МИНИСТЕРСТВО ТРАНСПОРТА РФ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «РОССИЙСКИЙ УНИВЕРСИТЕТ ТРАНСПОРТА (МИИТ)»

Кафедра ЦТУТП

**ОТЧЁТ**

**ПО КУРСОВОЙ РАБОТЕ**   
по дисциплине «Операционные системы и системное программирование»

Тема: «ОРГАНИЗАЦИЯ МНОГОПОТОЧНЫХ ПРИЛОЖЕНИЙ В ОС WINDOWS»

Выполнил: Шедания В. М.

Группа: УИС-311

Преподаватель: доц. Варфоломеев В. А.

Борисенко Ф. А.

­

Москва 2024 г.

**СОДЕРЖАНИЕ**

[ВАРИАНТ ЗАДАНИЯ 3](#_Toc167731772)

[БЛОК-СХЕМА АЛГОРИТМА 4](#_Toc167731773)

[ТЕКСТ ПРОГРАММ 5](#_Toc167731774)

[Программа A 5](#_Toc167731775)

[Программа B 11](#_Toc167731776)

[Программа C 20](#_Toc167731777)

[РЕЗУЛЬТАТ РАБОТЫ ПРОГРАММ 29](#_Toc167731778)

[СКРИНШОТЫ ОКНА УТИЛИТЫ PROCESS EXPLORER 30](#_Toc167731779)

[ОПИСАНИЕ И РЕЗУЛЬТАТЫ ПРОВЕДЕННОГО РЕЗУЛЬТАТА 31](#_Toc167731780)

[ДИАГРАММЫ ТРАСС ПОТОКОВ 32](#_Toc167731781)

[ВЫВОДЫ ПО РЕЗУЛЬТАТАМ ПРОВЕДЕННГО ЭКСПЕРИМЕНТА 33](#_Toc167731782)

[СПИСКО ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 34](#_Toc167731783)

# ВАРИАНТ ЗАДАНИЯ

Разработать три консольные программы, выполняющие обработку заданного множества текстовых файлов в соответствии с индивидуальным заданием (см. табл.1):

A – однопоточная программа с последовательной (циклической) обработкой файлов;

B – многопоточная программа с параллельной обработкой файлов;

C – многопоточная программа с параллельной обработкой файлов и повышенным приоритетом потоков.

В каждой программе производить измерение общего времени обработки всех файлов и время, затраченное на обработку каждого файла в отдельности. Программы должны содержать шапку в виде комментария с указанием фамилии студента и номера группы, номера варианта и назначения программы.

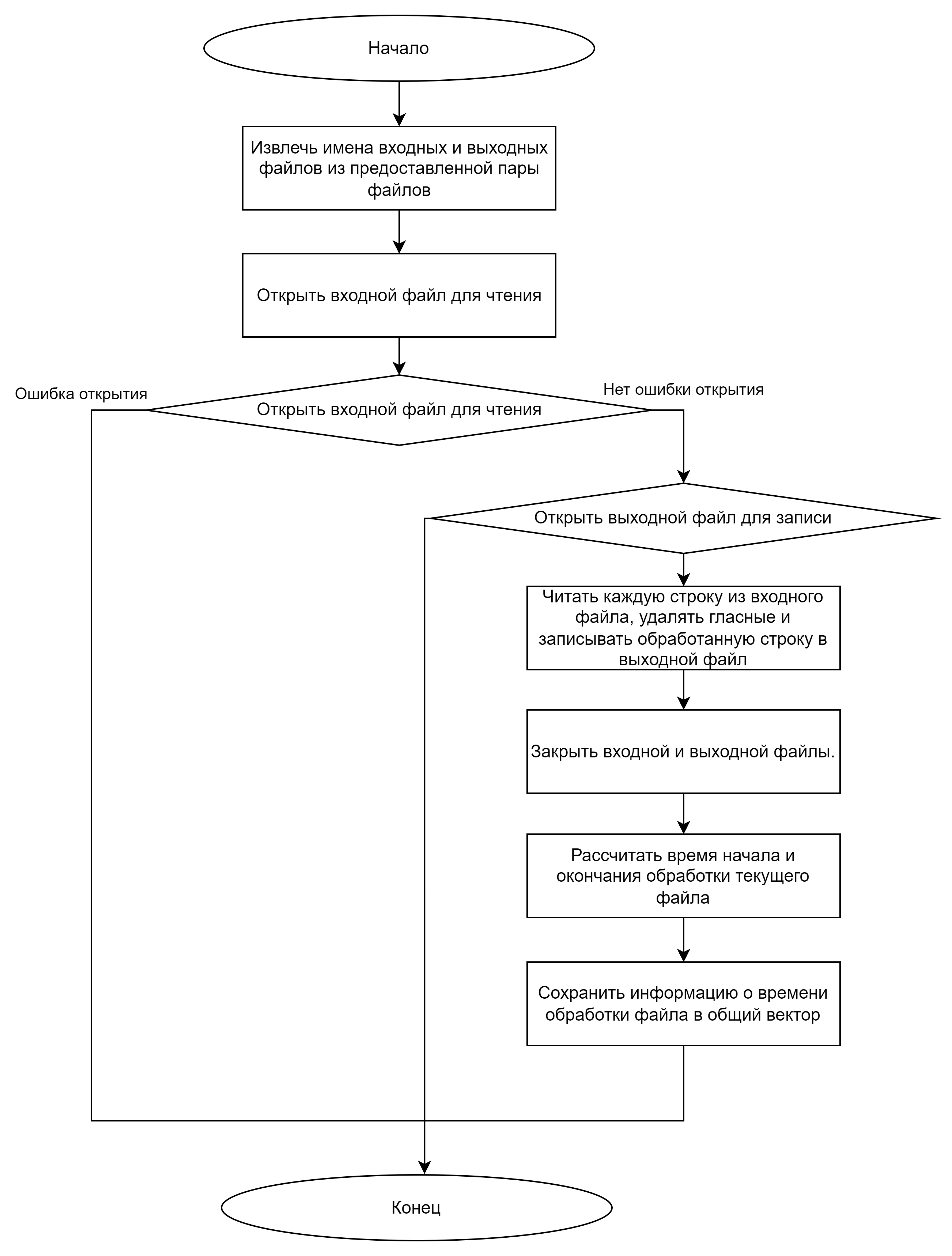
Вариант: 19

Таблица 1 – Вариант задания

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | № | Выполняемые действия | | 19 | Исключить главные буквы из всех слов текста | |  |
|  |  |

# БЛОК-СХЕМА АЛГОРИТМА





# ТЕКСТ ПРОГРАММ

## **Программа A**

#include <iostream>

#include <fstream>

#include <sstream>

#include <string>

#include <unordered\_set>

#include <cctype>

#include <windows.h>

#include <vector>

#include <time.h>

#include "json.hpp"

using json = nlohmann::json;

struct FileProcessingTime

{

    std::string filename;

    clock\_t start\_time;

    clock\_t end\_time;

};

std::vector<FileProcessingTime> fileProcessingTimes;

std::unordered\_set<std::string> fileNamesSet;

clock\_t startTime;

clock\_t endTime;

std::string ssdDirectoryIn = "C:\\CodingProjects\\C++\\osisp\\v\\kr\\files\\in\\";

std::string ssdDirectoryOut = "C:\\CodingProjects\\C++\\osisp\\v\\kr\\files\\out\\";

std::string flashDirectoryIn = "E:\\files\\in\\";

std::string flashDirectoryOut = "E:\\files\\out\\";

// Function to check if a character is a vowel

bool isVowel(char c)

{

    c = std::tolower(c);

    return (c == 'a' || c == 'e' || c == 'i' || c == 'o' || c == 'u' ||

            c == '?' || c == '?' || c == '?' || c == '?' || c == '?' ||

            c == '?' || c == '?' || c == '?' || c == '?' || c == '?');

}

// Function to remove vowels from a string

std::string removeVowels(const std::string &text)

{

    std::string result;

    for (char c : text)

    {

        if (!isVowel(c))

        {

            result += c;

        }

    }

    return result;

}

void processFile(const std::string &inputFilename, const std::string &outputFilename)

{

    FileProcessingTime processingTime;

    clock\_t startFileTime, endFileTime;

    startFileTime = clock();

    processingTime.filename = inputFilename;

    std::ifstream inFile(inputFilename);

    if (!inFile.is\_open())

    {

        std::cerr << "Error: Couldn't open input file '" << inputFilename << "'." << std::endl;

        return;

    }

    std::ofstream outFile(outputFilename);

    if (!outFile.is\_open())

    {

        std::cerr << "Error: Couldn't open output file '" << outputFilename << "'." << std::endl;

        inFile.close();

        return;

    }

    std::string line;

    while (std::getline(inFile, line))

    {

        std::string processedLine = removeVowels(line);

        outFile << processedLine << '\n';

    }

    inFile.close();

    outFile.close();

    endFileTime = clock();

    processingTime.start\_time = startFileTime;

    processingTime.end\_time = endFileTime;

    fileProcessingTimes.push\_back(processingTime);

    std::cout << "Text processed and saved to '" << outputFilename << "'." << std::endl;

}

void writeToJSON(const std::string &filename)

{

    std::string outputFilename = ssdDirectoryOut + filename; // Add the directory path

    json jsonData;

    json programTimes;

    programTimes["start\_program\_time"] = static\_cast<double>(startTime) / CLOCKS\_PER\_SEC;

    programTimes["end\_program\_time"] = static\_cast<double>(endTime) / CLOCKS\_PER\_SEC;

    programTimes["elapse\_program\_time"] = static\_cast<double>(endTime - startTime) / CLOCKS\_PER\_SEC;

    jsonData["program\_times"] = programTimes;

    // Create a JSON array to hold file data

    json filesArray = json::array();

    for (const auto &processingTime : fileProcessingTimes)

    {

        json fileData;

        fileData["filename"] = processingTime.filename;

        fileData["start\_time\_seconds"] = static\_cast<double>(processingTime.start\_time) / CLOCKS\_PER\_SEC;

        fileData["end\_time\_seconds"] = static\_cast<double>(processingTime.end\_time) / CLOCKS\_PER\_SEC;

        fileData["elapse\_time\_seconds"] = static\_cast<double>(processingTime.end\_time - processingTime.start\_time) / CLOCKS\_PER\_SEC;

        filesArray.push\_back(fileData);

    }

    jsonData["file\_processing\_times"] = filesArray;

    std::ofstream outFile(outputFilename);

    if (outFile.is\_open())

    {

        outFile << std::setw(4) << jsonData << std::endl;

        outFile.close();

        std::cout << "Data written to JSON file '" << outputFilename << "'." << std::endl;

    }

    else

    {

        std::cerr << "Error writing to file '" << outputFilename << "'." << std::endl;

    }

}

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

{

    startTime = clock();

    if (argc < 2)

    {

        std::cerr << "Usage: " << argv[0] << " <filename1> [<filename2> ... <filenameN>]" << std::endl;

        return 1;

    }

    for (int i = 1; i < argc; ++i)

    {

        std::string inputFilename = std::string(ssdDirectoryIn) + argv[i];

        if (fileNamesSet.find(inputFilename) != fileNamesSet.end())

        {

            std::cout << "File '" << inputFilename << "' reoccurs. Skipping." << std::endl;

            continue;

        }

        fileNamesSet.insert(inputFilename);

        std::string outputFilename = std::string(ssdDirectoryOut) + "out\_" + argv[i];

        processFile(inputFilename, outputFilename);

    }

    endTime = clock();

    writeToJSON("file\_processing\_times\_a.json");

    return 0;

}

## **Программа B**

#include <windows.h>

#include <iostream>

#include <fstream>

#include <sstream>

#include <string>

#include <unordered\_set>

#include <cctype>

#include <vector>

#include <time.h>

#include "json.hpp"

using json = nlohmann::json;

struct FileProcessingTime

{

    std::string filename;

    clock\_t start\_time;

    clock\_t end\_time;

};

std::vector<FileProcessingTime> fileProcessingTimes;

std::unordered\_set<std::string> fileNamesSet;

clock\_t startTime;

clock\_t endTime;

CRITICAL\_SECTION cs;

std::string ssdDirectoryIn = "C:\\CodingProjects\\C++\\osisp\\v\\kr\\files\\in\\";

std::string ssdDirectoryOut = "C:\\CodingProjects\\C++\\osisp\\v\\kr\\files\\out\\";

std::string flashDirectoryIn = "E:\\files\\in\\";

std::string flashDirectoryOut = "E:\\files\\out\\";

bool isVowel(char c)

{

    c = std::tolower(c);

    return (c == 'a' || c == 'e' || c == 'i' || c == 'o' || c == 'u' ||

            c == '?' || c == '?' || c == '?' || c == '?' || c == '?' ||

            c == '?' || c == '?' || c == '?' || c == '?' || c == '?');

}

// Function to remove vowels from a string

std::string removeVowels(const std::string &text)

{

    std::string result;

    for (char c : text)

    {

        if (!isVowel(c))

        {

            result += c;

        }

    }

    return result;

}

DWORD WINAPI processFileThread(LPVOID lpParam)

{

    std::pair<std::string, std::string> \*filePair = reinterpret\_cast<std::pair<std::string, std::string> \*>(lpParam);

    std::string inputFilename = filePair->first;

    std::string outputFilename = filePair->second;

    delete filePair;

    FileProcessingTime processingTime;

    clock\_t startFileTime, endFileTime;

    startFileTime = clock();

    processingTime.filename = inputFilename;

    std::ifstream inFile(inputFilename);

    if (!inFile.is\_open())

    {

        std::cerr << "Error: Couldn't open input file '" << inputFilename << "'." << std::endl;

        return 1;

    }

    std::ofstream outFile(outputFilename);

    if (!outFile.is\_open())

    {

        std::cerr << "Error: Couldn't open output file '" << outputFilename << "'." << std::endl;

        inFile.close();

        return 1;

    }

    std::string line;

    while (std::getline(inFile, line))

    {

        std::string processedLine = removeVowels(line);

        outFile << processedLine << '\n';

    }

    inFile.close();

    outFile.close();

    endFileTime = clock();

    processingTime.start\_time = startFileTime;

    processingTime.end\_time = endFileTime;

    EnterCriticalSection(&cs);

    fileProcessingTimes.push\_back(processingTime);

    LeaveCriticalSection(&cs);

    std::cout << "Text processed and saved to '" << outputFilename << "'." << std::endl;

    return 0;

}

void writeToJSON(const std::string &filename)

{

    std::string outputFilename = ssdDirectoryOut + filename; // Add the directory path

    json jsonData;

    json programTimes;

    programTimes["start\_program\_time"] = static\_cast<double>(startTime) / CLOCKS\_PER\_SEC;

    programTimes["end\_program\_time"] = static\_cast<double>(endTime) / CLOCKS\_PER\_SEC;

    programTimes["elapse\_program\_time"] = static\_cast<double>(endTime - startTime) / CLOCKS\_PER\_SEC;

    jsonData["program\_times"] = programTimes;

    // Create a JSON array to hold file data

    json filesArray = json::array();

    for (const auto &processingTime : fileProcessingTimes)

    {

        json fileData;

        fileData["filename"] = processingTime.filename;

        fileData["start\_time\_seconds"] = static\_cast<double>(processingTime.start\_time) / CLOCKS\_PER\_SEC;

        fileData["end\_time\_seconds"] = static\_cast<double>(processingTime.end\_time) / CLOCKS\_PER\_SEC;

        fileData["elapse\_time\_seconds"] = static\_cast<double>(processingTime.end\_time - processingTime.start\_time) / CLOCKS\_PER\_SEC;

        filesArray.push\_back(fileData);

    }

    jsonData["file\_processing\_times"] = filesArray;

    std::ofstream outFile(outputFilename);

    if (outFile.is\_open())

    {

        outFile << std::setw(4) << jsonData << std::endl;

        outFile.close();

        std::cout << "Data written to JSON file '" << outputFilename << "'." << std::endl;

    }

    else

    {

        std::cerr << "Error writing to file '" << outputFilename << "'." << std::endl;

    }

}

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

{

    startTime = clock();

    InitializeCriticalSection(&cs);

    if (argc < 2)

    {

        std::cerr << "Usage: " << argv[0] << " <filename1> [<filename2> ... <filenameN>]" << std::endl;

        return 1;

    }

    std::vector<HANDLE> threadHandles;

    for (int i = 1; i < argc; ++i)

    {

        std::string inputFilename = std::string(ssdDirectoryIn) + argv[i];

        if (fileNamesSet.find(inputFilename) != fileNamesSet.end())

        {

            std::cout << "File '" << inputFilename << "' reoccurs. Skipping." << std::endl;

            continue;

        }

        fileNamesSet.insert(inputFilename);

        std::string outputFilename = std::string(ssdDirectoryOut) + "out\_" + argv[i];

        std::pair<std::string, std::string> \*filePair = new std::pair<std::string, std::string>(inputFilename, outputFilename);

        HANDLE hThread = CreateThread(NULL, 0, processFileThread, filePair, 0, NULL);

        if (hThread == NULL)

        {

            std::cerr << "Error creating thread for file '" << inputFilename << "'." << std::endl;

            delete filePair;

        }

        else

        {

            threadHandles.push\_back(hThread);

        }

    }

    WaitForMultipleObjects(threadHandles.size(), threadHandles.data(), TRUE, INFINITE);

    for (auto &hThread : threadHandles)

    {

        CloseHandle(hThread);

    }

    endTime = clock();

    writeToJSON("file\_processing\_times\_b.json");

    return 0;

}

## **Программа C**

#include <windows.h>

#include <iostream>

#include <fstream>

#include <sstream>

#include <string>

#include <unordered\_set>

#include <cctype>

#include <vector>

#include <time.h>

#include "json.hpp"

using json = nlohmann::json;

struct FileProcessingTime

{

    std::string filename;

    clock\_t start\_time;

    clock\_t end\_time;

};

std::vector<FileProcessingTime> fileProcessingTimes;

std::unordered\_set<std::string> fileNamesSet;

clock\_t startTime;

clock\_t endTime;

CRITICAL\_SECTION cs;

std::string ssdDirectoryIn = "C:\\CodingProjects\\C++\\osisp\\v\\kr\\files\\in\\";

std::string ssdDirectoryOut = "C:\\CodingProjects\\C++\\osisp\\v\\kr\\files\\out\\";

std::string flashDirectoryIn = "E:\\files\\in\\";

std::string flashDirectoryOut = "E:\\files\\out\\";

bool isVowel(char c)

{

    c = std::tolower(c);

    return (c == 'a' || c == 'e' || c == 'i' || c == 'o' || c == 'u' ||

            c == '?' || c == '?' || c == '?' || c == '?' || c == '?' ||

            c == '?' || c == '?' || c == '?' || c == '?' || c == '?');

}

// Function to remove vowels from a string

std::string removeVowels(const std::string &text)

{

    std::string result;

    for (char c : text)

    {

        if (!isVowel(c))

        {

            result += c;

        }

    }

    return result;

}

DWORD WINAPI processFileThread(LPVOID lpParam)

{

    std::pair<std::string, std::string> \*filePair = reinterpret\_cast<std::pair<std::string, std::string> \*>(lpParam);

    std::string inputFilename = filePair->first;

    std::string outputFilename = filePair->second;

    delete filePair;

    FileProcessingTime processingTime;

    clock\_t startFileTime, endFileTime;

    startFileTime = clock();

    processingTime.filename = inputFilename;

    std::ifstream inFile(inputFilename);

    if (!inFile.is\_open())

    {

        std::cerr << "Error: Couldn't open input file '" << inputFilename << "'." << std::endl;

        return 1;

    }

    std::ofstream outFile(outputFilename);

    if (!outFile.is\_open())

    {

        std::cerr << "Error: Couldn't open output file '" << outputFilename << "'." << std::endl;

        inFile.close();

        return 1;

    }

    std::string line;

    while (std::getline(inFile, line))

    {

        std::string processedLine = removeVowels(line);

        outFile << processedLine << '\n';

    }

    inFile.close();

    outFile.close();

    endFileTime = clock();

    processingTime.start\_time = startFileTime;

    processingTime.end\_time = endFileTime;

    EnterCriticalSection(&cs);

    fileProcessingTimes.push\_back(processingTime);

    LeaveCriticalSection(&cs);

    std::cout << "Text processed and saved to '" << outputFilename << "'." << std::endl;

    return 0;

}

void writeToJSON(const std::string &filename)

{

    std::string outputFilename = ssdDirectoryOut + filename; // Add the directory path

    json jsonData;

    json programTimes;

    programTimes["start\_program\_time"] = static\_cast<double>(startTime) / CLOCKS\_PER\_SEC;

    programTimes["end\_program\_time"] = static\_cast<double>(endTime) / CLOCKS\_PER\_SEC;

    programTimes["elapse\_program\_time"] = static\_cast<double>(endTime - startTime) / CLOCKS\_PER\_SEC;

    jsonData["program\_times"] = programTimes;

    // Create a JSON array to hold file data

    json filesArray = json::array();

    for (const auto &processingTime : fileProcessingTimes)

    {

        json fileData;

        fileData["filename"] = processingTime.filename;

        fileData["start\_time\_seconds"] = static\_cast<double>(processingTime.start\_time) / CLOCKS\_PER\_SEC;

        fileData["end\_time\_seconds"] = static\_cast<double>(processingTime.end\_time) / CLOCKS\_PER\_SEC;

        fileData["elapse\_time\_seconds"] = static\_cast<double>(processingTime.end\_time - processingTime.start\_time) / CLOCKS\_PER\_SEC;

        filesArray.push\_back(fileData);

    }

    jsonData["file\_processing\_times"] = filesArray;

    std::ofstream outFile(outputFilename);

    if (outFile.is\_open())

    {

        outFile << std::setw(4) << jsonData << std::endl;

        outFile.close();

        std::cout << "Data written to JSON file '" << outputFilename << "'." << std::endl;

    }

    else

    {

        std::cerr << "Error writing to file '" << outputFilename << "'." << std::endl;

    }

}

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

{

    startTime = clock();

    InitializeCriticalSection(&cs);

    if (argc < 2)

    {

        std::cerr << "Usage: " << argv[0] << " <filename1> [<filename2> ... <filenameN>]" << std::endl;

        return 1;

    }

    std::vector<HANDLE> threadHandles;

    for (int i = 1; i < argc; ++i)

    {

        std::string inputFilename = std::string(ssdDirectoryIn) + argv[i];

        if (fileNamesSet.find(inputFilename) != fileNamesSet.end())

        {

            std::cout << "File '" << inputFilename << "' reoccurs. Skipping." << std::endl;

            continue;

        }

        fileNamesSet.insert(inputFilename);

        std::string outputFilename = std::string(ssdDirectoryOut) + "out\_" + argv[i];

        std::pair<std::string, std::string> \*filePair = new std::pair<std::string, std::string>(inputFilename, outputFilename);

        HANDLE hThread = CreateThread(NULL, 0, processFileThread, filePair, CREATE\_SUSPENDED, NULL);

        if (hThread == NULL)

        {

            std::cerr << "Error creating thread for file '" << inputFilename << "'." << std::endl;

            delete filePair;

        }

        else

        {

            if (!SetThreadPriority(hThread, THREAD\_PRIORITY\_ABOVE\_NORMAL))

            {

                std::cerr << "Failed to set thread priority." << std::endl;

            }

            ResumeThread(hThread);

            threadHandles.push\_back(hThread);

        }

    }

    WaitForMultipleObjects(threadHandles.size(), threadHandles.data(), TRUE, INFINITE);

    for (auto &hThread : threadHandles)

    {

        CloseHandle(hThread);

    }

    endTime = clock();

    writeToJSON("file\_processing\_times\_c.json");

    return 0;

# РЕЗУЛЬТАТ РАБОТЫ ПРОГРАММ

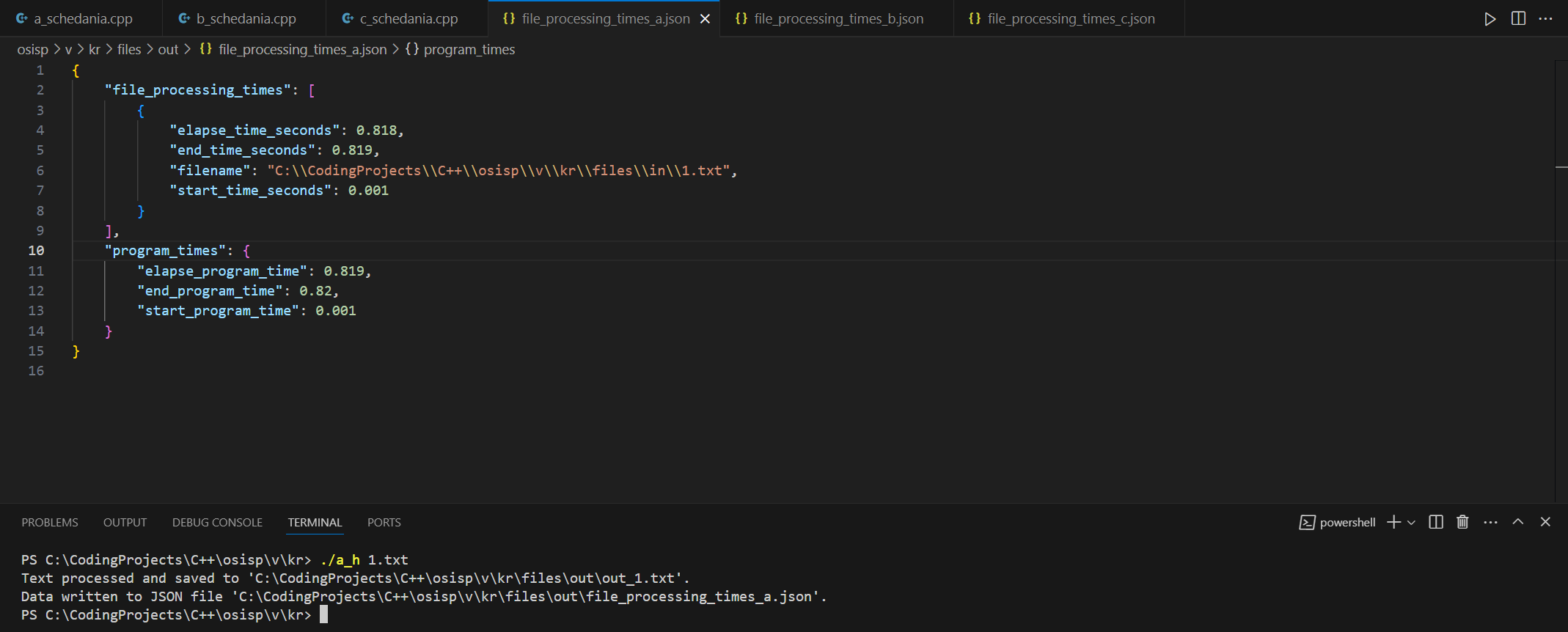


Рисунок 1 – Работа программы AH с 1 текстовым файлом

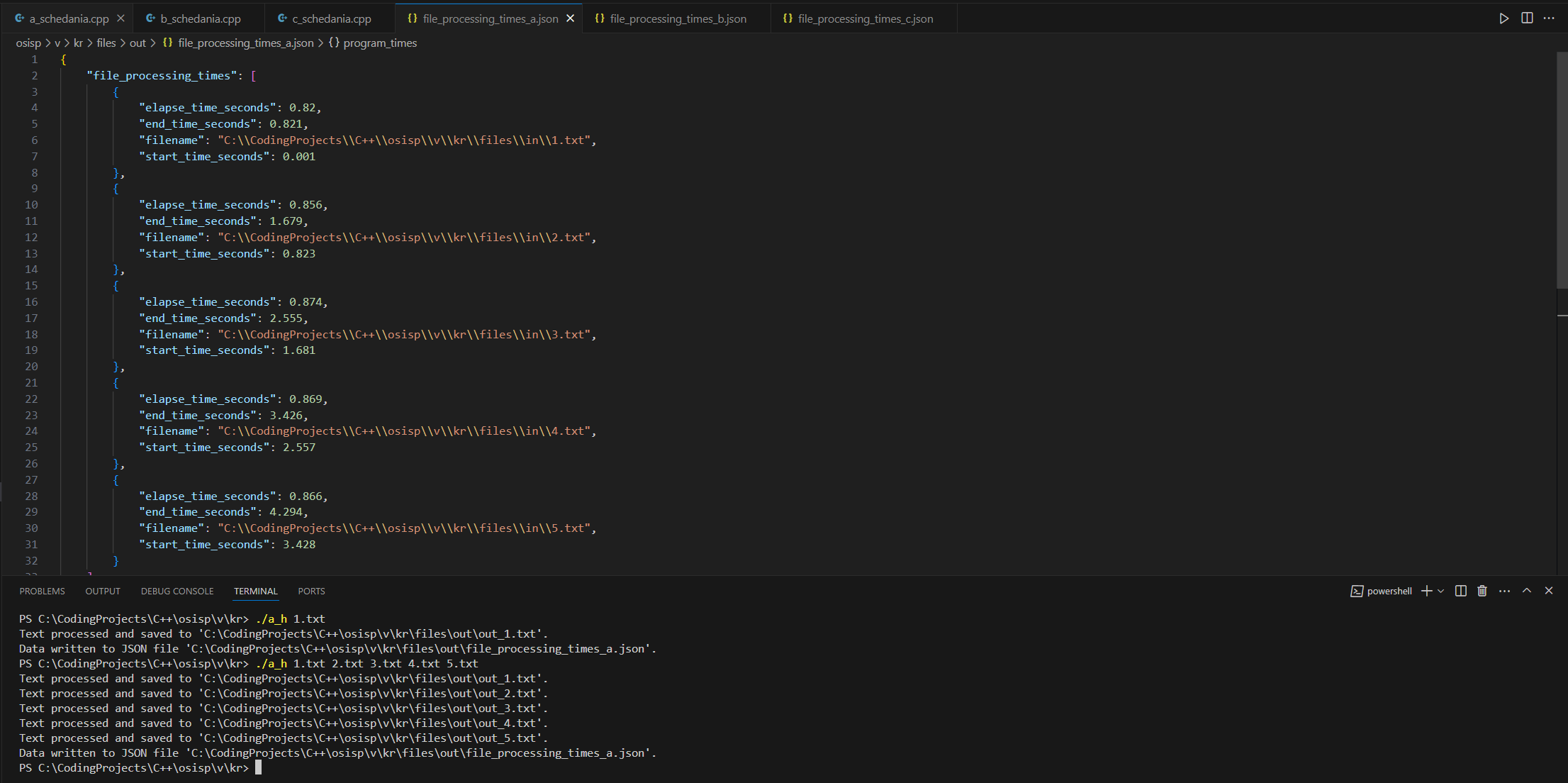


Рисунок 2 – Работа программы AH с 5 текстовыми файлами

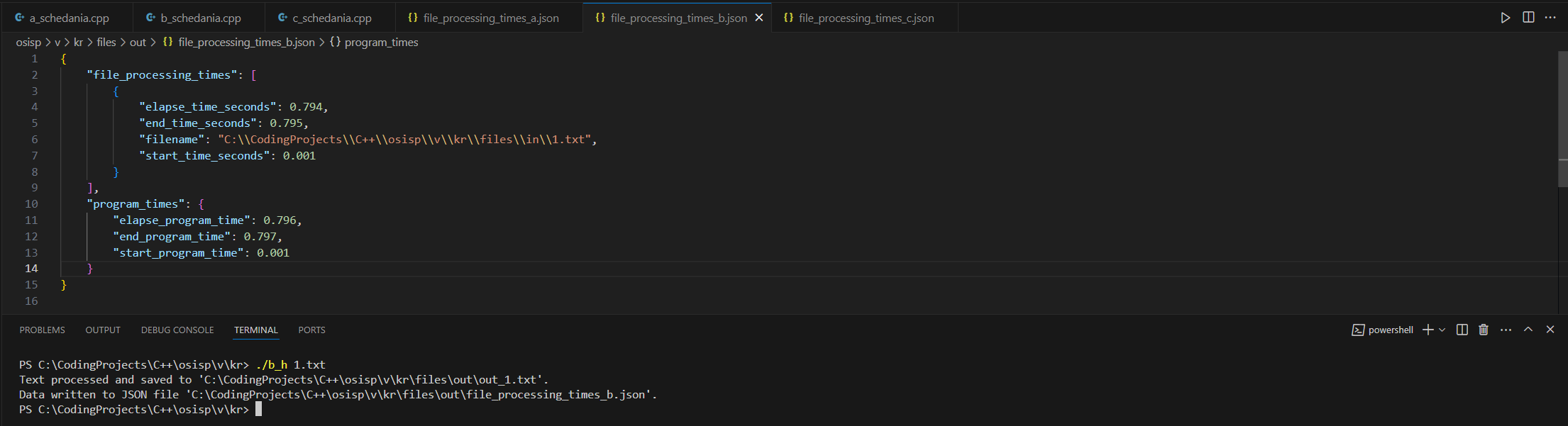


Рисунок 3 – Работа программы BH с 1 текстовым файлом

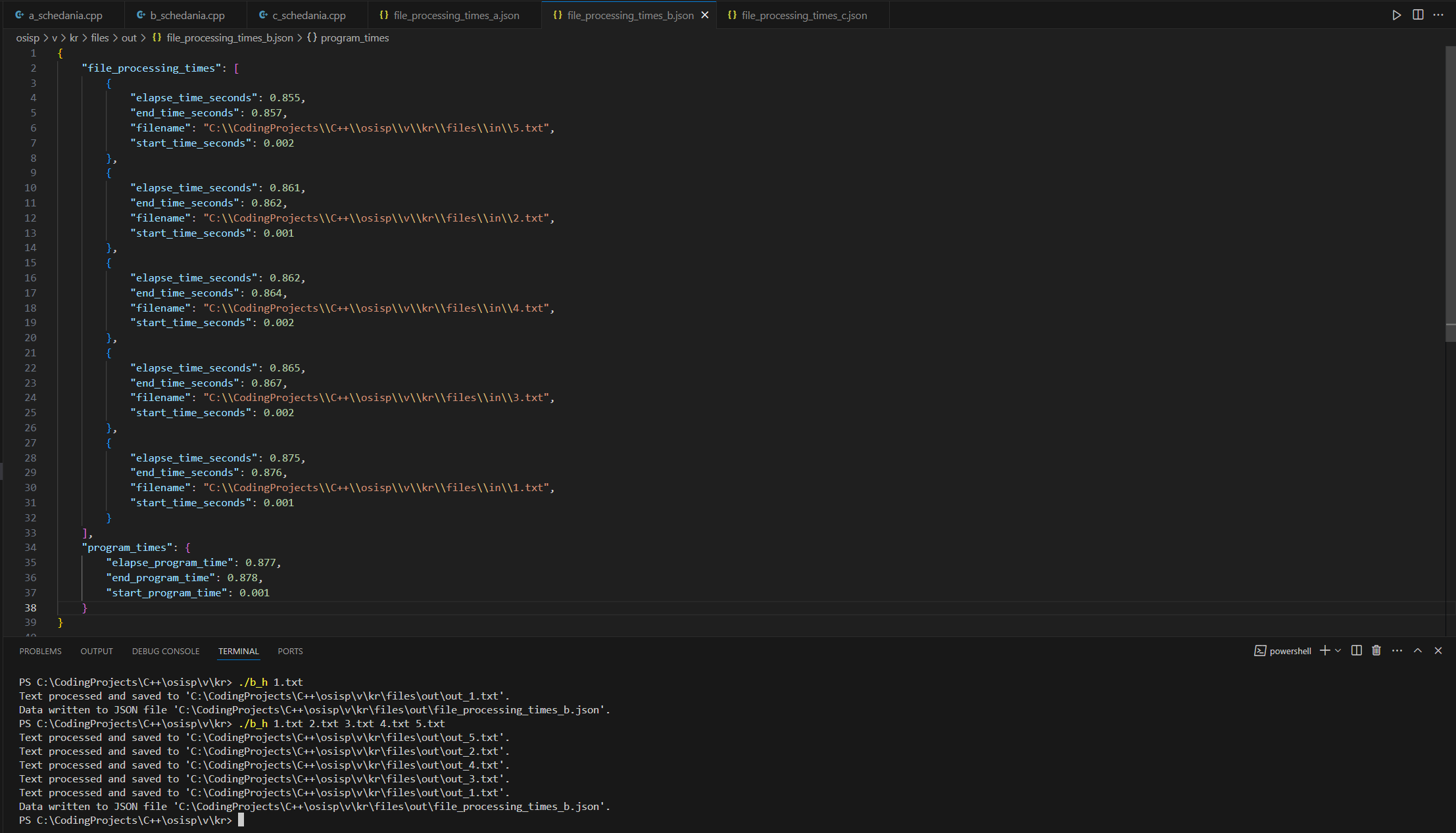


Рисунок 4 – Работа программы BH с 5 текстовыми файлами



Рисунок 5 – Работа программы СH с 1 текстовым файлом

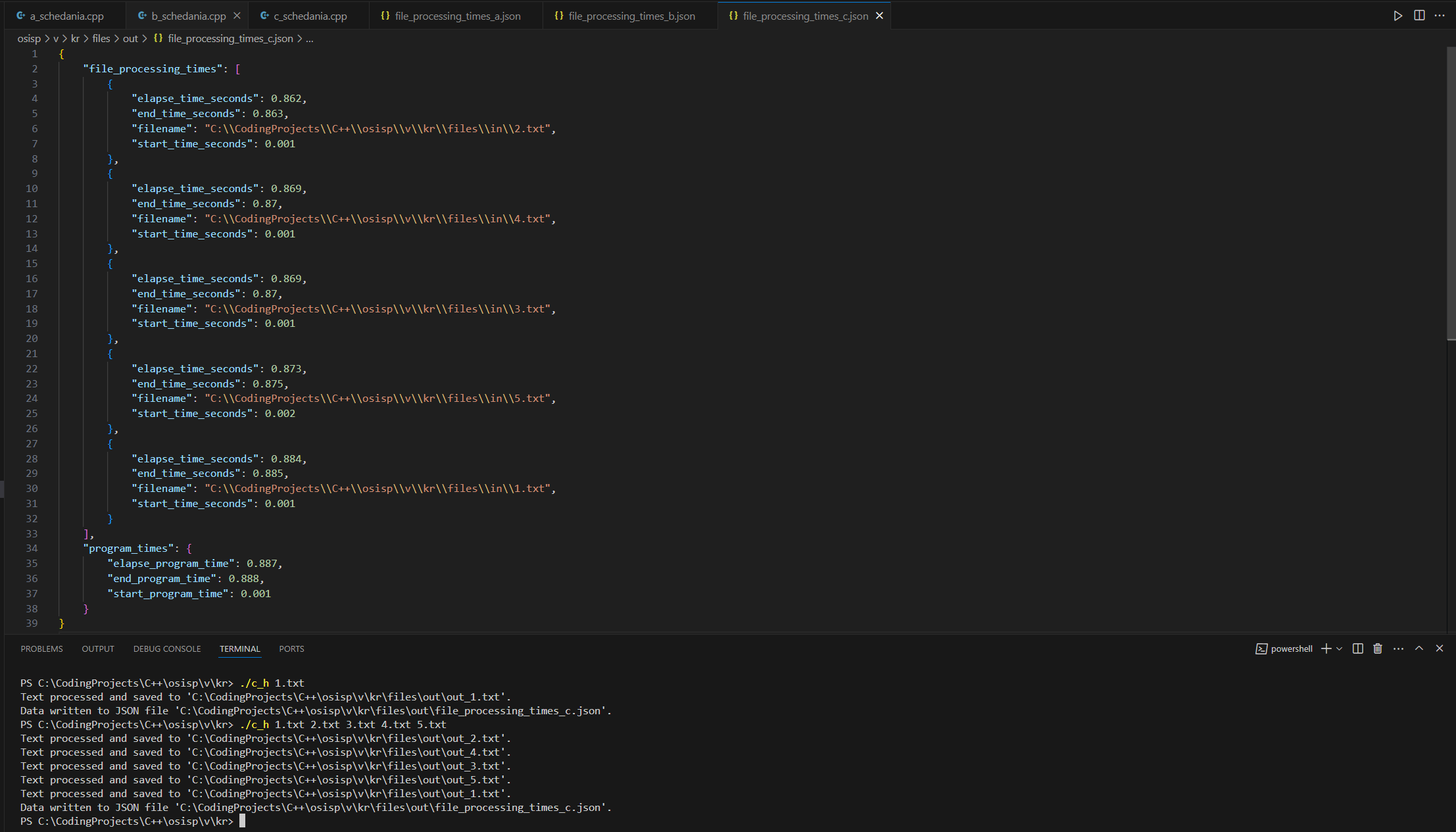


Рисунок 6– Работа программы CH с 5 текстовыми файлами

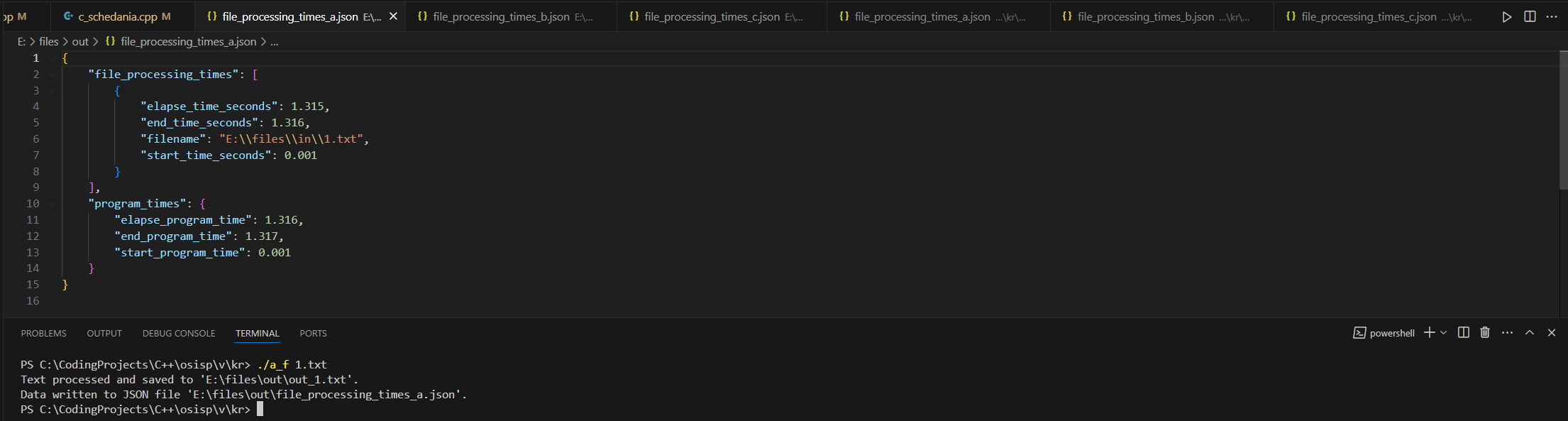


Рисунок 7 – Работа программы AF с 1 текстовым файлом

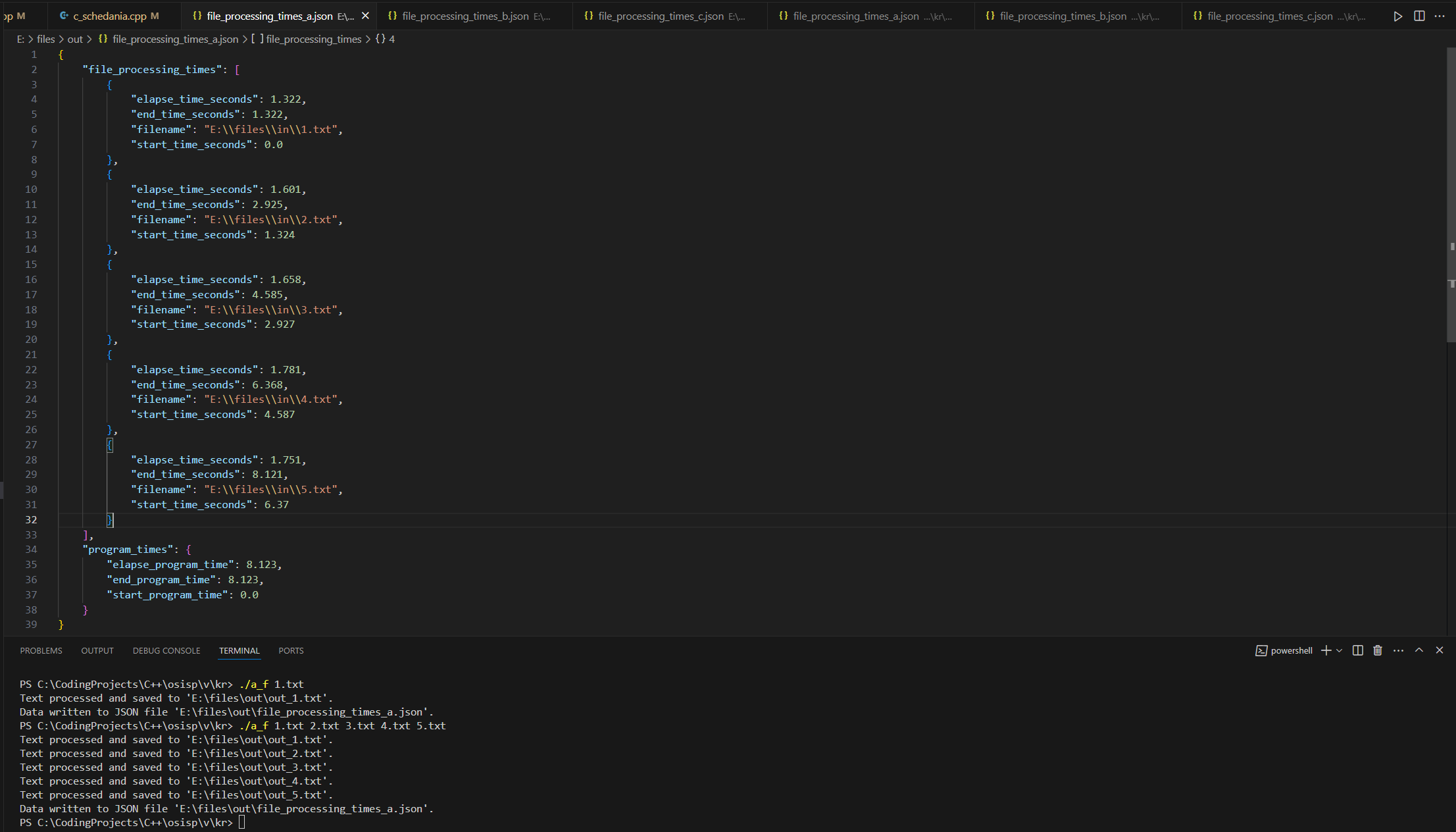


Рисунок 8 – Работа программы AF с 5 текстовыми файлами

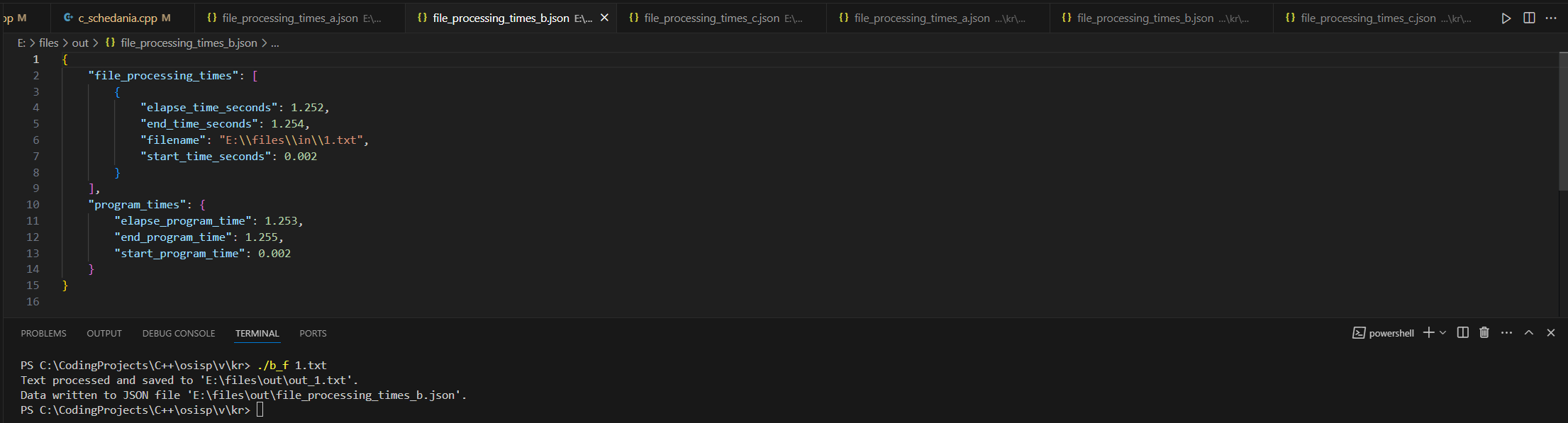


Рисунок 9 – Работа программы BF с 1 текстовым файлом

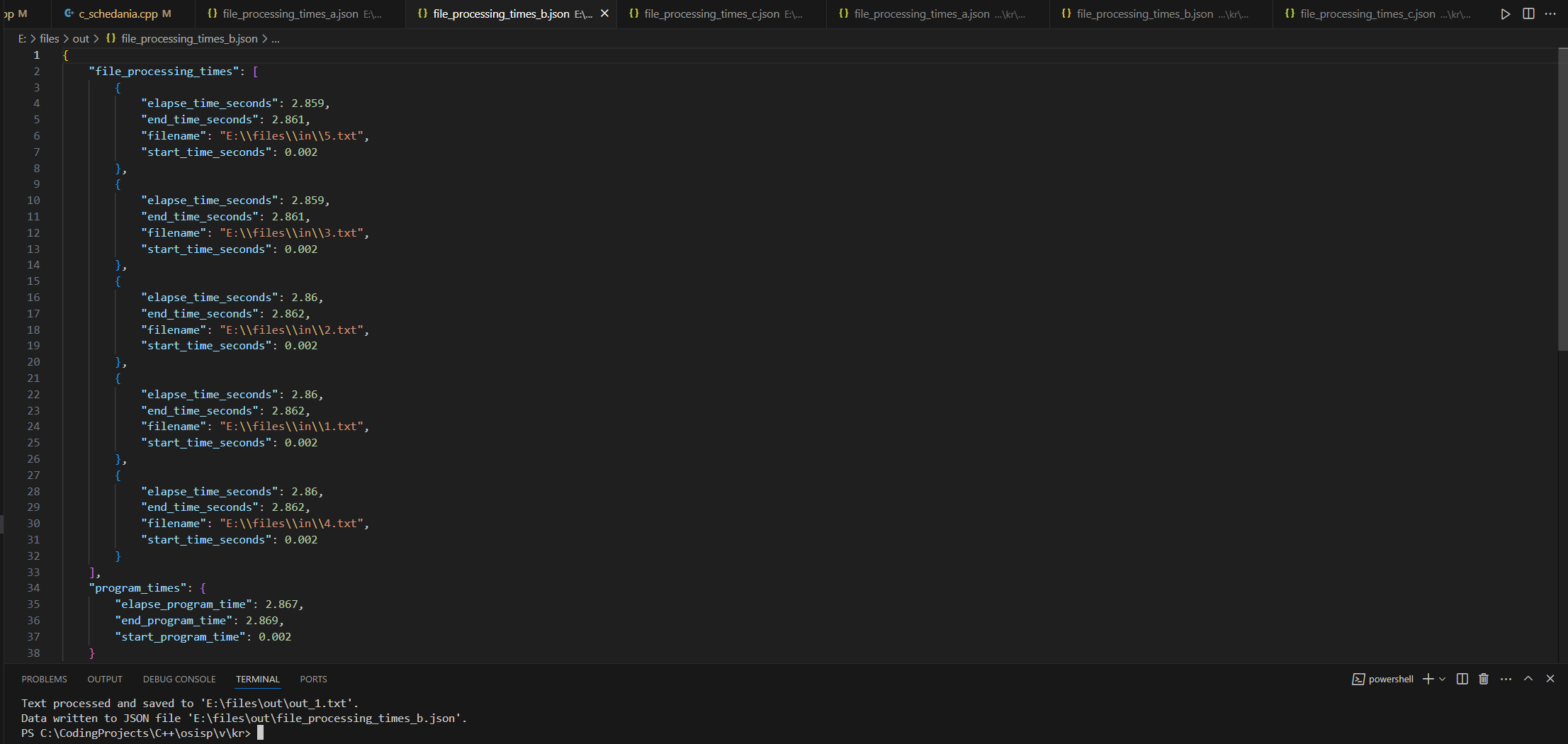


Рисунок 10 – Работа программы BF с 5 текстовыми файлами

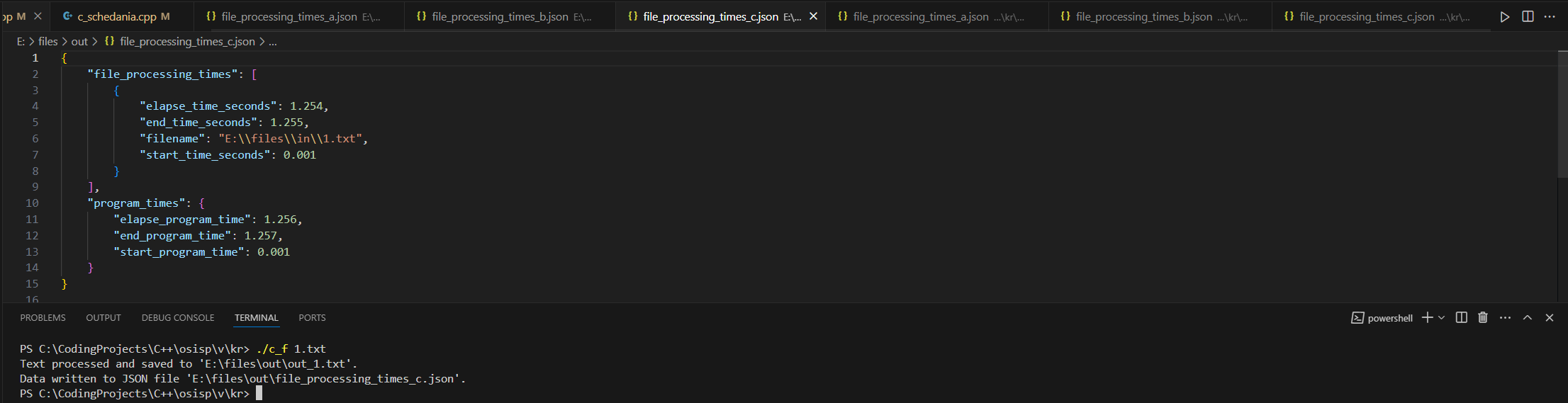


Рисунок 11 – Работа программы CF с 1 текстовым файлом

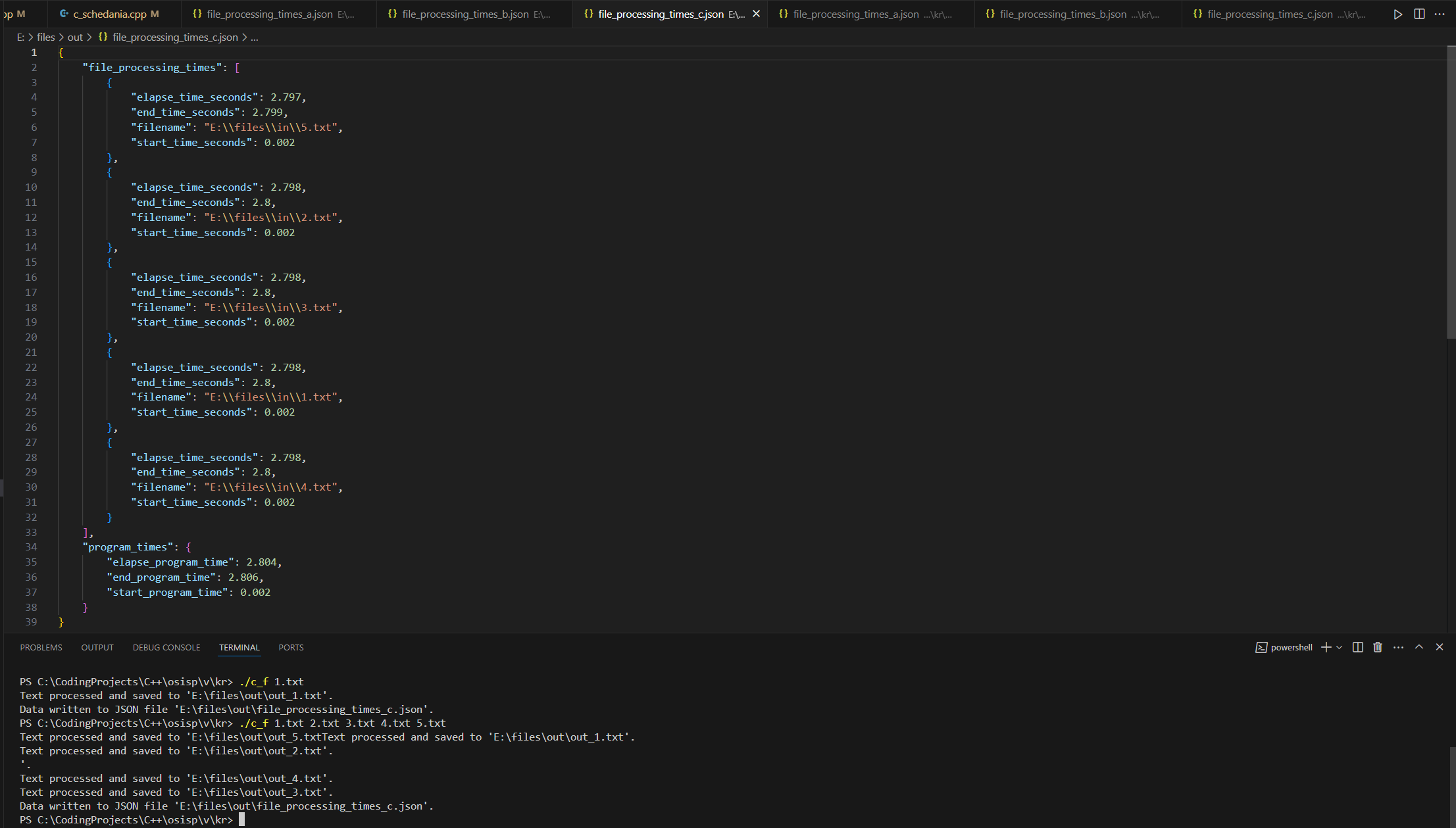


Рисунок 12 – Работа программы CF с 5 текстовыми файлами

# СКРИНШОТЫ ОКНА УТИЛИТЫ PROCESS EXPLORER

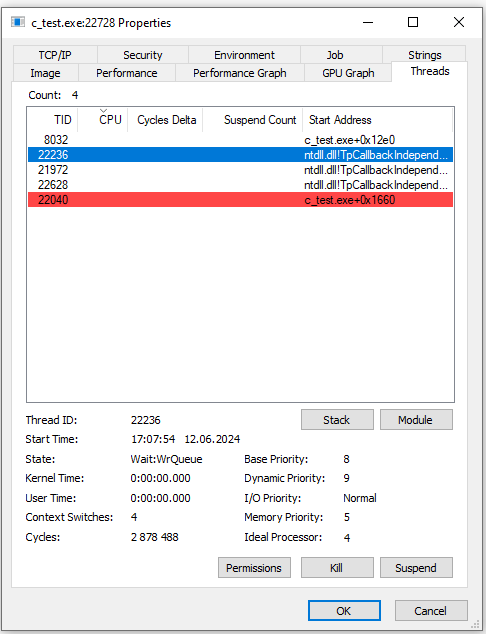


Рисунок – Дочерние потоки программы С

**Base Priority (Базовый приоритет)**: Это начальный приоритет потока, установленный при его создании. Он может варьироваться от 1 (самый низкий) до 31 (самый высокий) для пользовательских процессов в Windows. Базовый приоритет потока определяется на основе приоритета процесса, к которому он принадлежит, и может быть изменен функциями управления задачами. В данном случае базовый приоритет равен 8, что соответствует уровню "Normal" (нормальный).

**Dynamic Priority (Динамический приоритет)**: Это текущий приоритет потока, который может изменяться системой в зависимости от различных факторов, таких как использование процессора, наличие ввода/вывода и другие. Система Windows может временно повышать или понижать приоритеты потоков для оптимизации работы. Например, если поток долгое время не получал процессорного времени, его приоритет может быть временно повышен. В данном случае динамический приоритет равен 9, что означает, что система временно повысила приоритет потока для улучшения производительности.

# ОПИСАНИЕ И РЕЗУЛЬТАТЫ ПРОВЕДЕННОГО ЭКСПЕРИМЕНТА

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Режим | N | t1 | t2 | t3 | t4 | t5 | tср |
| AH | 1 | 878 | 896 | 879 | 877 | 877 | 881,4 |
|  | 2 | 1755 | 1758 | 1767 | 1756 | 1755 | 1758,2 |
|  | 3 | 2643 | 2,628 | 2633 | 2634 | 2644 | 2111,3 |
|  | 4 | 3512 | 3527 | 3517 | 3524 | 3544 | 3524,8 |
|  | 5 | 4389 | 4433 | 4395 | 4402 | 4389 | 4401,6 |
|  | 6 | 5300 | 5273 | 5285 | 5276 | 5294 | 5285,6 |
|  | 7 | 6172 | 6172 | 6145 | 6161 | 6155 | 6161 |
|  | 8 | 7050 | 7048 | 7041 | 7039 | 7052 | 7046 |
|  | 9 | 7906 | 7918 | 7912 | 7910 | 7918 | 7912,8 |
|  | 10 | 8815 | 8825 | 8822 | 8821 | 8791 | 8814,8 |
| BH | 1 | 889 | 888 | 863 | 863 | 863 | 873,2 |
|  | 2 | 870 | 880 | 889 | 882 | 903 | 884,8 |
|  | 3 | 869 | 865 | 892 | 885 | 870 | 876,2 |
|  | 4 | 881 | 866 | 872 | 859 | 883 | 872,2 |
|  | 5 | 880 | 880 | 877 | 868 | 880 | 877 |
|  | 6 | 928 | 895 | 881 | 885 | 877 | 893,2 |
|  | 7 | 921 | 907 | 910 | 888 | 909 | 907 |
|  | 8 | 927 | 932 | 943 | 899 | 926 | 925,4 |
|  | 9 | 954 | 966 | 991 | 942 | 969 | 964,4 |
|  | 10 | 1033 | 979 | 976 | 984 | 961 | 986,6 |
| CH | 1 | 888 | 890 | 890 | 865 | 895 | 885,6 |
|  | 2 | 884 | 870 | 903 | 876 | 887 | 884 |
|  | 3 | 871 | 892 | 887 | 866 | 875 | 878,2 |
|  | 4 | 887 | 885 | 898 | 874 | 885 | 885,8 |
|  | 5 | 925 | 899 | 878 | 884 | 881 | 893,4 |
|  | 6 | 926 | 885 | 885 | 885 | 879 | 892 |
|  | 7 | 897 | 894 | 896 | 940 | 898 | 905 |
|  | 8 | 921 | 922 | 901 | 896 | 923 | 912,6 |
|  | 9 | 925 | 894 | 915 | 904 | 926 | 912,8 |
|  | 10 | 1007 | 929 | 947 | 1006 | 914 | 960,6 |
| AF | 1 | 1228 | 1350 | 1285 | 1400 | 1320 | 1316,6 |
|  | 2 | 2493 | 2560 | 2600 | 2435 | 2510 | 2519,6 |
|  | 3 | 3469 | 3600 | 3500 | 3555 | 3480 | 3520,8 |
|  | 4 | 4938 | 4820 | 5050 | 4975 | 4900 | 4936,6 |
|  | 5 | 6028 | 6100 | 5930 | 6075 | 6000 | 6026,6 |
|  | 6 | 7271 | 7350 | 7200 | 7300 | 7250 | 7274,2 |
|  | 7 | 8424 | 8500 | 8370 | 8455 | 8400 | 8429,8 |
|  | 8 | 9572 | 9650 | 9480 | 9550 | 9600 | 9570,4 |
|  | 9 | 10698 | 10800 | 10700 | 10650 | 10750 | 10720 |
|  | 10 | 11884 | 12000 | 11900 | 11850 | 11950 | 11917 |
| BF | 1 | 1240 | 1300 | 1255 | 1280 | 1245 | 1264 |
|  | 2 | 1567 | 1580 | 1550 | 1600 | 1575 | 1574,4 |
|  | 3 | 1769 | 1800 | 1750 | 1780 | 1765 | 1772,8 |
|  | 4 | 2261 | 2280 | 2250 | 2275 | 2265 | 2266,2 |
|  | 5 | 2548 | 2560 | 2530 | 2555 | 2540 | 2546,6 |
|  | 6 | 3109 | 3 150 | 3120 | 3100 | 3115 | 3118,8 |
|  | 7 | 4173 | 4200 | 4150 | 4180 | 4165 | 4173,6 |
|  | 8 | 5200 | 5250 | 5220 | 5200 | 5215 | 5217 |
|  | 9 | 6222 | 6300 | 6250 | 6210 | 6235 | 6243,4 |
|  | 10 | 8113 | 8200 | 8120 | 8150 | 8100 | 8136,6 |
| CF | 1 | 1269 | 1300 | 1270 | 1250 | 1285 | 1274,8 |
|  | 2 | 1677 | 1700 | 1650 | 1685 | 1670 | 1676,4 |
|  | 3 | 2109 | 2150 | 2120 | 2100 | 2115 | 2118,8 |
|  | 4 | 2750 | 2780 | 2730 | 2765 | 2755 | 2756 |
|  | 5 | 4392 | 4420 | 4370 | 4400 | 4385 | 4393,4 |
|  | 6 | 4794 | 4820 | 4780 | 4800 | 4790 | 4796,8 |
|  | 7 | 5764 | 5800 | 5750 | 5780 | 5765 | 5771,8 |
|  | 8 | 8028 | 8050 | 8000 | 8030 | 8025 | 8026,6 |
|  | 9 | 8774 | 8800 | 8750 | 8780 | 8770 | 8774,8 |
|  | 10 | 10068 | 10100 | 10050 | 10020 | 10080 | 10064 |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Режим | Файлы на жестком диске (H) | | | | | | | | | |
| N=1 | N=2 | N=3 | N=4 | N=5 | N=6 | N=7 | N=8 | N=9 | N=10 |
| A | 881,4 | 1758,2 | 2111,3 | 3524,8 | 4401,6 | 5285,6 | 6161 | 7046 | 7912,8 | 8814,8 |
| B | 873,2 | 884,8 | 876,2 | 872,2 | 877 | 893,2 | 907 | 925,4 | 964,4 | 986,6 |
| C | 885,6 | 884 | 878,2 | 885,8 | 893,4 | 892 | 905 | 912,6 | 912,8 | 960,6 |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Файлы на flash-накопителе (F) | | | | | | | | | |
| N=1 | N=2 | N=3 | N=4 | N=5 | N=6 | N=7 | N=8 | N=9 | N=10 |
| 1316,6 | 2519,6 | 3520,8 | 4936,6 | 6026,6 | 7274,2 | 8429,8 | 9570,4 | 10720 | 11917 |
| 1264 | 1574,4 | 1772,8 | 2266,2 | 2546,6 | 3118,8 | 4173,6 | 5217 | 6243,4 | 8136,6 |
| 1274,8 | 1676,4 | 2118,8 | 2756 | 4393,4 | 4796,8 | 5771,8 | 8026,6 | 8774,8 | 10064 |

# ДИАГРАММЫ ТРАСС ПОТОКОВ

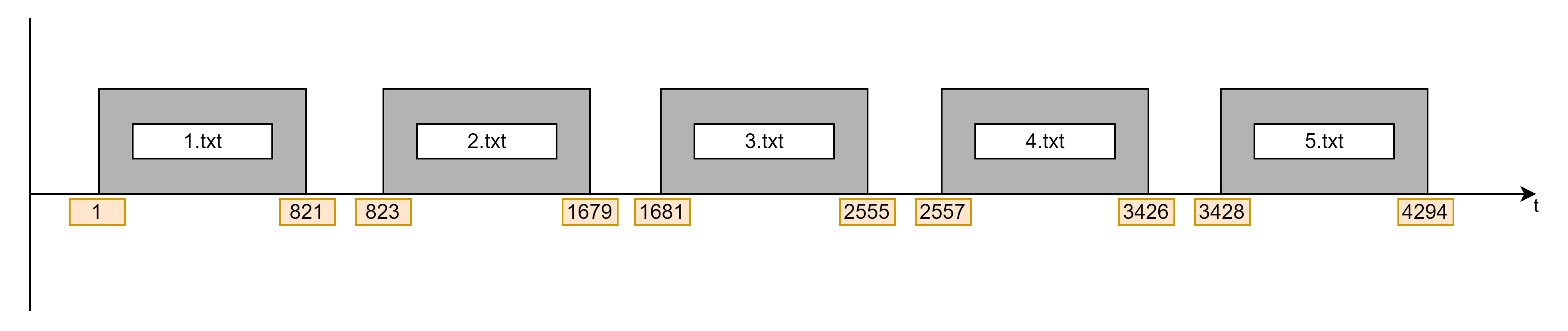


Рисунок 13 – Диаграмма трасс потоков программы AH с 5 текстовыми файлами

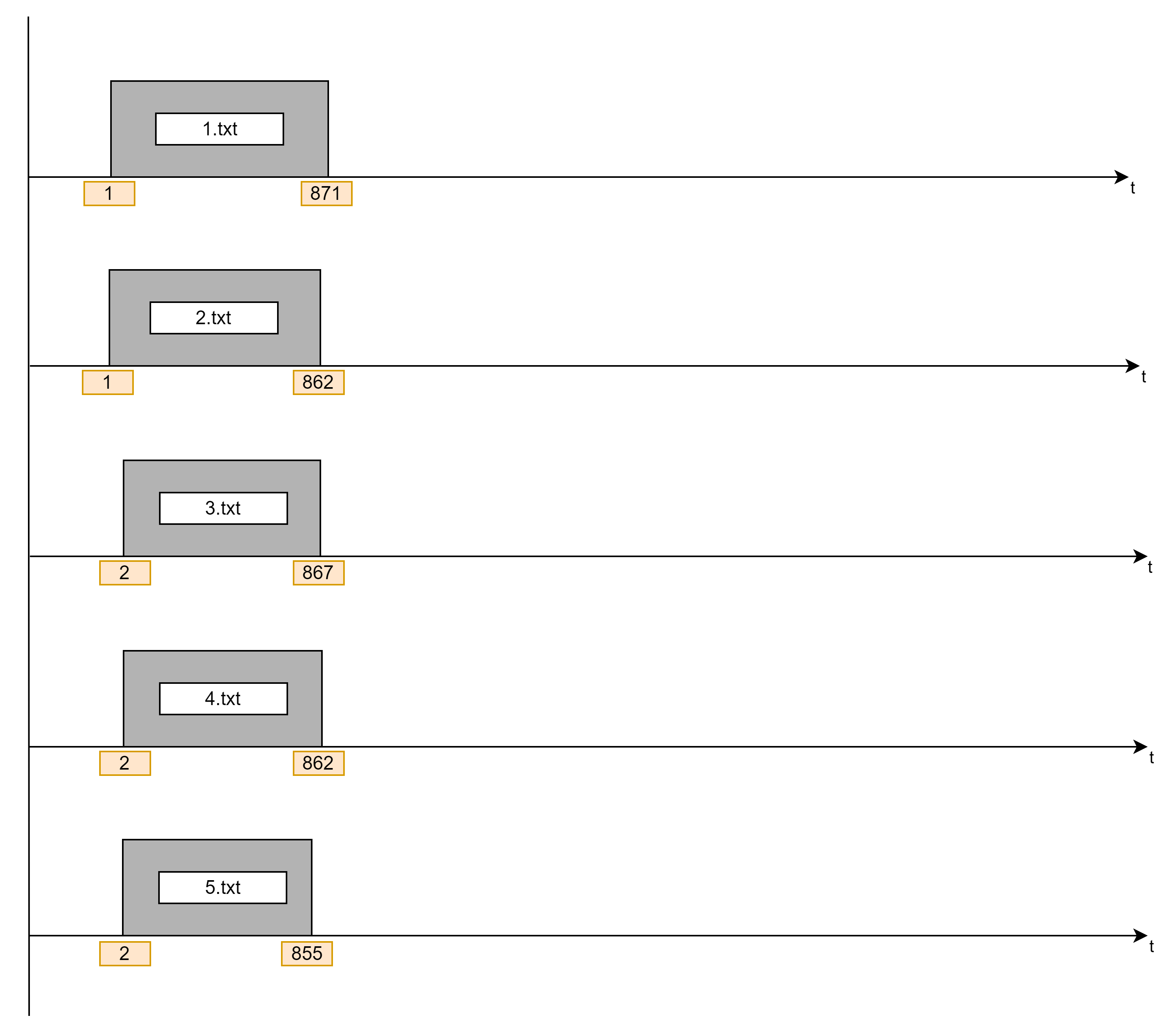


Рисунок 14 – Диаграмма трасс потоков программы BH с 5 текстовыми файлами

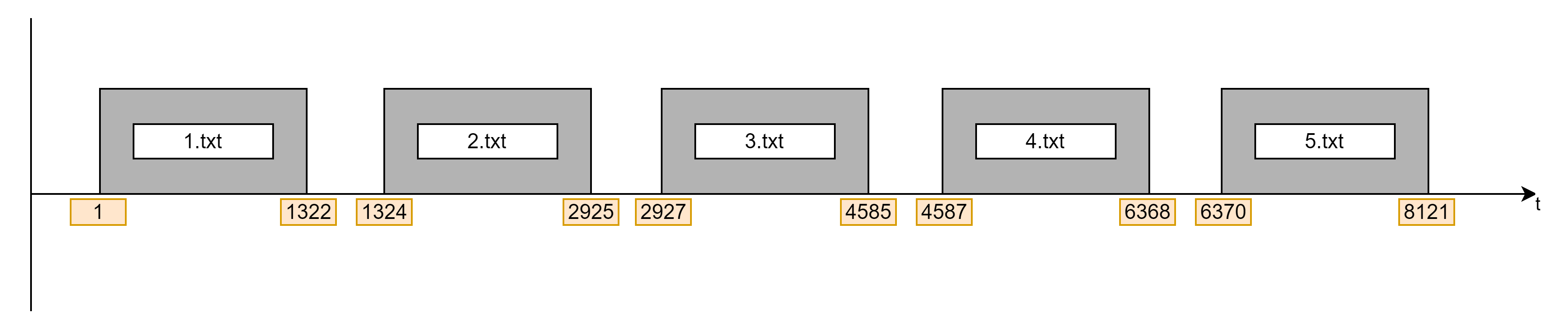


Рисунок 14 – Диаграмма трасс потоков программы AF с 5 текстовыми файлами

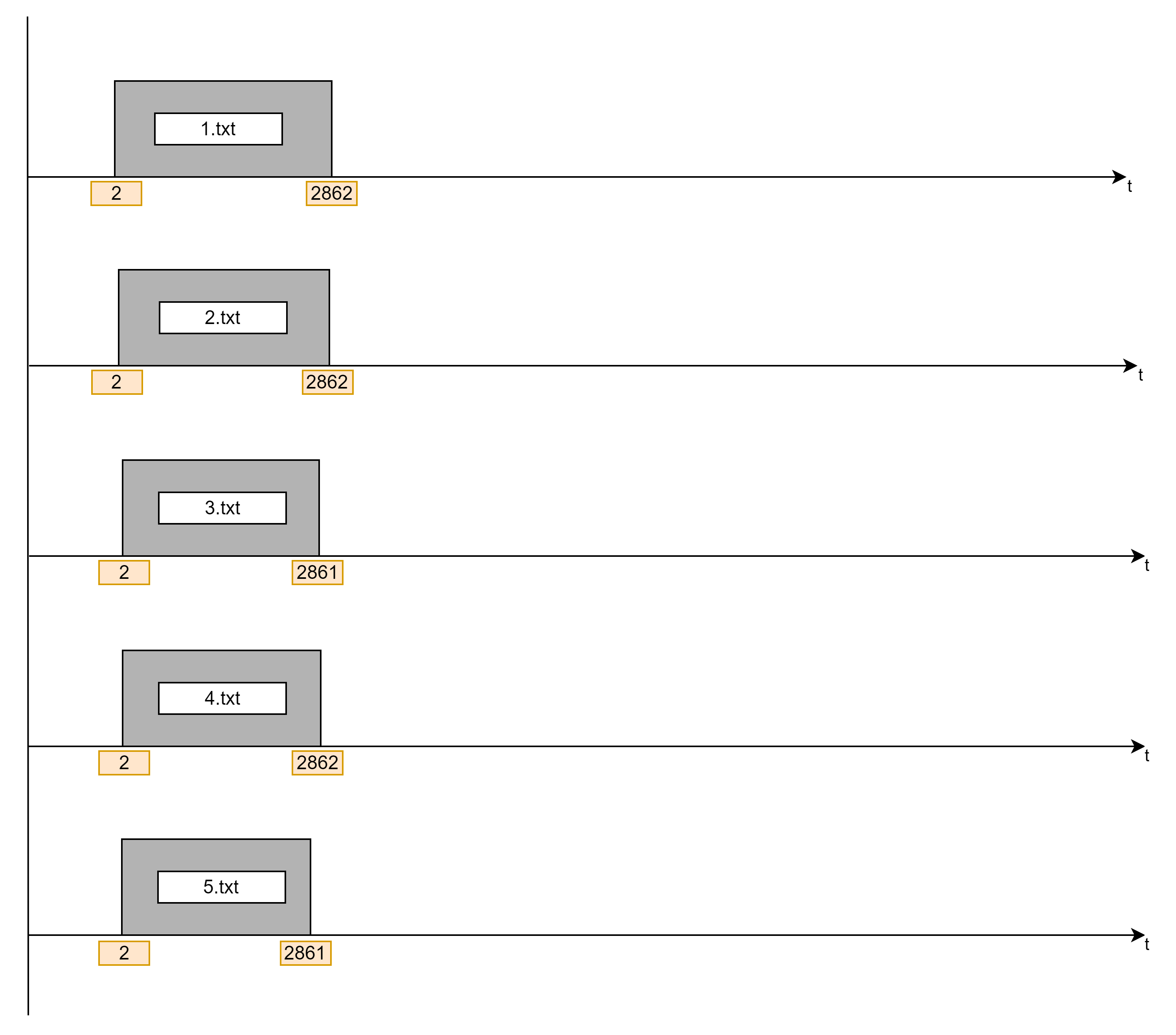


Рисунок 15 – Диаграмма трасс потоков программы BF с 5 текстовыми файлами

# ВЫВОДЫ ПО РЕЗУЛЬТАТАМ ПРОВЕДЕННГО ЭКСПЕРИМЕНТА

**Сравнение многопоточного режима (B) с однопоточным режимом (A) при размещении файлов на жестком диске (H)**:

В многопоточном режиме (B) время обработки заметно меньше, чем в однопоточном режиме (A). Например, при N=10 время в режиме A составляет 8814.8 мс, тогда как в режиме B – 986,6 мс. Многопоточность позволяет параллельно обрабатывать файлы, что снижает общее время за счет эффективного использования ресурсов процессора. Среднее время обработки в многопоточном режиме (B) уменьшилось примерно в 8.94 раза по сравнению с однопоточным режимом (A) на жестком диске.

**Сравнение среднего времени обработки в многопоточном режиме (B) по сравнению с режимом (A) при использовании съемного flash-накопителя (F)**:

В среднем многопоточный режим (B) показывает меньшие значения времени обработки по сравнению с однопоточным режимом (A). При N=10N=10N=10 время в режиме A - 11916.8 мс, в режиме B - 8136.6 мс. Многопоточность улучшает производительность даже при использовании медленных устройств ввода-вывода, таких как flash-накопители, за счет параллельной обработки.

**Объяснение с помощью диаграмм трасс потоков, каким образом многопоточность повлияла на общее время обработки файлов:**

Посмотрев на рисунки 13 и 14 соответственно, можно понять, что при использовании многопоточности, файлы будут обрабатывать параллельно, а не последовательно. Это занимает куда больше времени, так как программа может приступить к обработки следующего файла, не дожидаясь конца обработки предыдущего.

**Повышение приоритета потоков и его влияние на производительность обработки**:

Повышение приоритета потоков приводит к росту производительности обработки, так как потоки с более высоким приоритетом получают большее процессорное время, уменьшая общее время выполнения задачи. Однако в моем случае прирост производительности был незначителен, так как все файлы обрабатываются параллельно в каждом из дочерних потоков.

**Влияние количества ядер процессора на результаты**:

Количество ядер процессора положительно влияет на результаты многопоточной программы. Многопоточная программа работает быстрее однопоточной на многоядерных процессорах, но на одноядерном процессоре преимущество многопоточности уменьшается, так как все потоки будут конкурировать за одно ядро. Если отключить все ядра, кроме одного, многопоточная программа все равно будет иметь некоторые преимущества за счет распределения задач, но разница с однопоточной программой будет минимальна.

**Необходимость использования средств синхронизации потоков**:

В моем случае не потребовалось использование средств синхронизации потоков при работе с файлами, так как каждый поток обрабатывал отдельный файл, что исключало возможно наложения трасс потоков и искажений результатов.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Таненбаум Э., Бос Х. Современные операционные системы. - 4-е изд. - СПб.: Питер, 2015. - 1120 с.
2. Варфоломеев В.А. Организация многопоточных приложений в ОС Windows. Учебно-методическое пособие. — М.: МИИТ, 2024.— 24 с.
3. Соларес С. Windows System Programming. - 4-е изд. - Addison-Wesley, 2007. - 976 с.
4. Рихтер Д. Программирование под Windows. - М.: Диалектика, 2011. - 1360 с.
5. Гук М. Параллельное программирование для многопроцессорных систем. - М.: ДМК Пресс, 2014. - 400 с.
6. Microsoft. Multithreading and Concurrency: Windows Developer Documentation [Электронный ресурс] // Microsoft Docs. - 2023. URL: <https://docs.microsoft.com/en-us/windows/win32/procthread/multithreading-and-concurrency> (дата обращения: 12.06.2024).
7. Вейс М. Параллельное программирование с использованием C++. - М.: Вильямс, 2016. - 768 с.