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 №4 по дисциплине
«Системное программирование»
Тема работы: «Процессы»

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

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

Москва 2023

Оглавление

1 Цель работы

Целью данной работы является работа с процессами 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 linux.cpp программы для Linux находится функция чтения данных из файла file.txt, удаляя все символы, кроме букв (Листинг 1).

```
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(c);
        }
    }

    return letters;
}
```

Листинг 1 – Подключение библиотек и введение констант в main.c

В главной функции main считываются аргументы, переданные при запуске файла, и инициализируются переменные для работы со строкой и потоками (Листинг 2).

```
int N = atoi(argv[2]);
    char letter = argv[3][0];
    vector<pid_t> children(N);
    vector<int> sums(N);
    int chunk_size = (int) stringLine.size() / N;
    int rest = (int) stringLine.size() % N;
```

Листинг 2 – Функция для печати текста в консоль

Создаются потоки, в которых будет выполняться сравнение кодов символов подстрок полученной строки (Листинг 3).

```
for (int i = 0; i < N; ++i) {
        int start = i * chunk_size + min(i, rest);
```

```
        int end = start + chunk_size + (i < rest);
        pid_t pid = fork();
        switch(pid){
            case 0:
                exit(0);
            case -1:
                printf("Error %d\n", errno);
                exit(1);
            default:
                getchar();
                children[i] = pid;
                          sums[i] = accumulate(stringLine.begin() + start,
stringLine.begin() + end,
                                               0, [&letter](size_t a, char b) {
                    return (b > letter) ? ++a : a;
                });
        }
    }
```

Листинг 3 – Функция main файла main.c

После работы подпроцессов с помощью вызова PIDов общий результат складывает результаты подпроцессов. (Листинг 4).

```
   int result = 0;
    for (int i = 0; i < N; ++i) {
        waitpid(children[i], nullptr, 0);
        read(children[i], &sums[i], sizeof(sums[i]));
        result += sums[i];
    }
```

Листинг 4 – Создание дескриптора файла

Для операционной системы Windows нужно использовать два файла, основной, в котором создаются подпроцессы, в которых вызывается исполняемый файл другой программы.

Исполняемый файл принимает строку и символ, а выводит количество символов в строке, которые стоят после данной буквы в алфавите (Листинг 5).

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

    string stringLine = argv[1];
```

```
    char letter = argv[2][0];

        int sums = accumulate(stringLine.begin(), stringLine.begin() +
stringLine.size(),
                              0, [&letter](size_t a, char b) {
            return (b > letter) ? ++a : a;
        });

    return sums;
}
```

Листинг 6 – Запись строки из консоли

В основном файле создаются потоки, в которые передаётся исполняемый файл с аргументами, чтобы проделать операцию с подстроками для каждого подпроцесса (Листинг 7).

```
for (int i = 0; i < N; ++i) {
        int start = i * chunk_size + min(i, rest);
        int end = start + chunk_size + (i < rest);
        STARTUPINFO si = {0};
        string str = stringLine.substr(start, end - start);
        string lpCommandLine = "a.exe " + str + " " + letter;
        PROCESS_INFORMATION pi = {nullptr};

        if (!CreateProcess(nullptr,
                           lpCommandLine.data(),
                                   nullptr,
                                   nullptr,
                                   TRUE,
                                   0,
                                   nullptr,
                                   nullptr,
                                   &si,
                                   &pi)) {

            cout << "Error: " << GetLastError() << endl;
            ExitProcess(0);
        }
        else{
            cout << "hell: ";
            getchar();
            children[i] = pi.hProcess;
            CloseHandle(pi.hThread);
        }
    }
```

Листинг 7 – Продолжение функции main

Для вывода результатов работы исполняемого файла следует обращаться к функции GetExitCodeProcess, которая после ожидания завершения подпроцесса запишет в нужную переменную ответ (Листинг 8).

```
size_t result = 0;
 for (int i = 0; i < N; ++i) {
     WaitForSingleObject(children[i], INFINITE);
     DWORD lpExitCode;
     GetExitCodeProcess(children[i], &lpExitCode);
     result += lpExitCode;
     CloseHandle( children[i] );
 }
```

Листинг 8 – Импорт библиотек и создание констант в файле api.c

Работу программы можно посмотреть в Process Hacker, на рисунке 1 показано, как создается первый подпроцесс.

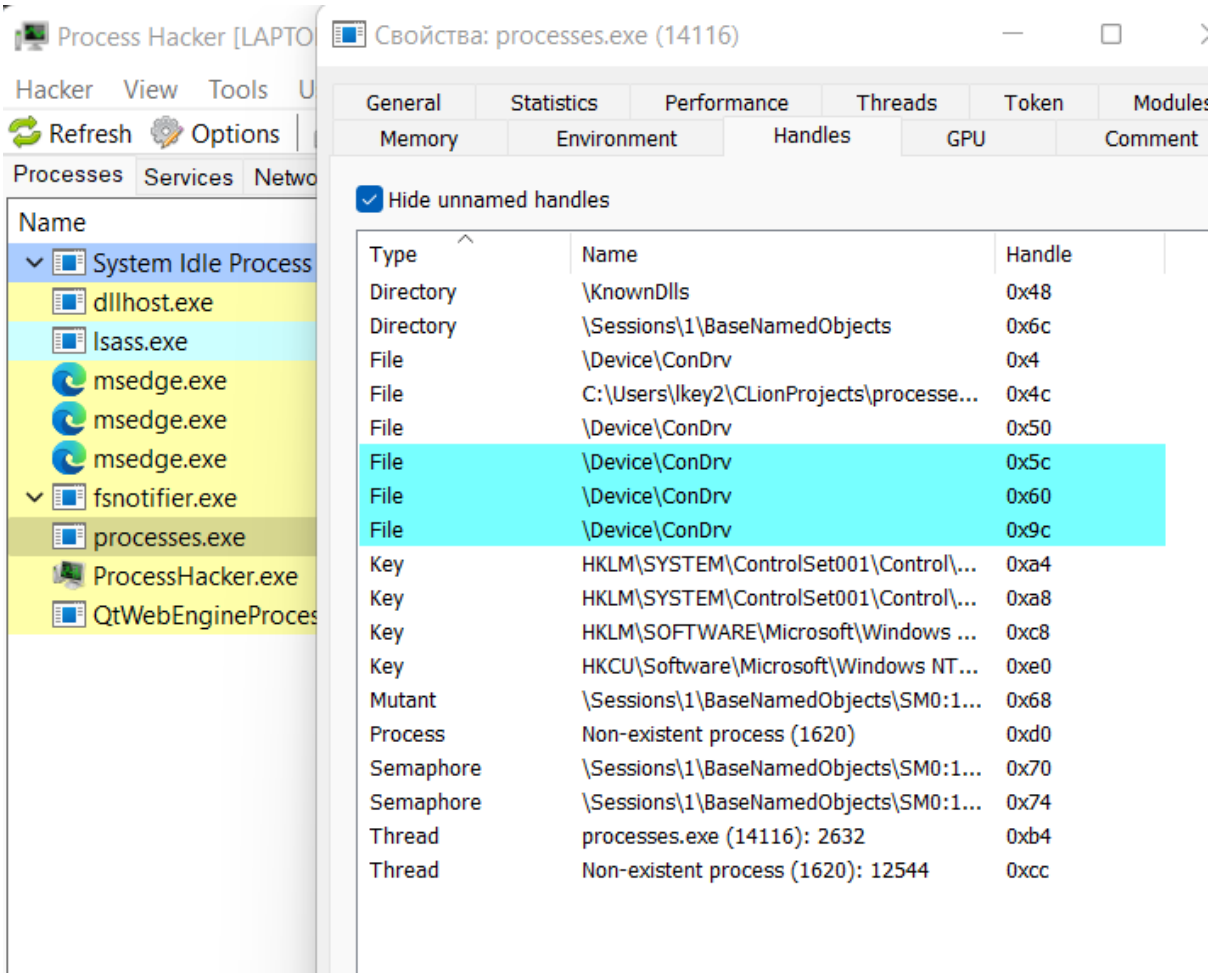

Рисунок 1 – Создание подпроцесса

Далее показано создание всех трех подпроцессов программой (рисунок 2).

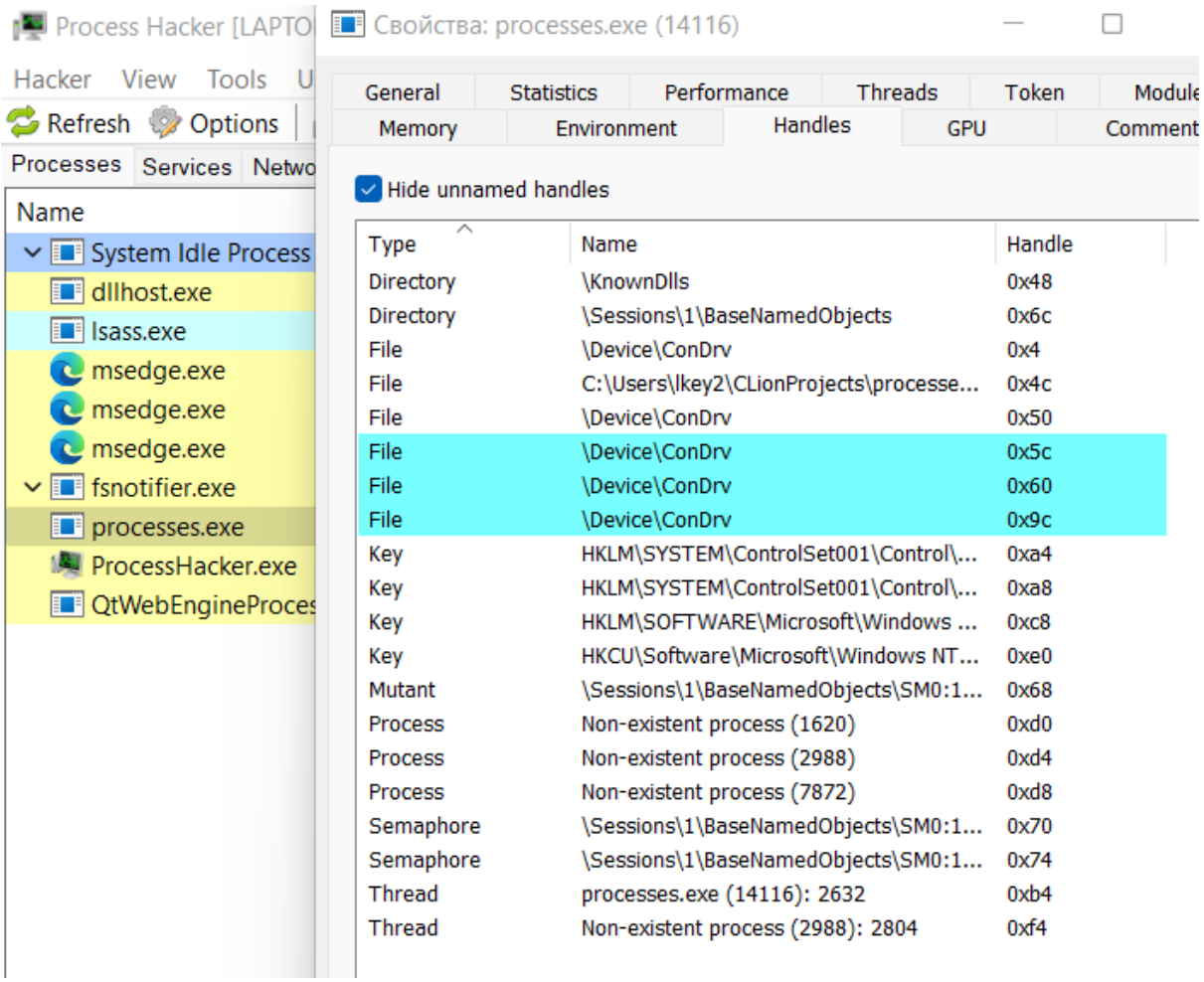

Рисунок 2 – Создание трех подпроцессов

На Linux с помощью top были найдены “зомби”, на рисунке 3 показано, как появился первых из трех.

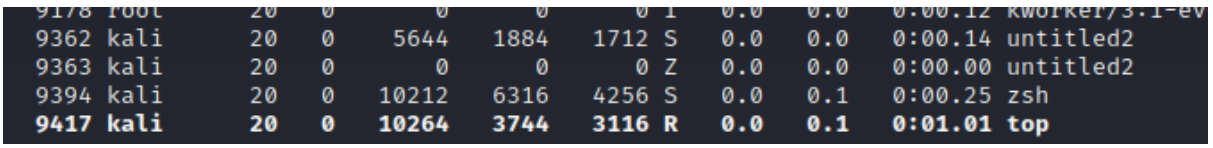

Рисунок 3 – Появление первого “зомби”

И появление всех “зомби” (рисунок 4).

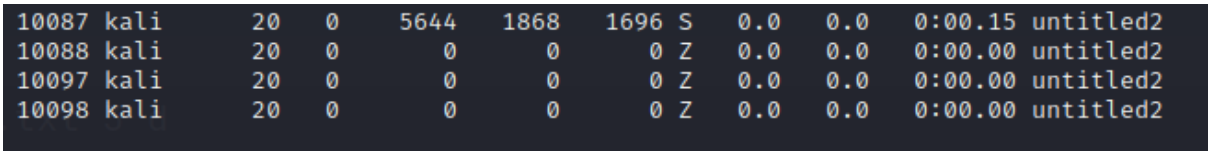

Рисунок 4 – Появление трех “зомби”

Пример работы программы: для начала создается текстовый файл и записывается какая-то строка (рисунок 5).

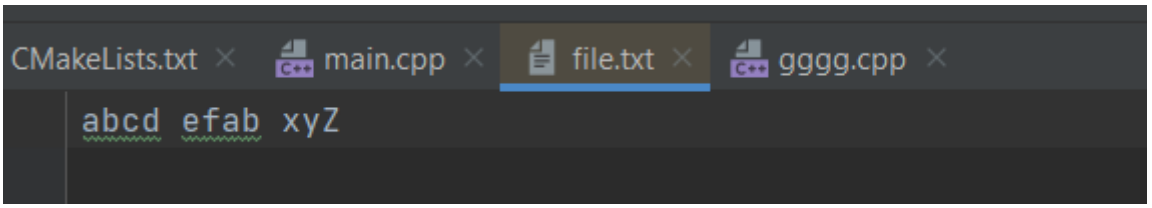

Рисунок 5 – Текстовый файл и его содержимое

Далее вызывается главный файл следующим образом: ./main.exe file.txt 3 d. Таким образом, прошу создать 3 дочерних процесса и в каждом из них посчитать количество символов в файле file.txt, код которых больше кода буквы d.

Результат программы представлен на рисунке 6.

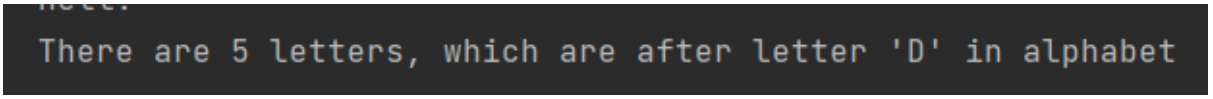

Рисунок 6 – Результат программы main.cpp

Также программа обрабатывает несколько ошибок:

1) Ошибка вызова программы (рисунок 7).

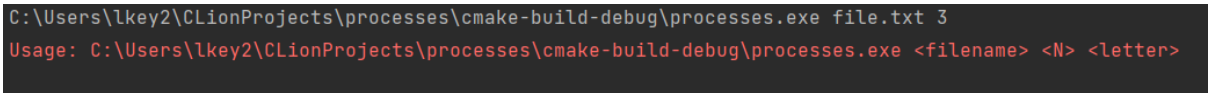

Рисунок 7 – Неправильный вызов программы

2) Нет файла с текстом (рисунок 8).

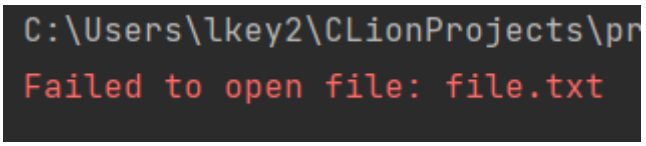

Рисунок 8 – Нет файла

3) Файл содержит символы меньше двух (рисунок 9).

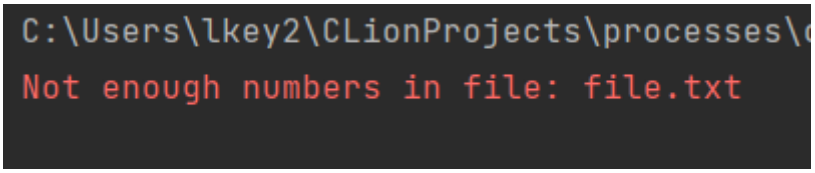

Рисунок 9 – Недостаточно символов в тексте

4) Не создался дочерний процесс (рисунок 10).

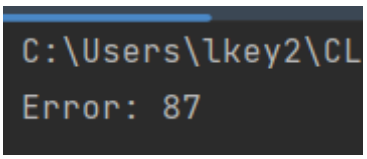

Рисунок 10 – код ошибки из-за не создания процесса

В режиме отладки можно увидеть, как возвращаются результаты подпроцессов в переменную lpExitCode на примере той же строки ввода (рисунок 11).

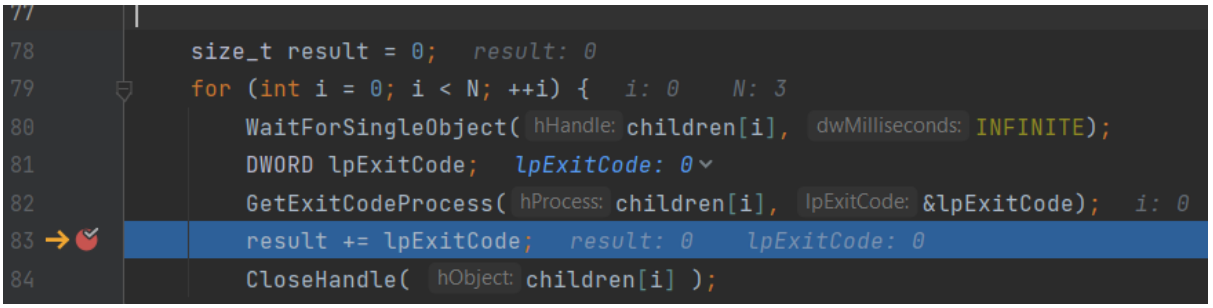

Рисунок 11 – Возврат 0 из первого подпроцесса

Далее происходит возврат значения из второго подпроцесса (рисунок 12).

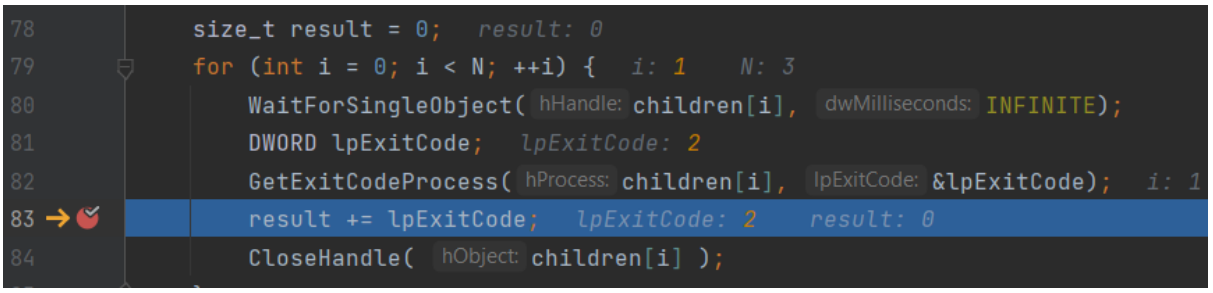

Рисунок 12 – Возврат 2 из второго подпроцесса

И из последнего дочернего процесса (рисунок 13).

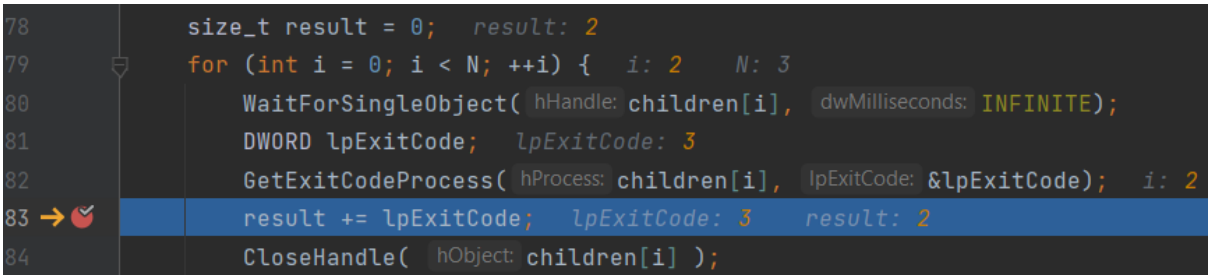

Рисунок 13 – Возврат 3 из третьего подпроцесса

3 Дополнительное задание

В качестве дополнительного задания было предложено написать кроссплатформенную программу не на языке Си/С++. Был выбран язык python. Код программы processes.py представлен в Приложении Д.

В этой программе есть функция подсчета символов в подстроке, которые проходят условие задания, результат кладется в result_queue, чтобы из дочернего процесса после передать в родительский для подсчета итоговой суммы (Листинг 9).

```
def count_characters(start, end, search_char, string, result_queue):
    count = 0
    for i in range(start, end):
        if ord(string[i]) > ord(search_char) and string[i].isalpha():
            count += 1
    result_queue.put(count)
```

Листинг 9 – Функция подсчета символов в подстроке

В основной части программы считается количество символов, которые будут в подстроке, начало и конец каждой из подстрок, выделяется подпроцесс для работы с приведенной выше функцией. После данный подпроцесс добавляется ко всем процессам (Листинг 10).

```
   chunk_size = len(string) // n_processes
    result_queue = Queue()
    processes = []
    start = 0
    for i in range(n_processes):
        end = start + chunk_size
        if i == n_processes - 1:
            end = len(string)
              process = Process(target=count_characters, args=(start, end,
search_char, string, result_queue))
        processes.append(process)
        process.start()
        start = end
```

Листинг 10 – Работа с дочерними процессами

Далее считается общая сумма нужных букв и поочередно завершаются все начатые подпроцессы (Листинг 11).

```
total_count = 0
 for i in range(n_processes):
     total_count += result_queue.get()

 for process in processes:
     process.join()
```

Листинг 11 – Получение итогового ответа и завершение работы дочерних процессов

3 Вывод

На этой практике были изучены процессы на операционных системах Windows и Linux и их программирование на c++.

Исходный код программы main.cpp, gggg.cpp для Windows и linux.cpp для Linux находятся в приложении А, Б и В соответственно.

ПРИЛОЖЕНИЕ А.

Программа main.cpp для операционной системы Windows

```
#include <iostream>
#include <fstream>
#include <vector>
#include <windows.h>
#include <cstdio>
#include <string>
#include <cctype>


using namespace std;

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;
}

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 numbers in file: " << filename << endl;
        return 1;
    }

    int N = atoi(argv[2]);
    string letter = argv[3];
    std::transform(letter.begin(), letter.end(), letter.begin(),
                   [](unsigned char c){ return std::tolower(c); });
    vector<HANDLE> children(N);
//    vector<int> sums(N);
    int chunk_size = (int) stringLine.size() / N;
    int rest = (int) stringLine.size() % N;

    for (int i = 0; i < N; ++i) {
        int start = i * chunk_size + min(i, rest);
        int end = start + chunk_size + (i < rest);
        STARTUPINFO si = {0};
        string str = stringLine.substr(start, end - start);
        string lpCommandLine = "a.exe " + str + " " + letter;
        PROCESS_INFORMATION pi = {nullptr};

        if (!CreateProcess(nullptr,
                           lpCommandLine.data(),
                                   nullptr,
                                   nullptr,
                                   TRUE,
                                   0,
                                   nullptr,
                                   nullptr,
                                   &si,
                                   &pi)) {

            cout << "Error: " << GetLastError() << endl;
            ExitProcess(0);
        }
        else{
//            cout << "hell: ";
//            getchar();
            children[i] = pi.hProcess;
            CloseHandle(pi.hThread);
        }
```

```
    }

    size_t result = 0;
    for (int i = 0; i < N; ++i) {
        WaitForSingleObject(children[i], INFINITE);
        DWORD lpExitCode;
        GetExitCodeProcess(children[i], &lpExitCode);
        result += lpExitCode;
        CloseHandle( children[i] );
    }

//    Sleep(50000);
    cout << "There are " << result << " letters, which are after letter '"
<< argv[3] << "' in alphabet" << endl;
    return 0;
}
```

ПРИЛОЖЕНИЕ Б.

Программа gggg.cpp для операционной системы Windows

```
#include <iostream>
#include <vector>
#include <numeric>
#include <cstdio>


using namespace std;

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

    string stringLine = argv[1];
    char letter = argv[2][0];

    int sums = accumulate(stringLine.begin(), stringLine.begin() +
stringLine.size(),
                          0, [&letter](size_t a, char b) {
            return (b > letter) ? ++a : a;
        });

    return sums;
}
```

ПРИЛОЖЕНИЕ В.

Программа linux.cpp для операционной системы Linux

```
#include <iostream>
#include <fstream>
#include <vector>
#include <algorithm>
#include <numeric>
#include <cstring>
#include <unistd.h>
#include <wait.h>

using namespace std;

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(c);
        }
    }

    return letters;
}

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 numbers in file: " << filename << endl;
        return 1;
    }

    int N = atoi(argv[2]);
    char letter = argv[3][0];
    vector<pid_t> children(N);
    vector<int> sums(N);
    int chunk_size = (int) stringLine.size() / N;
    int rest = (int) stringLine.size() % N;

    for (int i = 0; i < N; ++i) {
        int start = i * chunk_size + min(i, rest);
        int end = start + chunk_size + (i < rest);
        pid_t pid = fork();
        switch(pid){
            case 0:
                exit(0);
            case -1:
                printf("Error %d\n", errno);
                exit(1);
            default:
                getchar();
                children[i] = pid;
                sums[i] = accumulate(stringLine.begin() + start,
stringLine.begin() + end,
                                     0, [&letter](size_t a, char b) {
                    return (b > letter) ? ++a : a;
                });
        }
    }

    int result = 0;
    for (int i = 0; i < N; ++i) {
        waitpid(children[i], nullptr, 0);
        read(children[i], &sums[i], sizeof(sums[i]));
        result += sums[i];
    }
```

```
    cout << "There are " << result << " letters, which are after letter '"
<< argv[3] << "' in alphabet" << 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 процессов (N передается вторым аргументом командной строки) и передать каждому из них часть полученной строки. Каждый из дочерних процессов должен рассчитать количество соответствующих условию символов в переданной ему части строки и вернуть его родительскому. Родительский процесс 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.

ПРИЛОЖЕНИЕ Д.

Программа processes.py

```
import sys
import os
from multiprocessing import Process, Queue


def count_characters(start, end, search_char, string, result_queue):
    count = 0
    for i in range(start, end):
        if ord(string[i]) > ord(search_char) and string[i].isalpha():
            count += 1
    result_queue.put(count)


if __name__ == '__main__':
    file_name = sys.argv[1]
    n_processes = int(sys.argv[2])
    search_char = sys.argv[3].lower()

    if not os.path.exists(file_name):
        print(f"File {file_name} does not exist.")
        sys.exit(1)

    with open(file_name, 'r') as file:
        string = file.readline().replace(" ", "").lower()

    if len(string) < 2:
        print("The string in the file must contain at least 2 characters.")
        sys.exit(1)

    chunk_size = len(string) // n_processes
    result_queue = Queue()
    processes = []
    start = 0
    for i in range(n_processes):
        end = start + chunk_size
        if i == n_processes - 1:
            end = len(string)
```

```
        process = Process(target=count_characters, args=(start, end,
search_char, string, result_queue))
        processes.append(process)
        process.start()
        start = end

    total_count = 0
    for i in range(n_processes):
        total_count += result_queue.get()

    for process in processes:
        process.join()

    print(f"There are {total_count} letters, which are after letter
\'{search_char}\' in alphabet")
```