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

Практическая работа №6 по дисциплине
«Системное программирование»
Тема работы: «Потоки и синхронизация»

Студент гр. БИБ 201
Морин Денис Александрович
«26» февраля 2023 г.

Руководитель
Преподаватель В.И.Морозов
«___» _____ 2023 г.

Москва 2023

Оглавление

1 Цель работы

Целью данной работы является работа с потоками в операционных системах Windows и Linux и в качестве закрепления материала написание программы, которая должна выполнять условия, которые находятся в Приложении Г.

2 Ход работы

В начале файла winapi.cpp программы для Windows объявляется структура данных для синхронизации доступа к следующей глобальной переменной, структура, принимающая параметры указатель на строку и символ, а также функцию для чтения содержимых букв в файле (Листинг 1).

```
CRITICAL_SECTION cs;
int globalVar(0);

struct ThreadParams {
    string* subString;
    char letter;
}params;

string read_string_from_file(const string& filename) {
    ifstream file(filename);
    if (!file) {
        cerr << "Failed to open file: " << filename << endl;
        exit(1);
    }

    string line;
    getline(file, line);
    string letters;

    for (char c : line) {
        if (isalpha(c)) {
            letters.push_back(tolower(c));
        }
    }
    return letters;
}
```

Листинг 1 – Объявление переменных, структуры и функции работы с файлом

Еще одна функция нужна для работы с потоками, которая складывает промежуточный результат каждого из них в глобальную переменную (Листинг 2).

```
DWORD WINAPI sumLettersAfterGivenOne(LPVOID lpParameter) {

    auto pParams = (ThreadParams*) lpParameter;
    auto* str = pParams->subString;
    char letter = pParams->letter;
    int sum;

    sum = accumulate((*str).begin(), (*str).end(),
                     0, [letter](size_t a, char b) {
        return (b > letter) ? ++a : a;
    });
```

```
    EnterCriticalSection(&cs);
    globalVar += sum;
    LeaveCriticalSection(&cs);
    ExitThread(0);
}
```

Листинг 2 – Функция для работы с потоками

Создаются переменные, куда буду записывать аргументы, переданные при запуске файла, далее создаются потоки, в которые передаются подстроки и аргумент-символ (Листинг 3).

```
int nThreads = stoi(argv[2]);
    params.letter = tolower(argv[3][0]);
    int chunkSize = stringLine.size() / nThreads;


    vector<HANDLE> hThread(nThreads);
    InitializeCriticalSection(& cs);

    for (int i = 0; i < nThreads; i++) {
        int start = i * chunkSize;
        int end = start + chunkSize;
        if (i == nThreads - 1){
            end = stringLine.size();
        }
        string subString = stringLine.substr(start, end - start);
        params.subString = &subString;

            hThread[i] = CreateThread(nullptr, 0, sumLettersAfterGivenOne,
&params, 0, nullptr);

        if (hThread[i] == nullptr) {
            cout << "Error creating thread: " << GetLastError() << endl;
        }
        WaitForSingleObject(hThread[i], INFINITE);
    }
```

Листинг 3 – Функция main файла winapi.cpp

После работы потоков обрабатываю ошибки, если возникли с ними. (Листинг 4).

```
      for (int i = 0; i < nThreads; i++) {
        DWORD exitCode;
        if (!GetExitCodeThread(hThread[i], &exitCode)) {
             cout << "Error getting thread exit code: " << GetLastError() <<
endl;
        }
        CloseHandle(hThread[i]);
    }
```

Листинг 4 – Обработка ошибок потоков

Для операционной системы Linux нужно было использовать мьютексы. Код аналогичен коду на Windows, но вместо критических секций используются мьютексы (Листинг 5).

```
using namespace std;

pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;

int globalVar = 0;

void* thread_func(void* arg) {
    auto* data = static_cast<pair<string, char>*>(arg);
    string& str = data->first;
    char ch = data->second;

    int local_globalVar = 0;
    for (char c : str) {
        if (c > ch)
            local_globalVar++;
    }

    pthread_mutex_lock(&mutex);
    globalVar += local_globalVar;
    pthread_mutex_unlock(&mutex);

    delete data;
    // sleep(10000);
    pthread_exit(nullptr);
}
```

Листинг 5 – Функция для увеличения глобальной переменной

В основном файле создаются потоки, в которые передаются подстроки и символ, а после дожидаются завершения потоков и обрабатываются возможные возникновения ошибок работы с потоками (Листинг 6).

```
   for (int i = 0; i < num_threads; i++) {
        int start = i * part_size;
         int end = (i == num_threads - 1) ? file_contents.size() : (i + 1) *
part_size;
        parts[i] = file_contents.substr(start, end - start);

        auto *data = new pair<string, char>{parts[i], ch};
          pthread_create(&threads[i], nullptr, thread_func, static_cast<void
*>(data));
    }

    for (pthread_t &thread: threads) {
        int rc = pthread_join(thread, nullptr);
        if (rc != 0) {
            switch (rc) {
                case EINVAL:
                    cerr << "Error: invalid thread ID" << endl;
                    break;
                case ESRCH:
```

```
                    cerr << "Error: thread not found" << endl;
                    break;
                case EDEADLK:
                    cerr << "Error: deadlock detected" << endl;
                    break;
                default:
                     cerr << "Error: pthread_join failed with error code " <<
rc << endl;
            }
            exit(1);
        }
    }
```

Листинг 6 –Создание потоков и обработка их ошибок

Работу программы можно посмотреть в Process Hacker, на рисунке 1 показано, как создаются потоки.

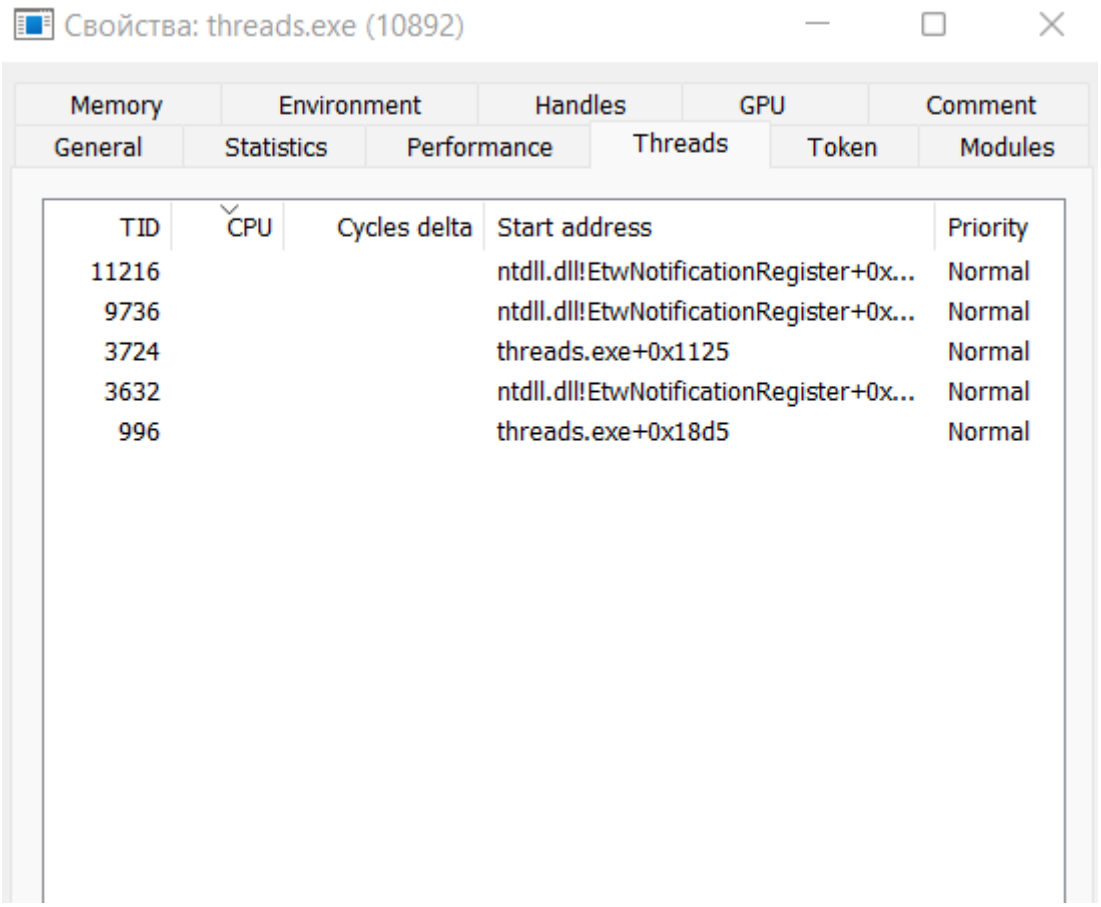

Рисунок 1 – Создание потоков

Далее показано создание всех потоков программой Process Hacker 2 (рисунок 2).

На Linux с помощью top на рисунке 2 показаны созданные потоки.

```
43700 root      20   0       0      0      0 I   0.0   0.0   0:00.00 kworker/0:0-events
44184 root      20   0       0      0      0 I   0.0   0.0   0:00.36 kworker/2:1-mm_percpu_wq
45471 root      20   0       0      0      0 I   0.0   0.0   0:00.12 kworker/2:2-ata_sff
45574 kali      20   0  374324   1892   1720 S   0.0   0.0   0:00.37 untitled
45575 kali      20   0  374324   1892   1720 S   0.0   0.0   0:00.00 untitled
45576 kali      20   0  374324   1892   1720 S   0.0   0.0   0:00.00 untitled
45577 kali      20   0  374324   1892   1720 S   0.0   0.0   0:00.00 untitled
45578 kali      20   0  374324   1892   1720 S   0.0   0.0   0:00.00 untitled
45579 kali      20   0  374324   1892   1720 S   0.0   0.0   0:00.00 untitled
45715 root      20   0       0      0      0 I   0.0   0.0   0:00.34 kworker/u8:1-events_unbo
```

Рисунок 2 –Потоки на ОС Linux

Пример работы программы: для начала создается текстовый файл и записывается какая-то строка (рисунок 3).

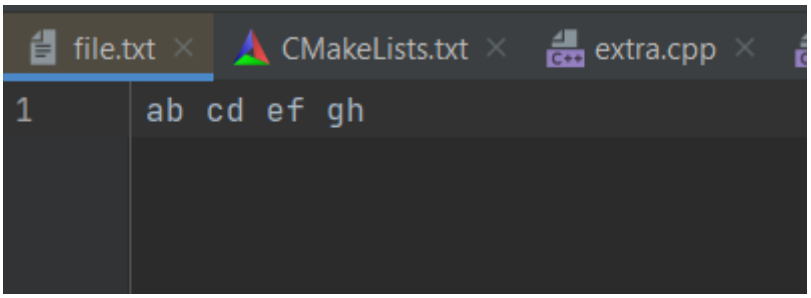

Рисунок 3 – Текстовый файл и его содержимое

Далее вызывается главный файл следующим образом: ./threads.exe file.txt 5 d. Таким образом, прошу создать 5 потоков и в каждом из них посчитать количество символов в файле file.txt, код которых больше кода буквы d.

Результат программы представлен на рисунке 4.

```
C:\Users\lkey2\CLionProjects\threads\cmake-build-debug\threads.exe file.txt 5 d
There are 4 letters, which are after letter 'd' in alphabet

Process finished with exit code 0
```

Рисунок 4 – Результат программы winapi.cpp

Также программа обрабатывает несколько ошибок:

1) Ошибка вызова программы (рисунок 5).

```
C:\Users\lkey2\CLionProjects\threads\cmake-build-debug\threads.exe file.txt 5
Usage: C:\Users\lkey2\CLionProjects\threads\cmake-build-debug\threads.exe <filename> <N> <letter>

Process finished with exit code 1
```

Рисунок 5 – Неправильный вызов программы

2) Нет файла с текстом (рисунок 6).

```
C:\Users\lkey2\CLionProjects\threads\cmake-build-debug\threads.exe file.txt 5 d
Failed to open file: file.txt

Process finished with exit code 1
```

Рисунок 6 – Нет файла

3) Файл содержит символы меньше двух (рисунок 7).

```
C:\Users\lkey2\CLionProjects\threads\cmake-build-debug\threads.exe file.txt 5 d
Not enough symbols in file: file.txt

Process finished with exit code 1
```

Рисунок 7 – Недостаточно символов в тексте

4) Потоки не создались (рисунок 8).

```
C:\Users\lkey2\CLionProjects\threads\cmake-build-debug\threads.exe file.txt 5 d
Error getting thread exit code: 6
Error getting thread exit code: 6
Error getting thread exit code: 6
Error getting thread exit code: 6
Error getting thread exit code: 6
```

Рисунок 8 – код ошибки из-за не создания процесса

В режиме отладки можно увидеть, как возвращаются результаты потоков в глобальную переменную на примере той же строки ввода (рисунок 9).

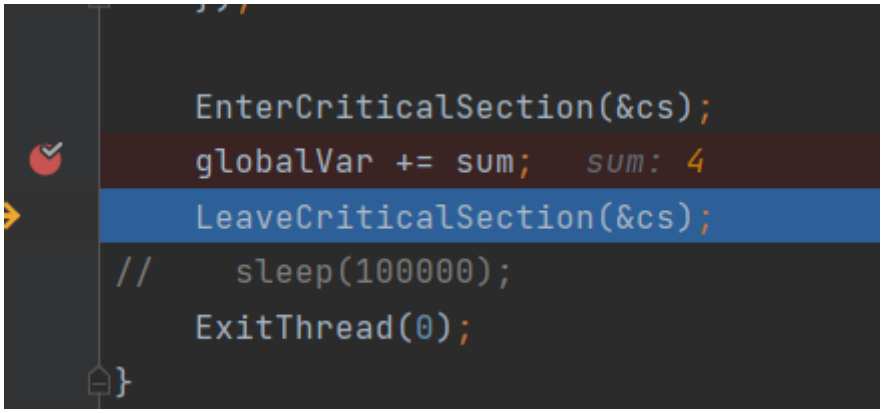

Рисунок 9 – Увеличение глобальной переменной на 4

3 Дополнительное задание

В качестве дополнительного задания была написана кроссплатформенная программу на языке C++. Код программы extra.cpp представлен в Приложении Д.

В данной программе используются атомарные операции для работы с глобальной переменной потокам по одному. Остальные две функции считают число букв после данной в алфавите и прибавляет сумму к глобальной переменной (Листинг 7).

```
atomic<int> globalVariable(0);

int f(const string& str, char ch){
    int count = 0;
    for (char c : str) {
        if (c > ch)
            count++;
    }
    return count;
}

void thread_func(const string& str, char ch) {
    globalVariable.fetch_add(f(str, ch));
}
```

Листинг 7 – Функция подсчета символов в подстроке

В основной части программы строка из файла делится на части для каждого потока, потом каждому потоку передаётся функция и её аргументы. Если ошибка с созданием потока, то она обрабатывается. И дожидается завершение потока (Листинг 8).

```
    thread threads[num_threads];
    vector<string> parts(num_threads);
    int part_size = file_contents.size() / num_threads;

    for (int i = 0; i < num_threads; i++) {
        int start = i * part_size;
         int end = (i == num_threads - 1) ? file_contents.size() : (i + 1) *
part_size;
        parts[i] = file_contents.substr(start, end - start);
    }

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

```
            threads[i] = thread(thread_func, ref(parts[i]), ch);
        } catch (const std::system_error& e) {
                cerr << "Error: failed to create thread " << i << ": " <<
e.what() << endl;
            for (int j = 0; j < i; j++) {
                threads[j].join();
            }
            return 1;
        }
    }

    for (thread& t : threads) {
        t.join();
    }
```

Листинг 8 – Работа с потоками

3 Вывод

Потоки и синхронизация являются важными аспектами многопоточного программирования как в операционной системе Windows, так и в Linux. В обеих операционных системах используются различные механизмы синхронизации и управления потоками, такие как мьютексы, семафоры и критические секции.

В Windows механизм синхронизации основан на объектах ядра, которые обеспечивают широкий спектр возможностей, включая поддержку ожидания на нескольких объектах. В Linux механизм синхронизации основан на примитивах синхронизации ядра, но также существуют примитивы синхронизации пользователя.

В обеих операционных системах предоставляются различные функции и возможности управления потоками, но API и синтаксис для работы с потоками могут отличаться. При разработке многопоточных приложений, которые должны работать на разных операционных системах, необходимо учитывать эти различия.

Исходный код программы winapi.cpp, extra.cpp и linux.cpp для Linux находятся в приложении А, Б и В соответственно.

ПРИЛОЖЕНИЕ А.

Программа winapi.cpp для операционной системы Windows

```
#include <windows.h>
#include <tchar.h>
#include <iostream>
#include <fstream>
#include <vector>
#include <algorithm>
#include <numeric>
#include <string>
//#include <unistd.h>

using namespace std;

CRITICAL_SECTION cs;
int globalVar(0);

struct ThreadParams {
    string* subString;
    char letter;
}params;

string read_string_from_file(const string& filename) {
    ifstream file(filename);
    if (!file) {
        cerr << "Failed to open file: " << filename << endl;
        exit(1);
    }

    string line;
    getline(file, line);
    string letters;

    for (char c : line) {
        if (isalpha(c)) {
            letters.push_back(tolower(c));
        }
    }
    return letters;
}


DWORD WINAPI sumLettersAfterGivenOne(LPVOID lpParameter) {

    auto pParams = (ThreadParams*) lpParameter;
    auto* str = pParams->subString;
    char letter = pParams->letter;
    int sum;

    sum = accumulate((*str).begin(), (*str).end(),
                     0, [letter](size_t a, char b) {
        return (b > letter) ? ++a : a;
    });

    EnterCriticalSection(&cs);
    globalVar += sum;
    LeaveCriticalSection(&cs);
//    sleep(100000);
    ExitThread(0);
```

```
}

int main(int argc, char *argv[]) {
    if (argc != 4) {
        cerr << "Usage: " << argv[0] << " <filename> <N> <letter>" << endl;
        return 1;
    }

    string filename = argv[1];
    string stringLine = read_string_from_file(filename);
    if (stringLine.size() < 2) {
        cerr << "Not enough symbols in file: " << filename << endl;
        return 1;
    }

    int nThreads = stoi(argv[2]);
    params.letter = tolower(argv[3][0]);
    int chunkSize = stringLine.size() / nThreads;


    vector<HANDLE> hThread(nThreads);
    InitializeCriticalSection(& cs);

    for (int i = 0; i < nThreads; i++) {
        int start = i * chunkSize;
        int end = start + chunkSize;
        if (i == nThreads - 1){
            end = stringLine.size();
        }
        string subString = stringLine.substr(start, end - start);
        params.subString = &subString;

        hThread[i] = CreateThread(nullptr, 0, sumLettersAfterGivenOne,
&params, 0, nullptr);
        if (hThread[i] == nullptr) {
            cout << "Error creating thread: " << GetLastError() << endl;
        }
        WaitForSingleObject(hThread[i], INFINITE);
    }



    for (int i = 0; i < nThreads; i++) {
        DWORD exitCode;
        if (!GetExitCodeThread(hThread[i], &exitCode)) {
            cout << "Error getting thread exit code: " << GetLastError() <<
endl;
        }
        CloseHandle(hThread[i]);
    }
    cout << "There are " << globalVar << " letters, which are after letter
'"
    << argv[3] << "' in alphabet" << endl;

    DeleteCriticalSection(& cs);

    ExitProcess(0);
}
```

ПРИЛОЖЕНИЕ Б.

Кроссплатформенная программа extra.cpp

```
#include <iostream>
#include <fstream>
#include <thread>
#include <vector>
#include <algorithm>
#include <string>
#include <atomic>


using namespace std;

atomic<int> globalVariable(0);

int f(const string& str, char ch){
    int count = 0;
    for (char c : str) {
        if (c > ch)
            count++;
    }
    return count;
}

void thread_func(const string& str, char ch) {
    globalVariable.fetch_add(f(str, ch));
}

int main(int argc, char* argv[]) {
    if (argc < 4) {
        cerr << "Usage: " << argv[0] << " <filename> <num_threads> <char>"
<< endl;
        return 1;
    }

    string filename = argv[1];
    int num_threads = stoi(argv[2]);
    char ch = argv[3][0];

    ifstream infile(filename);
    if (!infile.good()) {
        cerr << "Error: file '" << filename << "' does not exist or cannot
be read" << endl;
        return 1;
    }

    string file_contents((istreambuf_iterator<char>(infile)),
                         istreambuf_iterator<char>());

    if (file_contents.size() < 2) {
        cerr << "Error: file '" << filename << "' contains less than 2
characters" << endl;
        return 1;
    }

    thread threads[num_threads];
    vector<string> parts(num_threads);
    int part_size = file_contents.size() / num_threads;
```

```
    for (int i = 0; i < num_threads; i++) {
        int start = i * part_size;
        int end = (i == num_threads - 1) ? file_contents.size() : (i + 1) *
part_size;
        parts[i] = file_contents.substr(start, end - start);
    }

    for (int i = 0; i < num_threads; i++) {
        try {
            threads[i] = thread(thread_func, ref(parts[i]), ch);
        } catch (const std::system_error& e) {
            cerr << "Error: failed to create thread " << i << ": " <<
e.what() << endl;
            for (int j = 0; j < i; j++) {
                threads[j].join();
            }
            return 1;
        }
    }

    for (thread& t : threads) {
        t.join();
    }

    cout << "There are " << globalVariable << " letters, which are after
letter '"
         << argv[3] << "' in alphabet" << endl;

    return 0;
}
```

ПРИЛОЖЕНИЕ В.

Программа linux.cpp для операционной системы Linux

```
#include <iostream>
#include <fstream>
#include <pthread.h>
#include <vector>
#include <algorithm>
#include <string>
//#include <unistd.h>

using namespace std;

pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;

int globalVar = 0;

void* thread_func(void* arg) {
    auto* data = static_cast<pair<string, char>*>(arg);
    string& str = data->first;
    char ch = data->second;

    int local_globalVar = 0;
    for (char c : str) {
        if (c > ch)
            local_globalVar++;
    }

    pthread_mutex_lock(&mutex);
    globalVar += local_globalVar;
    pthread_mutex_unlock(&mutex);

    delete data;
    // sleep(10000);
    pthread_exit(nullptr);
}

int main(int argc, char* argv[]) {
    if (argc < 4) {
        cerr << "Usage: " << argv[0] << " <filename> <num_threads> <char>"
<< endl;
        return 1;
    }

    string filename = argv[1];
    int num_threads = stoi(argv[2]);
    char ch = argv[3][0];

    ifstream infile(filename);
    if (!infile.good()) {
        cerr << "Error: file '" << filename << "' does not exist or cannot
be read" << endl;
        return 1;
    }

    string file_contents((istreambuf_iterator<char>(infile)),
                         istreambuf_iterator<char>());

    if (file_contents.size() < 2) {
```

```
        cerr << "Error: file '" << filename << "' contains less than 2
characters" << endl;
        return 1;
    }

    if (file_contents.size() < num_threads) {
        cerr << "Error: file '" << filename << "' the data in file is less
than threads" << endl;
        num_threads /= 2;
        cerr << "The number of threads will be " << num_threads << endl;
    }

    int part_size = file_contents.size() / num_threads;
    vector<string> parts(num_threads);
    vector<pthread_t> threads(num_threads);

    for (int i = 0; i < num_threads; i++) {
        int start = i * part_size;
        int end = (i == num_threads - 1) ? file_contents.size() : (i + 1) *
part_size;
        parts[i] = file_contents.substr(start, end - start);

        auto *data = new pair<string, char>{parts[i], ch};
        pthread_create(&threads[i], nullptr, thread_func, static_cast<void
*>(data));
    }

    for (pthread_t &thread: threads) {
        int rc = pthread_join(thread, nullptr);
        if (rc != 0) {
            switch (rc) {
                case EINVAL:
                    cerr << "Error: invalid thread ID" << endl;
                    break;
                case ESRCH:
                    cerr << "Error: thread not found" << endl;
                    break;
                case EDEADLK:
                    cerr << "Error: deadlock detected" << endl;
                    break;
                default:
                    cerr << "Error: pthread_join failed with error code "
<< rc << endl;
            }
            exit(1);
        }
    }

    cout << "Total globalVar: " << globalVar << endl;

    return 0;
}
```

ПРИЛОЖЕНИЕ Г.

Текст задания

В файле записана строка. Программа должна считать имя файла из первого аргумента командной строки и рассчитать количество таких символов в этой строке, которые стоят в алфавите после символа, переданного в качестве третьего аргумента командной строки. Для расчёта количества символов программа должна создать N дочерних потоков (N передаётся вторым аргументом командной строки) и передать каждому из них часть полученной строки. Каждый из дочерних потоков должен рассчитать количество соответствующих условию символов в переданной ему части строки и вернуть его родительскому. Родительский поток должен просуммировать полученные от дочерних числа и вывести на консоль итоговую сумму. Если исходный файл не существует, или в нём записано менее 2 символов, следует вывести соответствующее сообщение для пользователя и завершить работу программы. Под позицией в алфавите в данной задаче понимается позиция в таблице ASCII.