечный индексы для обработки.

Функции:

1. errorExit(const char* message): Функция для вывода сообщения об ошибке вместе с сообщением об ошибке операционной системы (через strerror(errno)) и выхода из программы с кодом ошибки.
2. countLowerCase(void* param): Функция, которую выполнят потоки. Считает количество строчных букв в заданном сегменте данных и обновляет sharedCounter с помощью мьютекса для безопасного доступа к общей переменной.
3. main(int argc, char* argv[]): Основная функция программы. Здесь обрабатываются аргументы командной строки, открывается файл, выделяется память и создаются потоки для обработки данных.

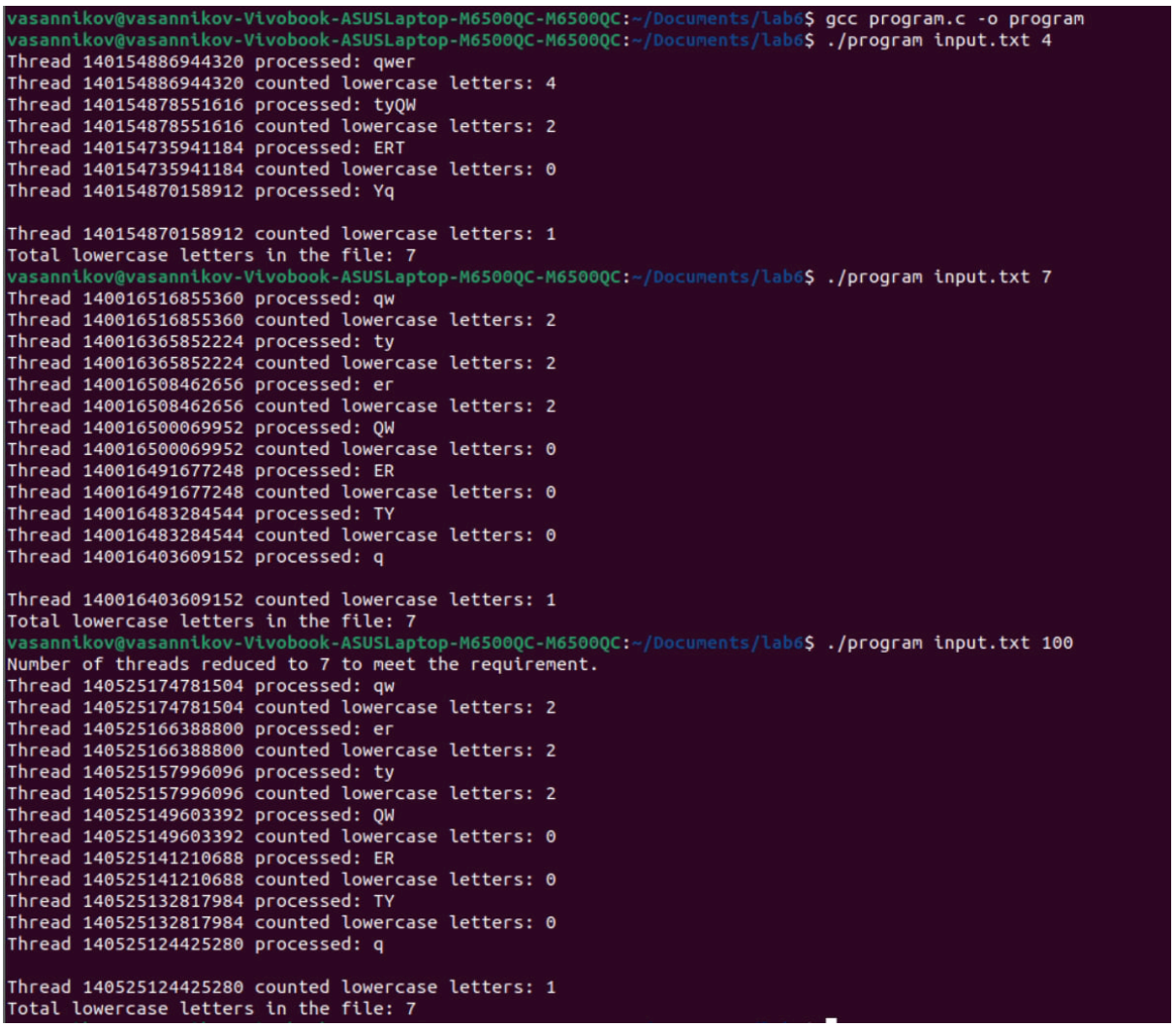
Основная логика программы:

1. Проверяются аргументы командной строки на корректность.
2. Открывается файл и проверяется его наличие и размер.
3. Разделяется файл на сегменты для каждого потока.
4. Создаются потоки для обработки сегментов данных. Каждый поток запускает countLowerCase для подсчета строчных букв в своем сегменте.
5. После завершения работы всех потоков происходит освобождение памяти и мьютекса, закрытие файла и вывод общего количества строчных букв в файле.

Особенности кода:

1. Использование мьютекса pthread_mutex_t для синхронизации доступа к общей переменной sharedCounter, обеспечивающее корректное обновление значения счетчика в многопоточной среде.
2. Динамическое выделение памяти для данных каждого потока и массива потоков.
3. Разделение файла на равные сегменты для обработки множеством потоков, что помогает улучшить производительность при обработке больших файлов.

Этот код позволяет эффективно распараллеливать обработку текстовых данных для подсчета определенных символов (строчных букв в данном случае) с использованием многопоточности и безопасности доступа к общим данным.




Скриншот 2 – Сборка и тестирование программы в ОС Linux.

## 3 Выводы о проделанной работе

Я проделал большую работу, написав две программные реализации для ОС Windows и Linux, удовлетворяющие поставленным требованиям. Моё исследование включает в себя изучение языка C, разработку многопоточных приложений, работу с аргументами командной строки и управление файлами.

Программы, которые я создал, эффективно считывают имя файла и количество потоков из аргументов командной строки, а затем выполняют следующие шаги:

1. Открывают указанный файл и проверяют его наличие и минимальную длину.

2. Делят строку из файла на равные части для каждого дочернего потока.

3. Создают необходимое количество дочерних потоков для обработки каждой части строки.

4. Каждый поток анализирует свою часть строки, подсчитывает количество строчных букв и добавляет результат к общему счетчику.

5. Родительский поток ожидает завершения работы дочерних потоков и выводит общее количество строчных букв в консоль.

Мои программы эффективно используют многопоточность для параллельной обработки данных и общих переменных, обеспечивая корректное выполнение задачи. Также программы проактивно обрабатывают возможные ошибочные сценарии, такие как отсутствие файла или недостаточная длина файла.