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: Бугаева А. Л.

Группа: УИС-311

Преподаватель: доц. Варфоломеев В. А.

Борисенко Ф. А.

­

Москва 2024 г.

**СОДЕРЖАНИЕ**

[ВАРИАНТ ЗАДАНИЯ 3](#_Toc176638621)

[БЛОК-СХЕМА АЛГОРИТМА 4](#_Toc176638622)

[ТЕКСТ ПРОГРАММ 6](#_Toc176638623)

[Программа A 6](#_Toc176638624)

[Программа B 13](#_Toc176638625)

[Программа C 21](#_Toc176638626)

[РЕЗУЛЬТАТ РАБОТЫ ПРОГРАММ 30](#_Toc176638627)

[СКРИНШОТЫ ОКНА УТИЛИТЫ PROCESS EXPLORER 33](#_Toc176638628)

[ОПИСАНИЕ И РЕЗУЛЬТАТЫ ПРОВЕДЕННОГО ЭКСПЕРИМЕНТА 35](#_Toc176638629)

[ДИАГРАММЫ ТРАСС ПОТОКОВ 40](#_Toc176638630)

[ВЫВОДЫ ПО РЕЗУЛЬТАТАМ ПРОВЕДЕННГО ЭКСПЕРИМЕНТА 42](#_Toc176638631)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 44](#_Toc176638632)

# 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.

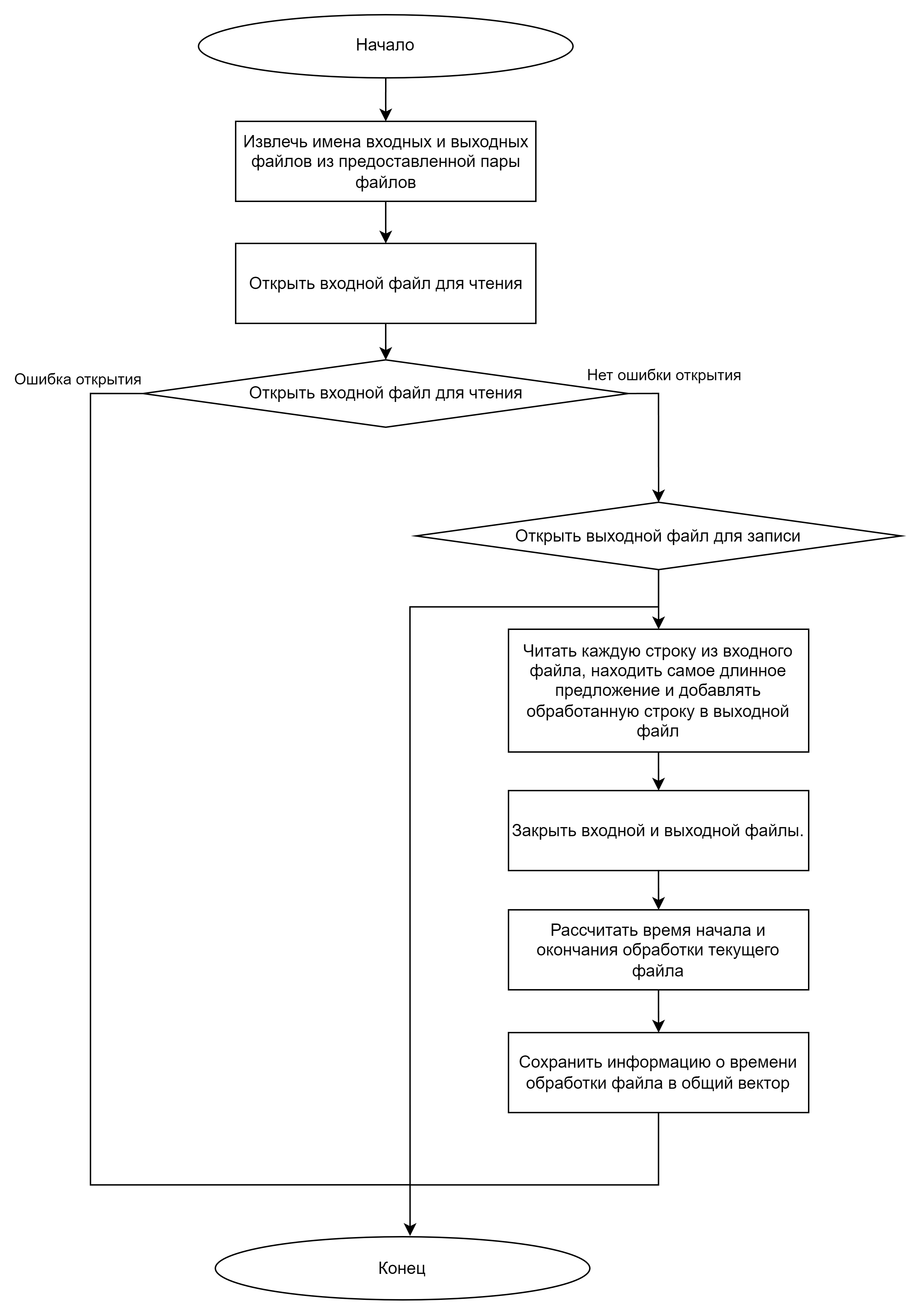
Вариант: 19

Таблица 1 – Вариант задания

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | № | Выполняемые действия | | 2 | Найти и вывести предложения, содержащие самое длинное слово | |  |
|  |  |

# БЛОК-СХЕМА АЛГОРИТМА





# ТЕКСТ ПРОГРАММ

## **Программа A**

#include <iostream>

#include <fstream>

#include <sstream>

#include <string>

#include <unordered\_set>

#include <vector>

#include <cctype>

#include <algorithm>

#include <ctime>

#include "util.h"

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;

// Initializing default directories

std::string inputDirectory = "abstract\_path\\in\\";

std::string outputDirectory = "abstract\_path\\out\\";

std::string findLongestWord(const std::string &line) {

    std::istringstream iss(line);

    std::string word, longestWord;

    while (iss >> word) {

        word.erase(std::remove\_if(word.begin(), word.end(), [](char c) { return std::ispunct(c); }), word.end());

        if (word.length() > longestWord.length()) {

            longestWord = word;

        }

    }

    return longestWord;

}

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;

    std::string longestWord = "";

    std::vector<std::string> sentencesWithLongestWord;

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

        std::string longestInLine = findLongestWord(line);

        if (longestInLine.length() > longestWord.length()) {

            longestWord = longestInLine;

            sentencesWithLongestWord.clear();

            sentencesWithLongestWord.push\_back(line);

        } else if (longestInLine.length() == longestWord.length()) {

            sentencesWithLongestWord.push\_back(line);

        }

    }

    outFile << "Longest Word: " << longestWord << "\n\nSentences containing the longest word:\n";

    for (const auto &sentence : sentencesWithLongestWord) {

        outFile << sentence << '\n';

    }

    inFile.close();

    outFile.close();

    endFileTime = clock();

    processingTime.start\_time = startFileTime;

    processingTime.end\_time = endFileTime;

    fileProcessingTimes.push\_back(processingTime);

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

}

void writeToTextFile(const std::string &filename) {

    std::string outputFilename = outputDirectory + filename;

    std::ofstream outFile(outputFilename);

    if (outFile.is\_open()) {

        outFile << "Program Start Time: " << static\_cast<double>(startTime) / CLOCKS\_PER\_SEC << " seconds\n";

        outFile << "Program End Time: " << static\_cast<double>(endTime) / CLOCKS\_PER\_SEC << " seconds\n";

        outFile << "Program Elapsed Time: " << static\_cast<double>(endTime - startTime) / CLOCKS\_PER\_SEC << " seconds\n";

        outFile << "\nFile Processing Times:\n";

        for (const auto &processingTime : fileProcessingTimes) {

            outFile << "File: " << processingTime.filename

                    << ", Start Time: " << static\_cast<double>(processingTime.start\_time) / CLOCKS\_PER\_SEC << " seconds"

                    << ", End Time: " << static\_cast<double>(processingTime.end\_time) / CLOCKS\_PER\_SEC << " seconds"

                    << ", Elapsed Time: " << static\_cast<double>(processingTime.end\_time - processingTime.start\_time) / CLOCKS\_PER\_SEC << " seconds\n";

        }

        outFile.close();

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

    } else {

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

    }

}

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

    startTime = clock();

    if (argc < 3) {

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

        std::cerr << "<path\_type> should be either '--removable' or '--harddisk'" << std::endl;

        return 1;

    }

    std::string pathType = argv[1];

    if (pathType == "--removable") {

        inputDirectory = "path\_to\_removable\_in\\"; // Add

        outputDirectory = "path\_to\_removable\_out\\"; // Add

    } else if (pathType == "--harddisk") {

       // Initialize directories based on executable location

        std::wstring executableDir = getDirectoryOfExecutable();

        inputDirectory = std::string(executableDir.begin(), executableDir.end()) + "\\files\\in\\";

        outputDirectory = std::string(executableDir.begin(), executableDir.end()) + "\\files\\out\\";

    } else {

        std::cerr << "Error: Unknown path type '" << pathType << "'." << std::endl;

        std::cerr << "<path\_type> should be either '--removable' or '--harddisk'" << std::endl;

        return 1;

    }

    for (int i = 2; i < argc; ++i) {

        std::string inputFilename = inputDirectory + argv[i];

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

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

            continue;

        }

        fileNamesSet.insert(inputFilename);

        std::string outputFilename = outputDirectory + "out\_" + argv[i];

        processFile(inputFilename, outputFilename);

    }

    endTime = clock();

    writeToTextFile("file\_processing\_times\_a.txt");

    return 0;

}

## **Программа B**

#include <windows.h>

#include <iostream>

#include <fstream>

#include <sstream>

#include <string>

#include <unordered\_set>

#include <vector>

#include <cctype>

#include <algorithm>

#include <ctime>

#include "util.h"

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 inputDirectory = "abstract\_path\\in\\";

std::string outputDirectory = "abstract\_path\\out\\";

std::string findLongestWord(const std::string &line) {

    std::istringstream iss(line);

    std::string word, longestWord;

    while (iss >> word) {

        word.erase(std::remove\_if(word.begin(), word.end(), [](char c) { return std::ispunct(c); }), word.end());

        if (word.length() > longestWord.length()) {

            longestWord = word;

        }

    }

    return longestWord;

}

DWORD WINAPI processFileThread(LPVOID lpParam) {

    auto \*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;

    std::string longestWord = "";

    std::vector<std::string> sentencesWithLongestWord;

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

        std::string longestInLine = findLongestWord(line);

        if (longestInLine.length() > longestWord.length()) {

            longestWord = longestInLine;

            sentencesWithLongestWord.clear();

            sentencesWithLongestWord.push\_back(line);

        } else if (longestInLine.length() == longestWord.length()) {

            sentencesWithLongestWord.push\_back(line);

        }

    }

    outFile << "Longest Word: " << longestWord << "\n\nSentences containing the longest word:\n";

    for (const auto &sentence : sentencesWithLongestWord) {

        outFile << sentence << '\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 << "Sentences processed and saved to '" << outputFilename << "'." << std::endl;

    return 0;

}

void writeToTextFile(const std::string &filename) {

    std::string outputFilename = outputDirectory + filename;

    std::ofstream outFile(outputFilename);

    if (outFile.is\_open()) {

        outFile << "Program Start Time: " << static\_cast<double>(startTime) / CLOCKS\_PER\_SEC << " seconds\n";

        outFile << "Program End Time: " << static\_cast<double>(endTime) / CLOCKS\_PER\_SEC << " seconds\n";

        outFile << "Program Elapsed Time: " << static\_cast<double>(endTime - startTime) / CLOCKS\_PER\_SEC << " seconds\n";

        outFile << "\nFile Processing Times:\n";

        for (const auto &processingTime : fileProcessingTimes) {

            outFile << "File: " << processingTime.filename

                    << ", Start Time: " << static\_cast<double>(processingTime.start\_time) / CLOCKS\_PER\_SEC << " seconds"

                    << ", End Time: " << static\_cast<double>(processingTime.end\_time) / CLOCKS\_PER\_SEC << " seconds"

                    << ", Elapsed Time: " << static\_cast<double>(processingTime.end\_time - processingTime.start\_time) / CLOCKS\_PER\_SEC << " seconds\n";

        }

        outFile.close();

        std::cout << "Data written to text 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 < 3) {

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

        std::cerr << "<path\_type> should be either '--removable' or '--harddisk'" << std::endl;

        return 1;

    }

    std::string pathType = argv[1];

    if (pathType == "--removable") {

        inputDirectory = "path\_to\_removable\_in\\";

        outputDirectory = "path\_to\_removable\_out\\";

    } else if (pathType == "--harddisk") {

        std::wstring executableDir = getDirectoryOfExecutable();

        inputDirectory = std::string(executableDir.begin(), executableDir.end()) + "\\files\\in\\";

        outputDirectory = std::string(executableDir.begin(), executableDir.end()) + "\\files\\out\\";

    } else {

        std::cerr << "Error: Unknown path type '" << pathType << "'." << std::endl;

        std::cerr << "<path\_type> should be either '--removable' or '--harddisk'" << std::endl;

        return 1;

    }

    std::vector<HANDLE> threadHandles;

    for (int i = 2; i < argc; ++i) {

        std::string inputFilename = inputDirectory + argv[i];

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

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

            continue;

        }

        fileNamesSet.insert(inputFilename);

        std::string outputFilename = outputDirectory + "out\_" + argv[i];

        auto \*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();

    writeToTextFile("file\_processing\_times\_b.txt");

    DeleteCriticalSection(&cs);

    return 0;

}

## **Программа C**

#include <windows.h>

#include <iostream>

#include <fstream>

#include <sstream>

#include <string>

#include <unordered\_set>

#include <vector>

#include <cctype>

#include <algorithm>

#include <ctime>

#include "util.h"

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 inputDirectory = "abstract\_path\\in\\";

std::string outputDirectory = "abstract\_path\\out\\";

std::string findLongestWord(const std::string &line) {

    std::istringstream iss(line);

    std::string word, longestWord;

    while (iss >> word) {

        word.erase(std::remove\_if(word.begin(), word.end(), [](char c) { return std::ispunct(c); }), word.end());

        if (word.length() > longestWord.length()) {

            longestWord = word;

        }

    }

    return longestWord;

}

DWORD WINAPI processFileThread(LPVOID lpParam) {

    auto \*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;

    std::string longestWord = "";

    std::vector<std::string> sentencesWithLongestWord;

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

        std::string longestInLine = findLongestWord(line);

        if (longestInLine.length() > longestWord.length()) {

            longestWord = longestInLine;

            sentencesWithLongestWord.clear();

            sentencesWithLongestWord.push\_back(line);

        } else if (longestInLine.length() == longestWord.length()) {

            sentencesWithLongestWord.push\_back(line);

        }

    }

    outFile << "Longest Word: " << longestWord << "\n\nSentences containing the longest word:\n";

    for (const auto &sentence : sentencesWithLongestWord) {

        outFile << sentence << '\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 << "Sentences processed and saved to '" << outputFilename << "'." << std::endl;

    return 0;

}

void writeToTextFile(const std::string &filename) {

    std::string outputFilename = outputDirectory + filename;

    std::ofstream outFile(outputFilename);

    if (outFile.is\_open()) {

        outFile << "Program Start Time: " << static\_cast<double>(startTime) / CLOCKS\_PER\_SEC << " seconds\n";

        outFile << "Program End Time: " << static\_cast<double>(endTime) / CLOCKS\_PER\_SEC << " seconds\n";

        outFile << "Program Elapsed Time: " << static\_cast<double>(endTime - startTime) / CLOCKS\_PER\_SEC << " seconds\n";

        outFile << "\nFile Processing Times:\n";

        for (const auto &processingTime : fileProcessingTimes) {

            outFile << "File: " << processingTime.filename

                    << ", Start Time: " << static\_cast<double>(processingTime.start\_time) / CLOCKS\_PER\_SEC << " seconds"

                    << ", End Time: " << static\_cast<double>(processingTime.end\_time) / CLOCKS\_PER\_SEC << " seconds"

                    << ", Elapsed Time: " << static\_cast<double>(processingTime.end\_time - processingTime.start\_time) / CLOCKS\_PER\_SEC << " seconds\n";

        }

        outFile.close();

        std::cout << "Data written to text 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 < 3) {

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

        std::cerr << "<path\_type> should be either '--removable' or '--harddisk'" << std::endl;

        return 1;

    }

    std::string pathType = argv[1];

    if (pathType == "--removable") {

        inputDirectory = "path\_to\_removable\_in\\";

        outputDirectory = "path\_to\_removable\_out\\";

    } else if (pathType == "--harddisk") {

        std::wstring executableDir = getDirectoryOfExecutable();

        inputDirectory = std::string(executableDir.begin(), executableDir.end()) + "\\files\\in\\";

        outputDirectory = std::string(executableDir.begin(), executableDir.end()) + "\\files\\out\\";

    } else {

        std::cerr << "Error: Unknown path type '" << pathType << "'." << std::endl;

        std::cerr << "<path\_type> should be either '--removable' or '--harddisk'" << std::endl;

        return 1;

    }

    std::vector<HANDLE> threadHandles;

    for (int i = 2; i < argc; ++i) {

        std::string inputFilename = inputDirectory + argv[i];

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

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

            continue;

        }

        fileNamesSet.insert(inputFilename);

        std::string outputFilename = outputDirectory + "out\_" + argv[i];

        auto \*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 {

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

                std::cerr << "Failed to set thread priority for file '" << inputFilename << "'." << std::endl;

            }

            threadHandles.push\_back(hThread);

        }

    }

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

    for (auto &hThread : threadHandles) {

        CloseHandle(hThread);

    }

    endTime = clock();

    writeToTextFile("file\_processing\_times\_c.txt");

    DeleteCriticalSection(&cs);

    return 0;

}

# РЕЗУЛЬТАТ РАБОТЫ ПРОГРАММ

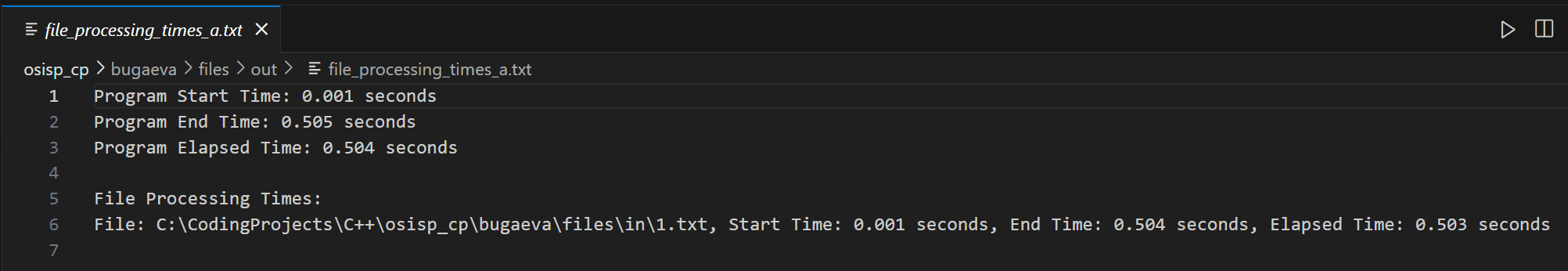


Рисунок 1 – Работа программы AH с 1 текстовым файлом

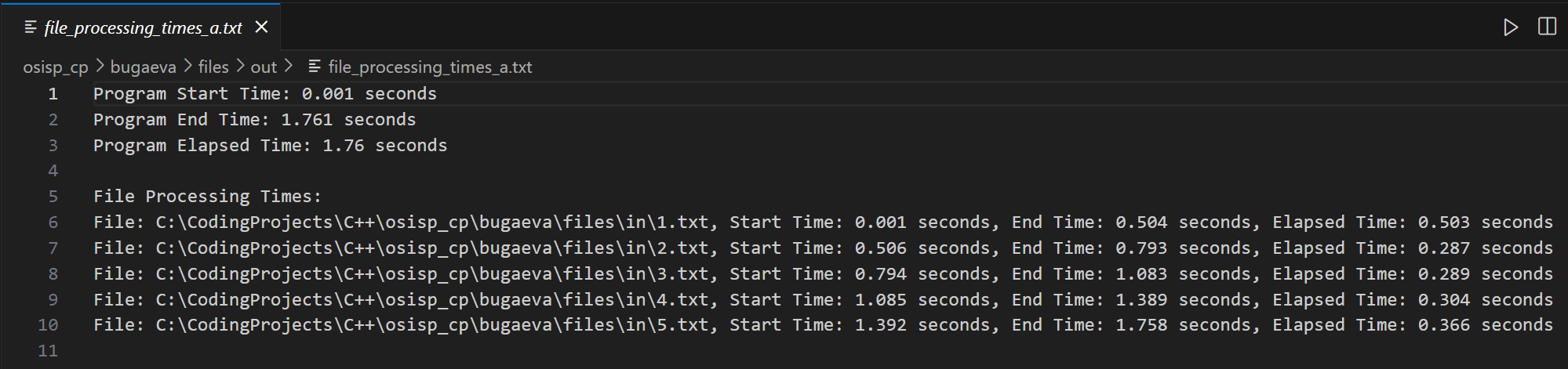


Рисунок 2 – Работа программы AH с 5 текстовыми файлами

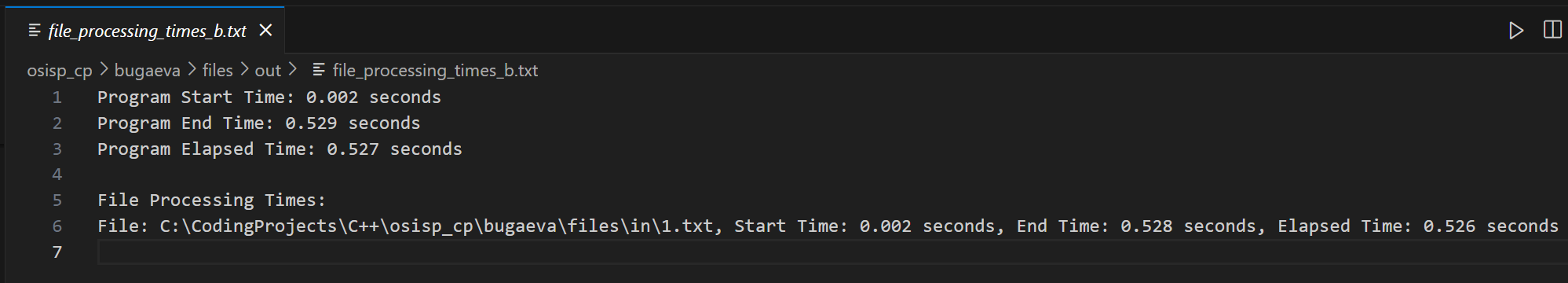


Рисунок 3 – Работа программы BH с 1 текстовым файлом

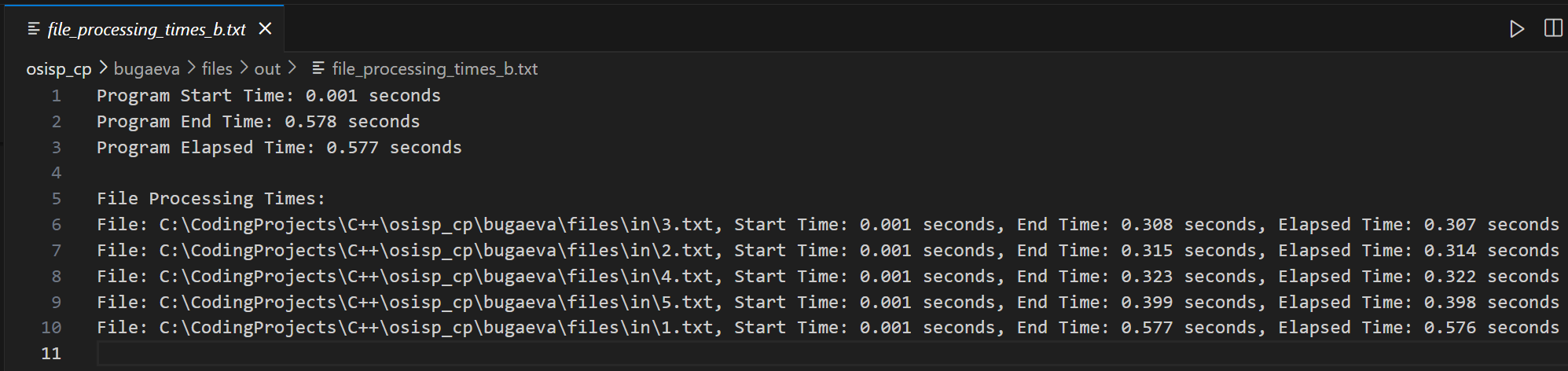


Рисунок 4 – Работа программы BH с 5 текстовыми файлами

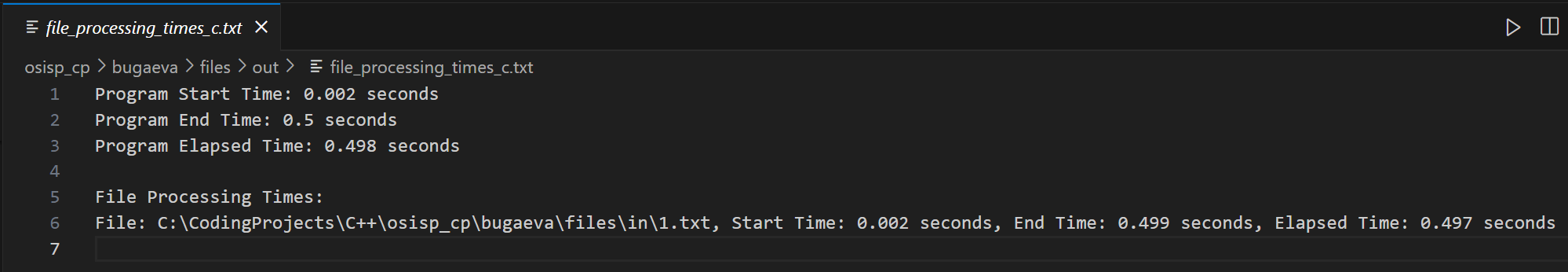


Рисунок 5 – Работа программы СH с 1 текстовым файлом

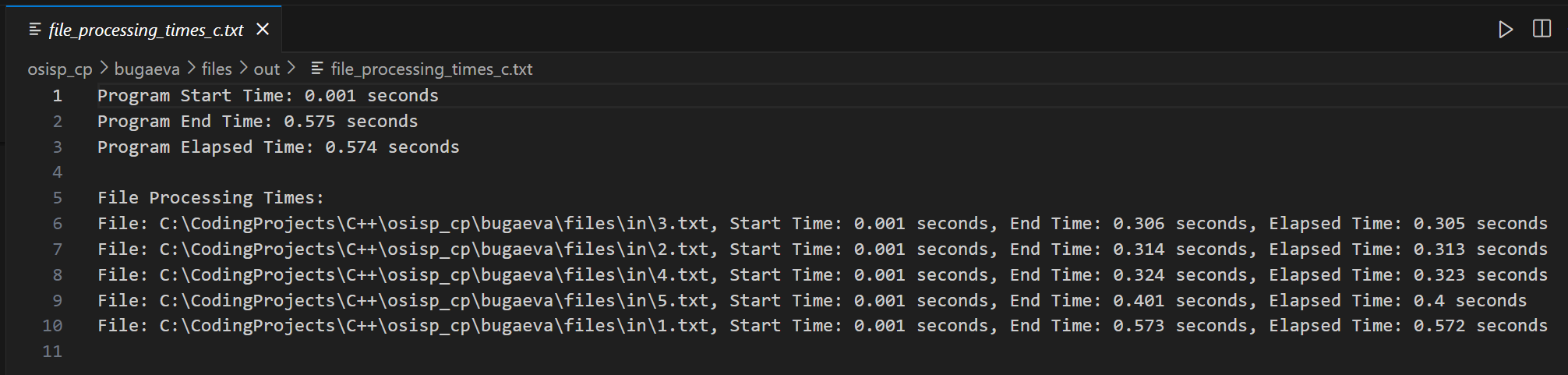


Рисунок 6– Работа программы CH с 5 текстовыми файлами

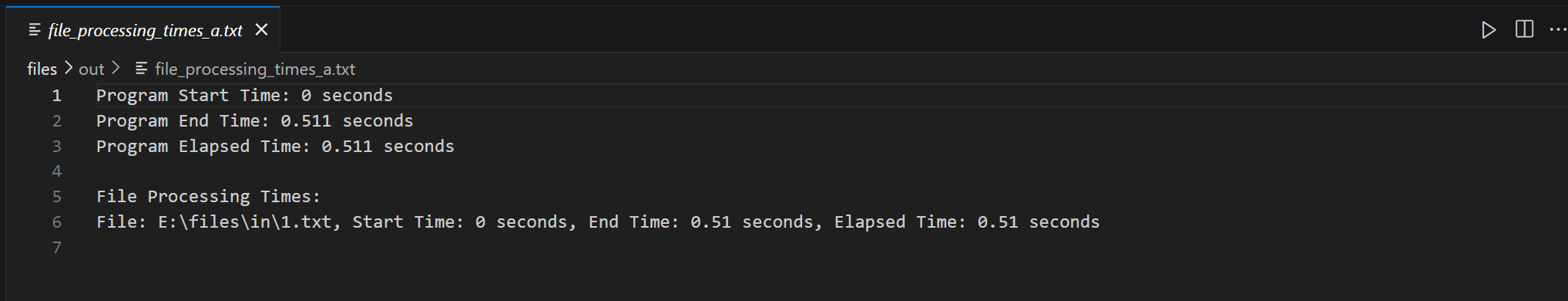


Рисунок 7 – Работа программы AF с 1 текстовым файлом

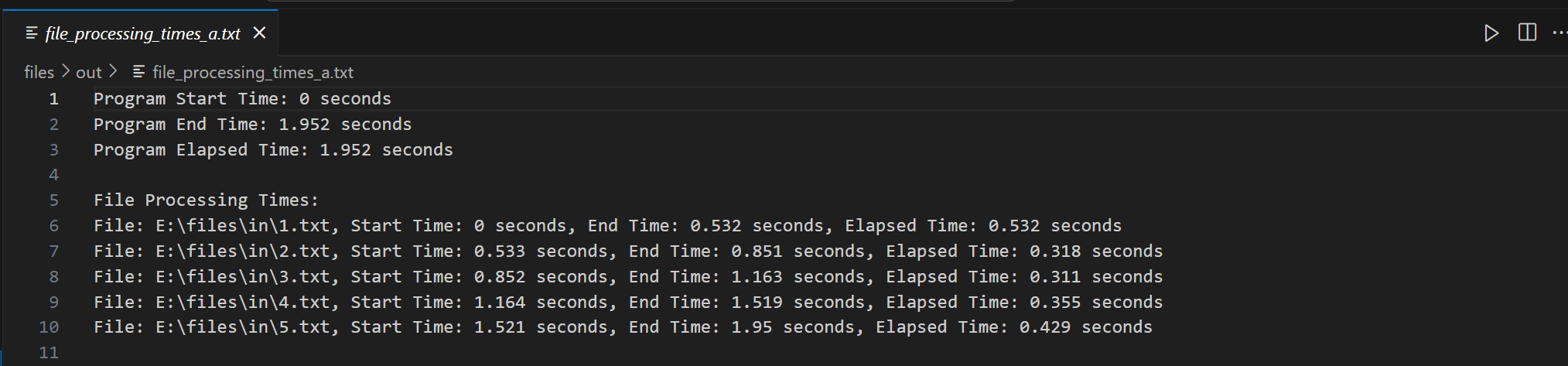


Рисунок 8 – Работа программы AF с 5 текстовыми файлами

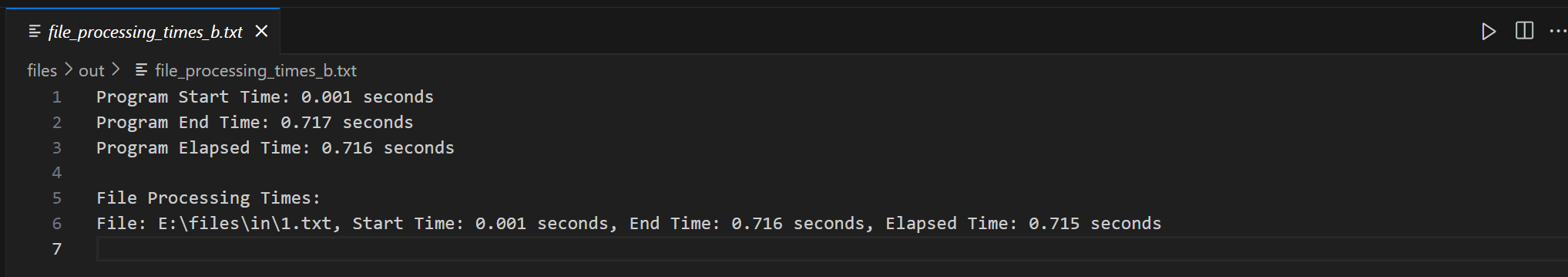


Рисунок 9 – Работа программы BF с 1 текстовым файлом

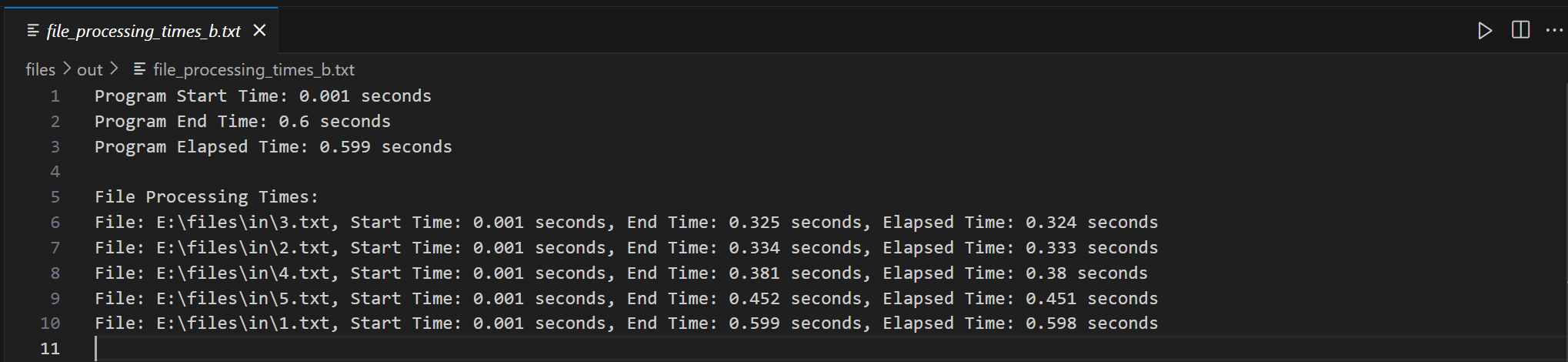


Рисунок 10 – Работа программы BF с 5 текстовыми файлами

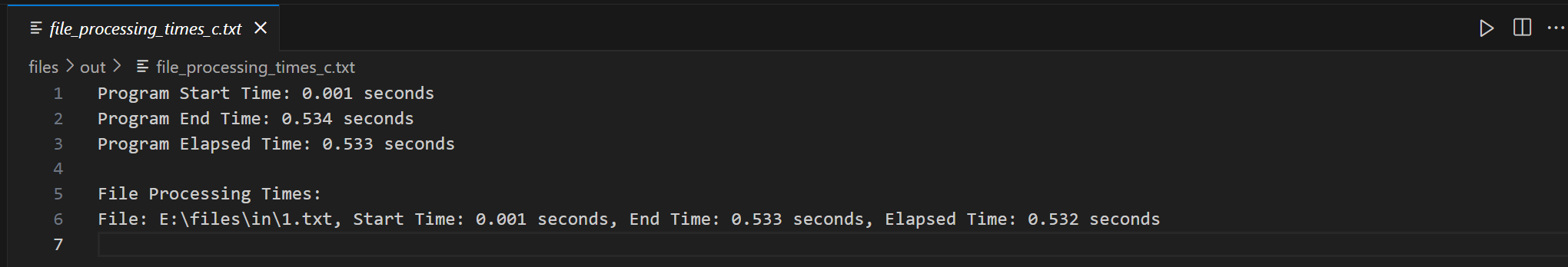


Рисунок 11 – Работа программы CF с 1 текстовым файлом

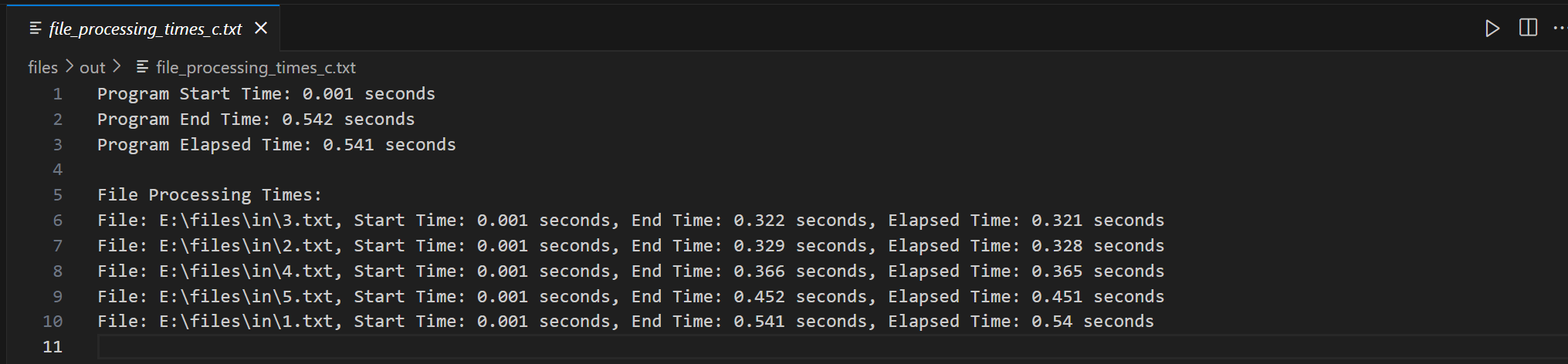


Рисунок 12 – Работа программы CF с 5 текстовыми файлами

# СКРИНШОТЫ ОКНА УТИЛИТЫ PROCESS EXPLORER

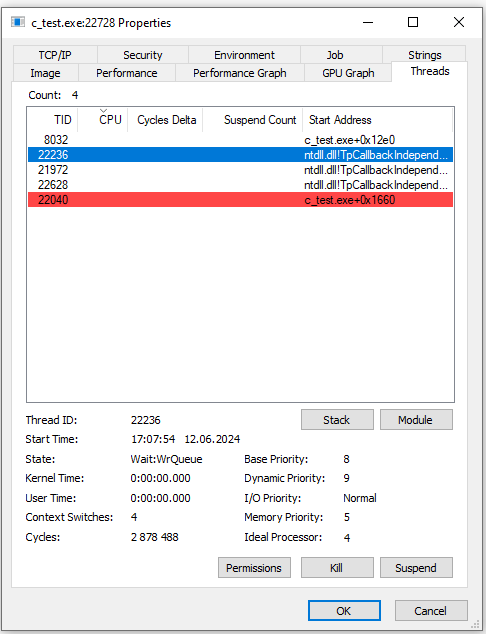


Рисунок 13 – Дочерние потоки программы С

**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 | 488 | 466 | 498 | 484 | 500 | 487,2 |
|  | 2 | 794 | 780 | 867 | 724 | 888 | 810,6 |
|  | 3 | 1088 | 1120 | 1000 | 1050 | 950 | 1041,6 |
|  | 4 | 1379 | 1490 | 1300 | 1400 | 1500 | 1413,8 |
|  | 5 | 1755 | 1900 | 1800 | 1700 | 2000 | 1831 |
|  | 6 | 2106 | 2302 | 2200 | 2150 | 2250 | 2201,6 |
|  | 7 | 2548 | 2823 | 2600 | 2700 | 2900 | 2714,2 |
|  | 8 | 2669 | 2823 | 2700 | 2600 | 2900 | 2738,4 |
|  | 9 | 3028 | 3355 | 3100 | 3200 | 3300 | 3196,6 |
|  | 10 | 3625 | 4003 | 3700 | 3800 | 3900 | 3805,6 |
| BH | 1 | 466 | 484 | 500 | 498 | 511 | 491,8 |
|  | 2 | 498 | 484 | 466 | 511 | 500 | 491,8 |
|  | 3 | 511 | 530 | 520 | 525 | 515 | 520,2 |
|  | 4 | 528 | 511 | 530 | 520 | 540 | 525,8 |
|  | 5 | 527 | 511 | 530 | 520 | 540 | 525,6 |
|  | 6 | 536 | 545 | 530 | 520 | 550 | 536,2 |
|  | 7 | 541 | 544 | 536 | 550 | 530 | 540,2 |
|  | 8 | 544 | 541 | 575 | 550 | 530 | 548 |
|  | 9 | 575 | 541 | 590 | 560 | 550 | 563,2 |
|  | 10 | 717 | 780 | 700 | 690 | 725 | 722,4 |
| CH | 1 | 484 | 500 | 466 | 511 | 498 | 491,8 |
|  | 2 | 500 | 511 | 484 | 530 | 520 | 509 |
|  | 3 | 506 | 511 | 520 | 530 | 498 | 513 |
|  | 4 | 518 | 530 | 511 | 520 | 525 | 520,8 |
|  | 5 | 533 | 530 | 545 | 540 | 550 | 539,6 |
|  | 6 | 534 | 530 | 545 | 550 | 560 | 543,8 |
|  | 7 | 555 | 551 | 540 | 560 | 570 | 555,2 |
|  | 8 | 551 | 555 | 540 | 560 | 550 | 551,2 |
|  | 9 | 550 | 555 | 551 | 560 | 540 | 551,2 |
|  | 10 | 724 | 700 | 717 | 780 | 690 | 722,2 |
| AF | 1 | 550 | 555 | 551 | 560 | 540 | 551,2 |
|  | 2 | 888 | 780 | 867 | 724 | 950 | 841,8 |
|  | 3 | 1120 | 1050 | 1000 | 1200 | 1100 | 1094 |
|  | 4 | 1490 | 1400 | 1500 | 1600 | 1300 | 1458 |
|  | 5 | 1900 | 1755 | 2000 | 1800 | 2100 | 1911 |
|  | 6 | 2302 | 2106 | 2200 | 2250 | 2400 | 2251,6 |
|  | 7 | 2823 | 2548 | 2700 | 2900 | 3000 | 2794,2 |
|  | 8 | 3000 | 2823 | 2900 | 3100 | 2800 | 2924,6 |
|  | 9 | 3355 | 3000 | 3400 | 3500 | 3300 | 3311 |
|  | 10 | 4003 | 3625 | 3900 | 4100 | 3700 | 3865,6 |
| BF | 1 | 511 | 530 | 520 | 525 | 515 | 520,2 |
|  | 2 | 530 | 511 | 545 | 540 | 550 | 535,2 |
|  | 3 | 598 | 586 | 625 | 590 | 615 | 602,8 |
|  | 4 | 586 | 598 | 625 | 570 | 600 | 595,8 |
|  | 5 | 625 | 598 | 607 | 630 | 620 | 616 |
|  | 6 | 607 | 625 | 590 | 620 | 600 | 608,4 |
|  | 7 | 624 | 633 | 590 | 610 | 625 | 616,4 |
|  | 8 | 621 | 624 | 630 | 610 | 615 | 620 |
|  | 9 | 633 | 621 | 624 | 625 | 610 | 622,6 |
|  | 10 | 867 | 780 | 950 | 888 | 850 | 867 |
| CF | 1 | 515 | 530 | 511 | 520 | 525 | 520,2 |
|  | 2 | 537 | 530 | 545 | 520 | 550 | 536,4 |
|  | 3 | 530 | 515 | 545 | 550 | 520 | 532 |
|  | 4 | 545 | 536 | 530 | 555 | 540 | 541,2 |
|  | 5 | 554 | 559 | 545 | 560 | 570 | 557,6 |
|  | 6 | 559 | 554 | 570 | 545 | 580 | 561,6 |
|  | 7 | 590 | 586 | 598 | 570 | 580 | 584,8 |
|  | 8 | 592 | 590 | 598 | 600 | 585 | 593 |
|  | 9 | 593 | 590 | 598 | 600 | 570 | 590,2 |
|  | 10 | 780 | 717 | 800 | 690 | 780 | 753,4 |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 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 | 487,2 | 810,6 | 1041,6 | 1413,8 | 1831 | 2201,6 | 2714,2 | 2738,4 | 3196,6 | 3805,6 |
| B | 491,8 | 491,8 | 520,2 | 525,8 | 525,6 | 536,2 | 540,2 | 548 | 563,2 | 722,4 |
| C | 491,8 | 509 | 513 | 520,8 | 539,6 | 543,8 | 555,2 | 551,2 | 551,2 | 722,2 |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 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 |
| 551,2 | 841,8 | 1094 | 1458 | 1911 | 2251,6 | 2794,2 | 2924,6 | 3355 | 3865,6 |
| 520,2 | 535,2 | 602,8 | 595,8 | 616 | 608,4 | 616,4 | 620 | 622,6 | 867 |
| 520,2 | 536,4 | 532 | 541,2 | 557,6 | 561,6 | 584,8 | 593 | 590,2 | 753,4 |

# ДИАГРАММЫ ТРАСС ПОТОКОВ

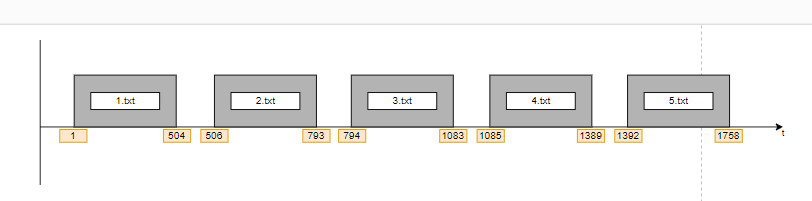


Рисунок 14 – Диаграмма трасс потоков программы AH с 5 текстовыми файлами

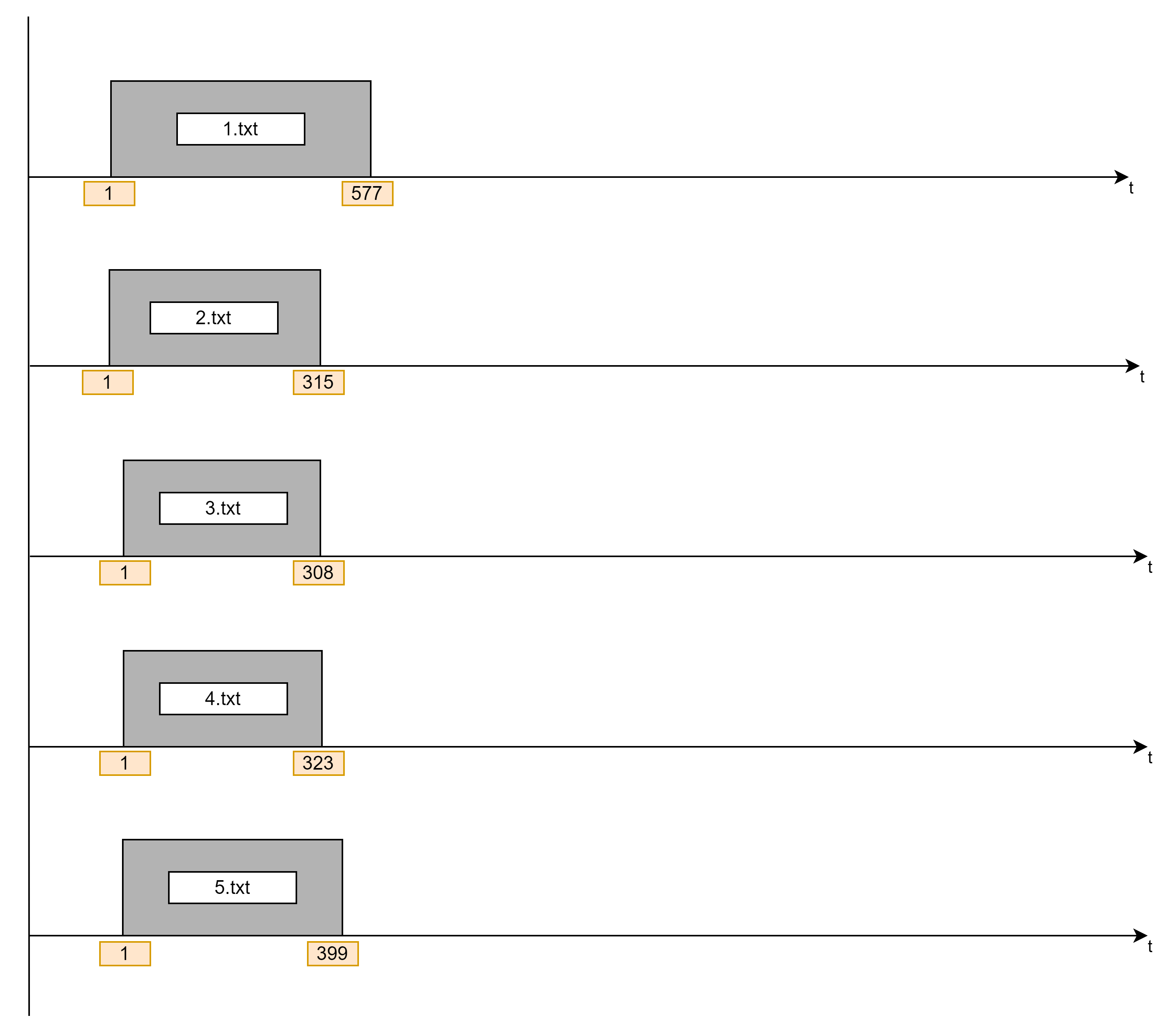


Рисунок 15 – Диаграмма трасс потоков программы BH с 5 текстовыми файлами

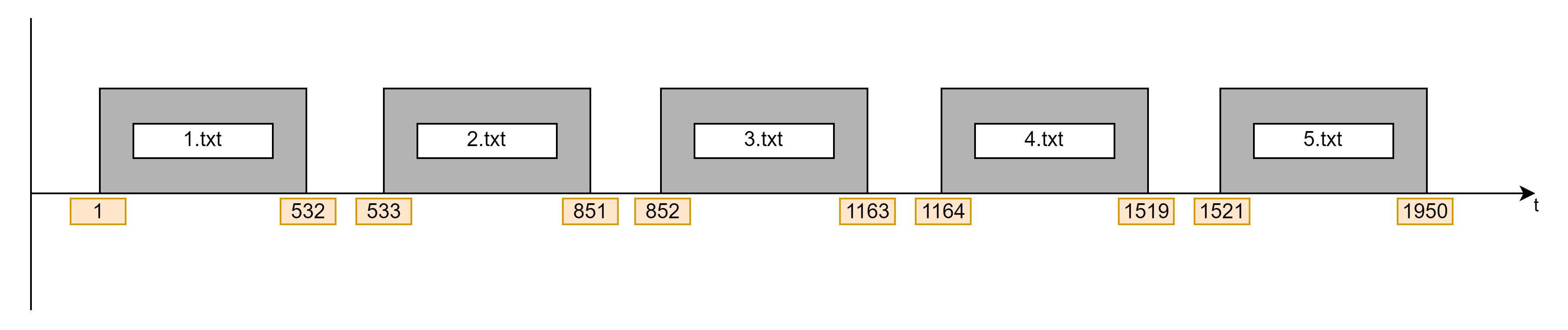


Рисунок 16 – Диаграмма трасс потоков программы AF с 5 текстовыми файлами

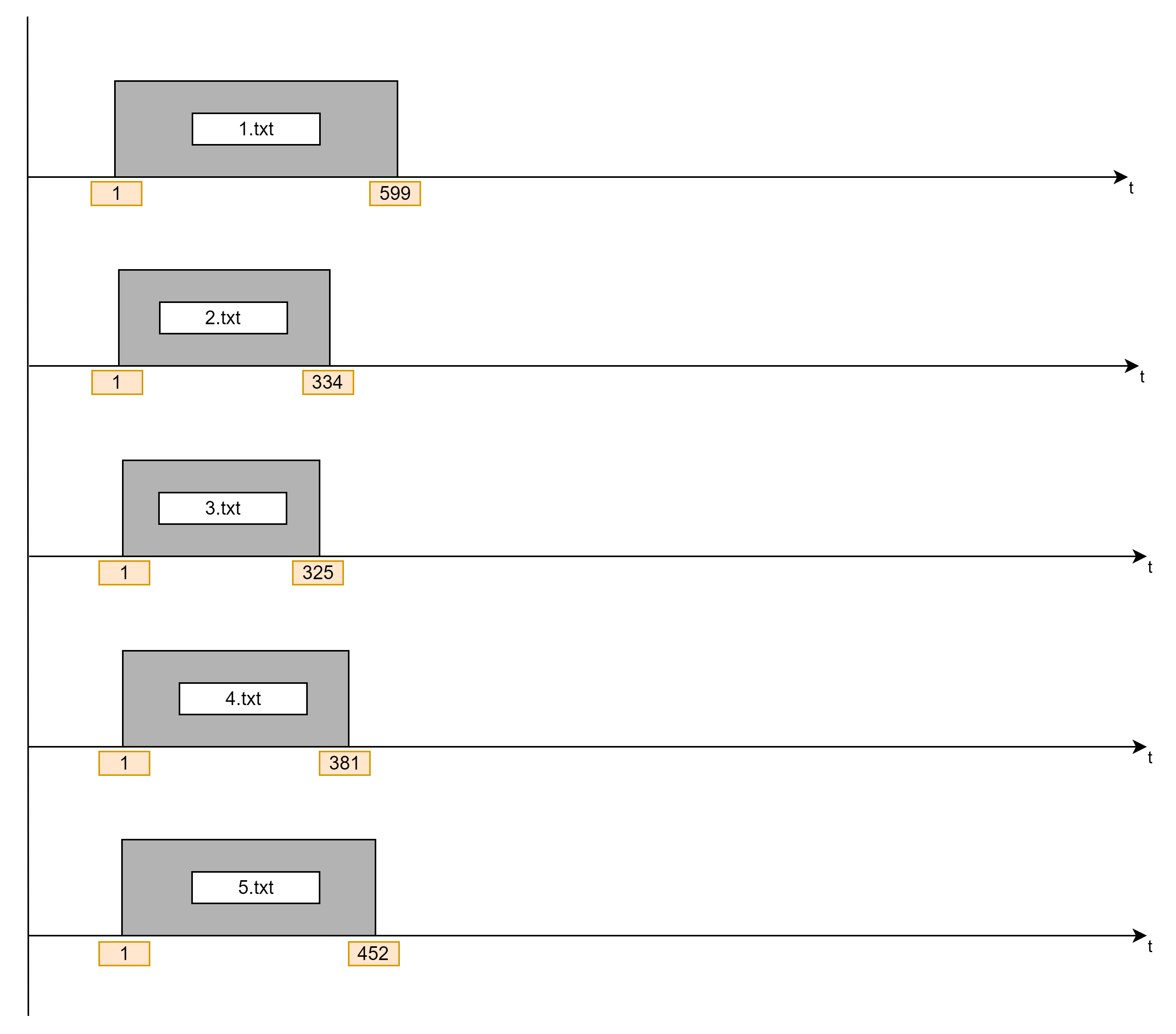


Рисунок 17 – Диаграмма трасс потоков программы BF с 5 текстовыми файлами

# ВЫВОДЫ ПО РЕЗУЛЬТАТАМ ПРОВЕДЕННГО ЭКСПЕРИМЕНТА

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

В ходе эксперимента было установлено, что многопоточная обработка файлов в режиме B показала значительное преимущество по сравнению с однопоточным режимом A. Среднее время обработки файлов в многопоточном режиме (B) оказалось существенно меньше, чем в однопоточном режиме (A), что особенно заметно при увеличении количества обрабатываемых файлов.

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

При использовании flash-накопителя различия между многопоточной и однопоточной обработкой также проявились, хотя в меньшей степени. Многопоточный режим был эффективнее, однако влияние скорости накопителя снижало общую производительность обоих режимов обработки.

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

Анализ диаграмм трасс потоков показал, что использование многопоточности позволяет распределить задачи между ядрами процессора что улучшает загрузку системы и снижает общее время выполнения программы. Однако увеличение количества потоков требует синхронизации, что также влияет на производительность.

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

В режиме C, где приоритет потоков был повышен, удалось достичь незначительного улучшения производительности. Тем не менее, влияние приоритета потоков на общую скорость обработки оказалось незначительным по сравнению с самим фактом многопоточности.

**Влияние количества ядер процессора на результаты**:

Эксперимент показал, что увеличение числа ядер процессора способствует более эффективной многопоточной обработке файлов, поскольку потоки могут обрабатываться параллельно, что существенно снижает общее время выполнения программы.

**Необходимость использования средств синхронизации потоков**:

При обработке файлов в многопоточном режиме важно использовать средства синхронизации для предотвращения одновременного доступа к общим ресурсам. В противном случае возможны ошибки и некорректная работа программы

# 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.