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: Дронов А. А.

Группа: УИС-311

Преподаватель: доц. Варфоломеев В. А.

­

Москва 2024 г.

**СОДЕРЖАНИЕ**

[ВАРИАНТ ЗАДАНИЯ 3](#_Toc176644761)

[БЛОК-СХЕМА АЛГОРИТМА 4](#_Toc176644762)

[ТЕКСТ ПРОГРАММ 6](#_Toc176644763)

[Программа A 6](#_Toc176644764)

[Программа B 12](#_Toc176644765)

[Программа C 20](#_Toc176644766)

[РЕЗУЛЬТАТ РАБОТЫ ПРОГРАММ 28](#_Toc176644767)

[СКРИНШОТЫ ОКНА УТИЛИТЫ PROCESS EXPLORER 31](#_Toc176644768)

[ОПИСАНИЕ И РЕЗУЛЬТАТЫ ПРОВЕДЕННОГО ЭКСПЕРИМЕНТА 33](#_Toc176644769)

[ДИАГРАММЫ ТРАСС ПОТОКОВ 38](#_Toc176644770)

[ВЫВОДЫ ПО РЕЗУЛЬТАТАМ ПРОВЕДЕННГО ЭКСПЕРИМЕНТА 40](#_Toc176644771)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 42](#_Toc176644772)

# 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.

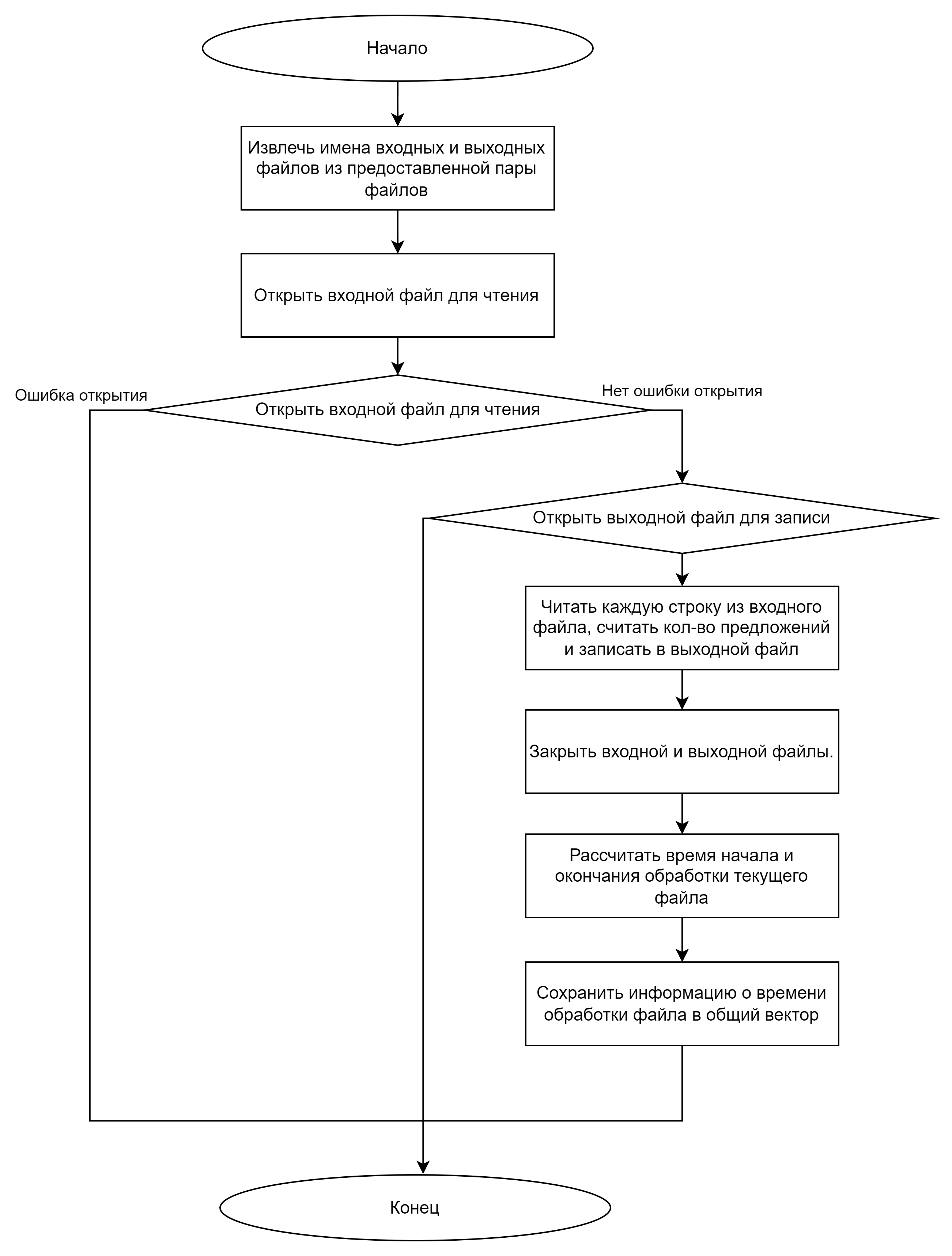
Вариант: 5

Таблица 1 – Вариант задания

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | № | Выполняемые действия | | 5 | Подсчитать общее количество предложений | |  |
|  |  |

# БЛОК-СХЕМА АЛГОРИТМА





# ТЕКСТ ПРОГРАММ

## **Программа A**

#include <iostream>

#include <fstream>

#include <sstream>

#include <string>

#include <unordered\_set>

#include <vector>

#include <algorithm>

#include <ctime>

#include "util.h"

struct FileProcessingTime {

    std::string filename;

    clock\_t start\_time;

    clock\_t end\_time;

    int sentence\_count;  //  Sentences counter

};

std::vector<FileProcessingTime> fileProcessingTimes;

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

clock\_t startTime;

clock\_t endTime;

// Directories initializing

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

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

// Function for counting sentences from file

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;

    int sentenceCount = 0;

    // Reading file content

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

        for (char ch : line) {

            if (ch == '.' || ch == '?' || ch == '!') {

                sentenceCount++;

            }

        }

    }

    processingTime.sentence\_count = sentenceCount;

    outFile << "Number of sentences: " << sentenceCount << '\n';

    inFile.close();

    outFile.close();

    endFileTime = clock();

    processingTime.start\_time = startFileTime;

    processingTime.end\_time = endFileTime;

    fileProcessingTimes.push\_back(processingTime);

    std::cout << "File '" << inputFilename << "' processed with " << sentenceCount << " sentences. Result 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"

                    << ", Number of Sentences: " << processingTime.sentence\_count << '\n';  // Добавляем количество предложений

        }

        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 = "E:\\files\\in\\";

        outputDirectory = "E:\\files\\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;

    }

    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 <algorithm>

#include <ctime>

#include "util.h"

// Struct to store file processing time details

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;

// Initializing default directories

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

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

// Function to trim leading and trailing whitespace

std::string trimWhitespace(const std::string &str) {

    auto start = str.find\_first\_not\_of(" \t");

    auto end = str.find\_last\_not\_of(" \t");

    return (start == std::string::npos || end == std::string::npos) ? "" : str.substr(start, end - start + 1);

}

// Thread function to process a file and sort lines by the first character

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::vector<std::string> lines;

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

        line = trimWhitespace(line);

        lines.push\_back(line);

    }

    // Sort lines by the first character, considering that some lines may be empty

    std::sort(lines.begin(), lines.end(), [](const std::string &a, const std::string &b) {

        return a.empty() || b.empty() ? a < b : a[0] < b[0];

    });

    for (const auto &sortedLine : lines) {

        outFile << sortedLine << '\n';

    }

    inFile.close();

    outFile.close();

    endFileTime = clock();

    processingTime.start\_time = startFileTime;

    processingTime.end\_time = endFileTime;

    // Use critical section to safely add data to shared vector

    EnterCriticalSection(&cs);

    fileProcessingTimes.push\_back(processingTime);

    LeaveCriticalSection(&cs);

    std::cout << "Lines sorted and saved to '" << outputFilename << "'." << std::endl;

    return 0;

}

// Function to write processing times to a text file

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 = "E:\\files\\in\\";

        outputDirectory = "E:\\files\\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];

        // Create a pair of input and output filenames and pass to the thread

        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);

        }

    }

    // Wait for all threads to finish

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

    // Close all thread handles

    for (auto &hThread : threadHandles) {

        CloseHandle(hThread);

    }

    endTime = clock();

    // Write processing times to the text file

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

    // Delete the critical section

    DeleteCriticalSection(&cs);

    return 0;

}

## **Программа C**

#include <windows.h>

#include <iostream>

#include <fstream>

#include <sstream>

#include <string>

#include <unordered\_set>

#include <vector>

#include <algorithm>

#include <ctime>

#include "util.h"

// Struct to store file processing time details

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;

// Initializing default directories

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

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

// Function to trim leading and trailing whitespace

std::string trimWhitespace(const std::string &str) {

    auto start = str.find\_first\_not\_of(" \t");

    auto end = str.find\_last\_not\_of(" \t");

    return (start == std::string::npos || end == std::string::npos) ? "" : str.substr(start, end - start + 1);

}

// Thread function to process a file and sort lines by the first character

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::vector<std::string> lines;

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

        line = trimWhitespace(line);

        lines.push\_back(line);

    }

    // Sort lines by the first character, considering that some lines may be empty

    std::sort(lines.begin(), lines.end(), [](const std::string &a, const std::string &b) {

        return a.empty() || b.empty() ? a < b : a[0] < b[0];

    });

    for (const auto &sortedLine : lines) {

        outFile << sortedLine << '\n';

    }

    inFile.close();

    outFile.close();

    endFileTime = clock();

    processingTime.start\_time = startFileTime;

    processingTime.end\_time = endFileTime;

    // Use critical section to safely add data to shared vector

    EnterCriticalSection(&cs);

    fileProcessingTimes.push\_back(processingTime);

    LeaveCriticalSection(&cs);

    std::cout << "Lines sorted and saved to '" << outputFilename << "'." << std::endl;

    return 0;

}

// Function to write processing times to a text file

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 = "E:\\files\\in\\";

        outputDirectory = "E:\\files\\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];

        // Create a pair of input and output filenames and pass to the thread

        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);

        }

    }

    // Wait for all threads to finish

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

    // Close all thread handles

    for (auto &hThread : threadHandles) {

        CloseHandle(hThread);

    }

    endTime = clock();

    // Write processing times to the text file

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

    // Delete the critical section

    DeleteCriticalSection(&cs);

    return 0;

}

# РЕЗУЛЬТАТ РАБОТЫ ПРОГРАММ

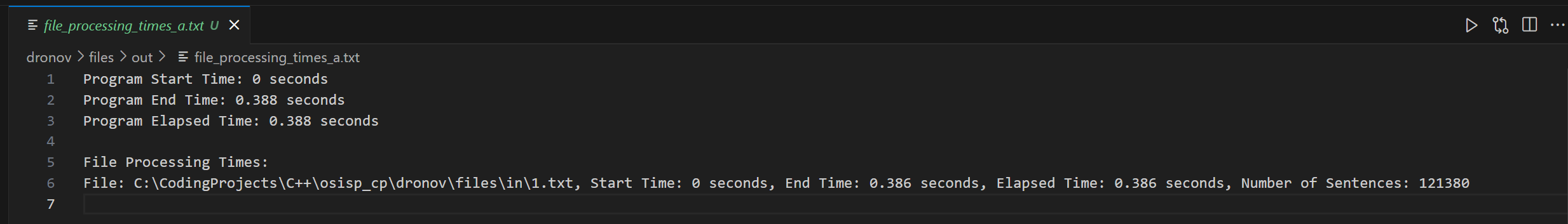


Рисунок 1 – Работа программы AH с 1 текстовым файлом

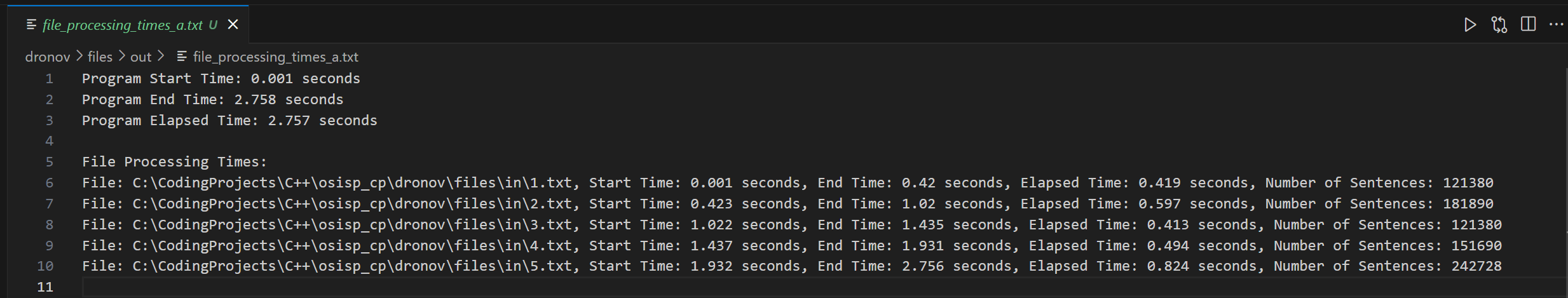


Рисунок 2 – Работа программы AH с 5 текстовыми файлами

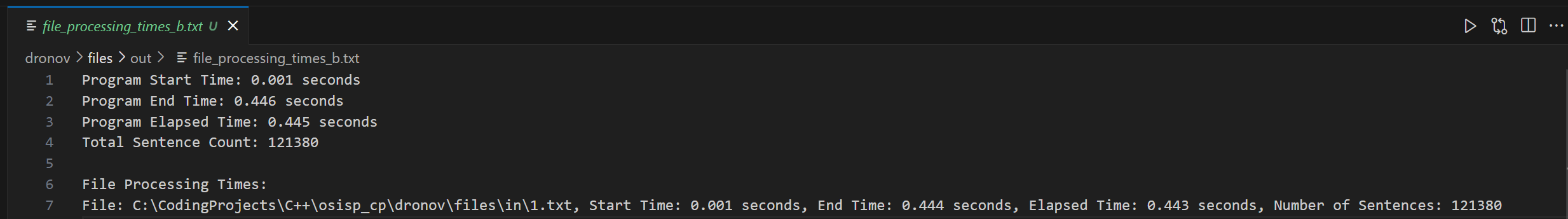


Рисунок 3 – Работа программы BH с 1 текстовым файлом

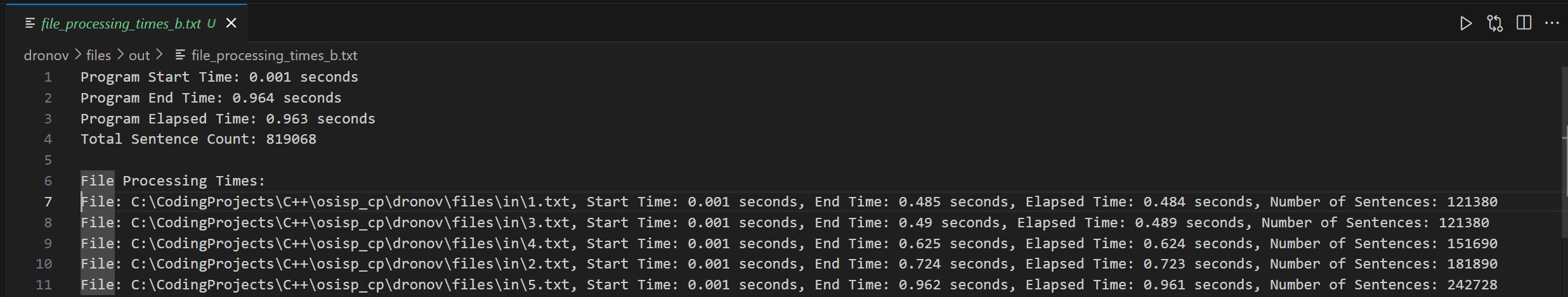


Рисунок 4 – Работа программы BH с 5 текстовыми файлами

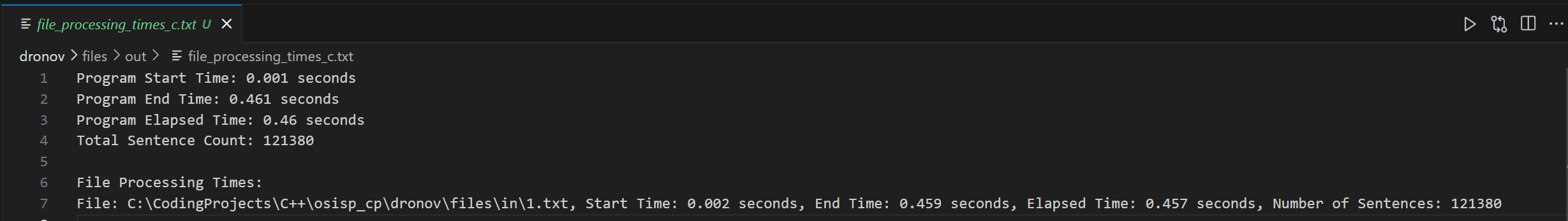


Рисунок 5 – Работа программы СH с 1 текстовым файлом

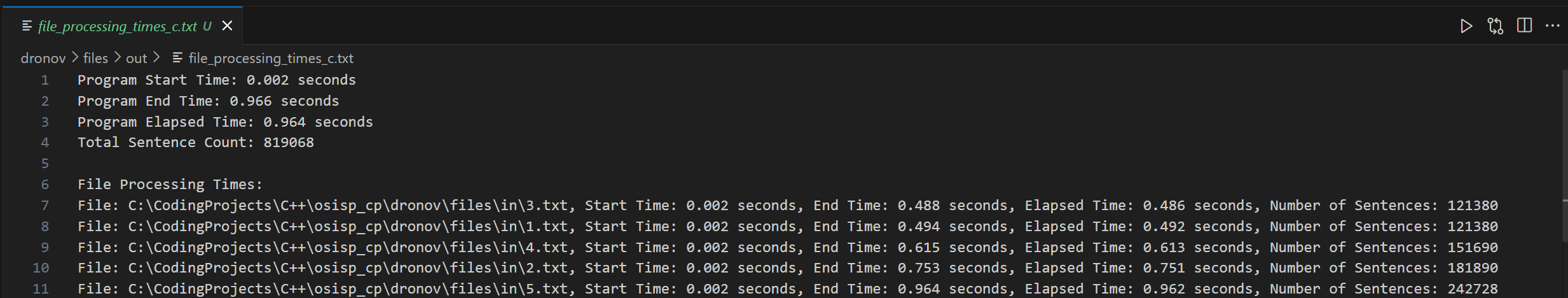


Рисунок 6– Работа программы CH с 5 текстовыми файлами

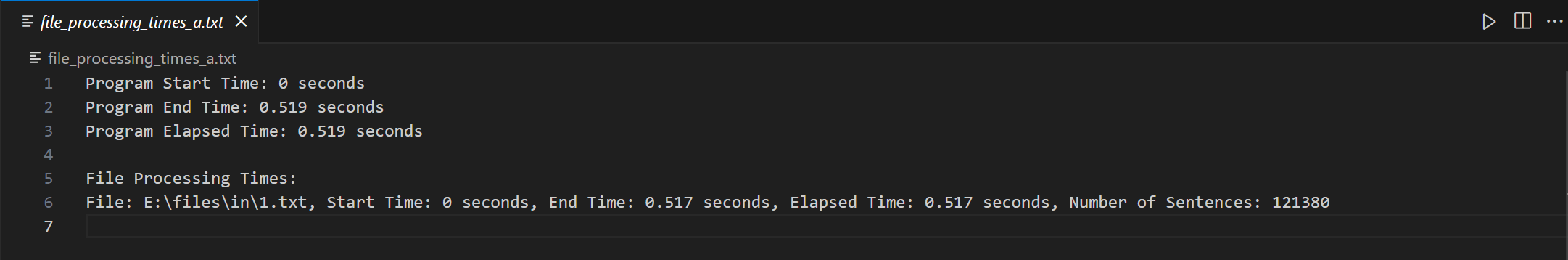


Рисунок 7 – Работа программы AF с 1 текстовым файлом

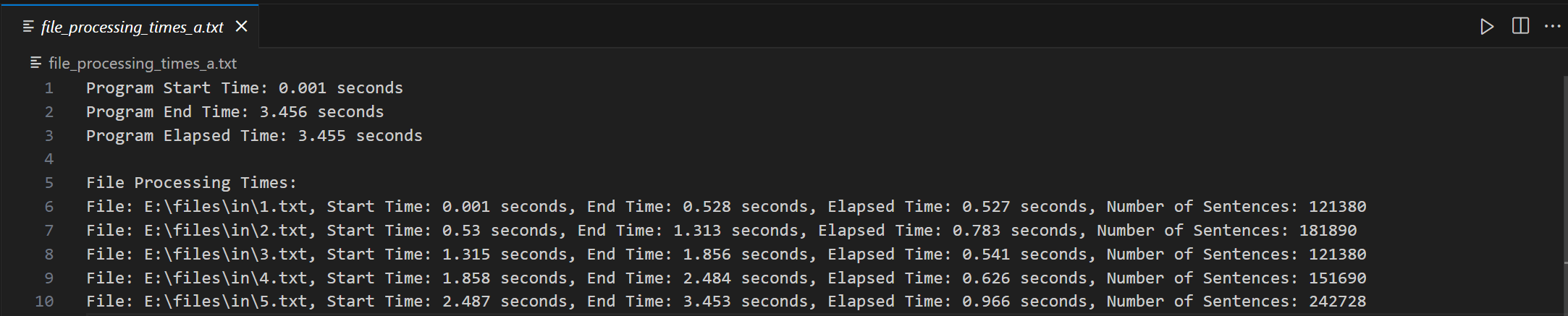


Рисунок 8 – Работа программы AF с 5 текстовыми файлами

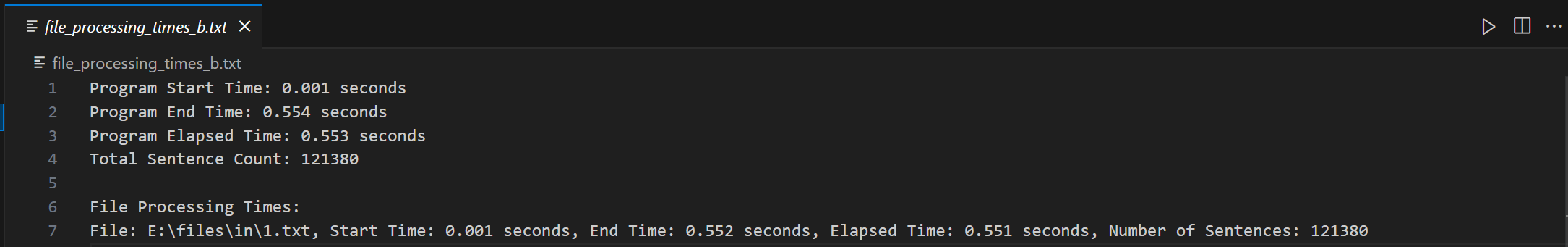


Рисунок 9 – Работа программы BF с 1 текстовым файлом

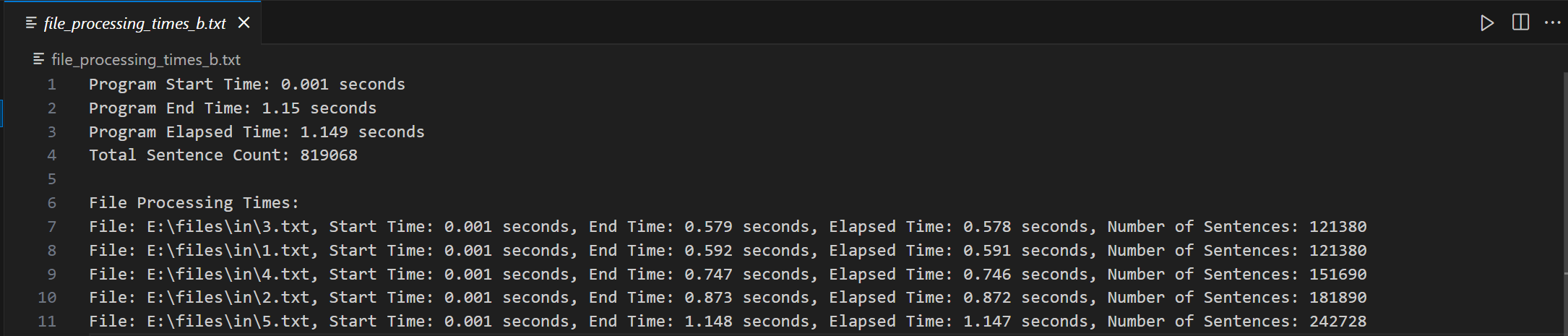


Рисунок 10 – Работа программы BF с 5 текстовыми файлами

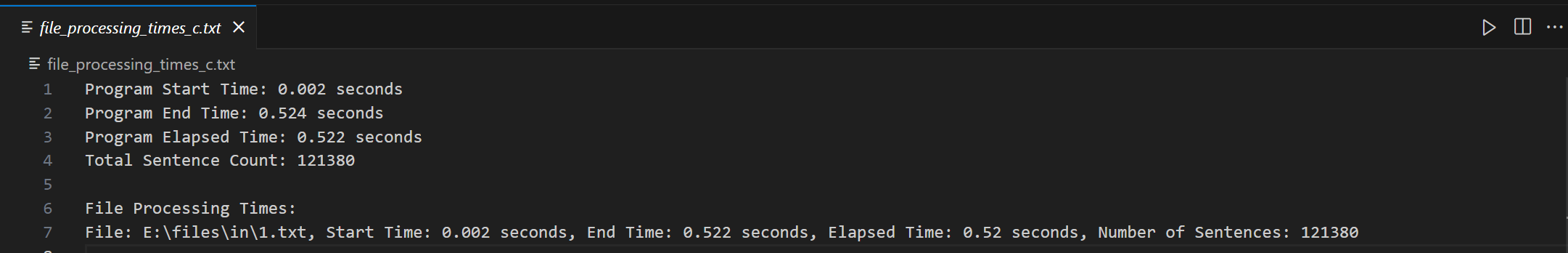


Рисунок 11 – Работа программы CF с 1 текстовым файлом

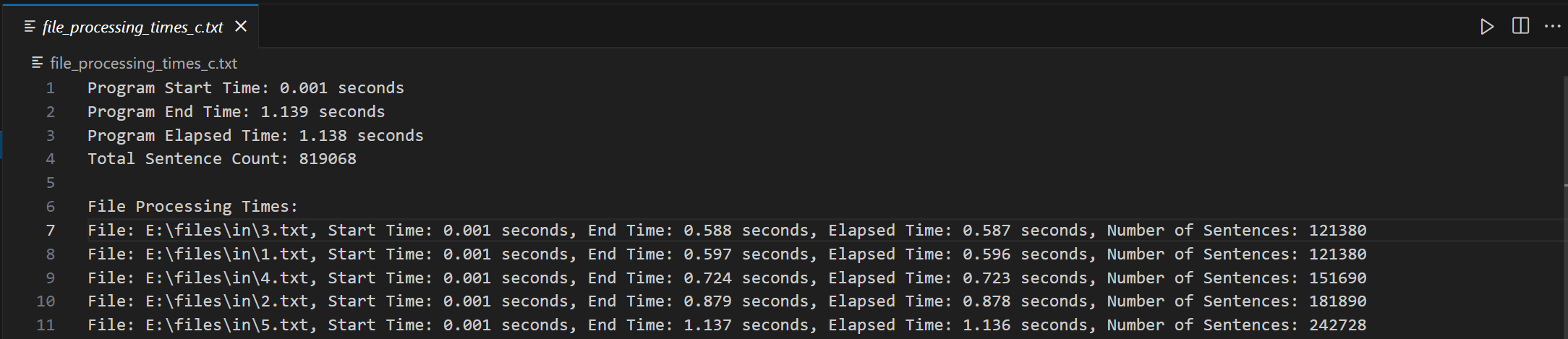


Рисунок 12 – Работа программы CF с 5 текстовыми файлами

# СКРИНШОТЫ ОКНА УТИЛИТЫ PROCESS EXPLORER

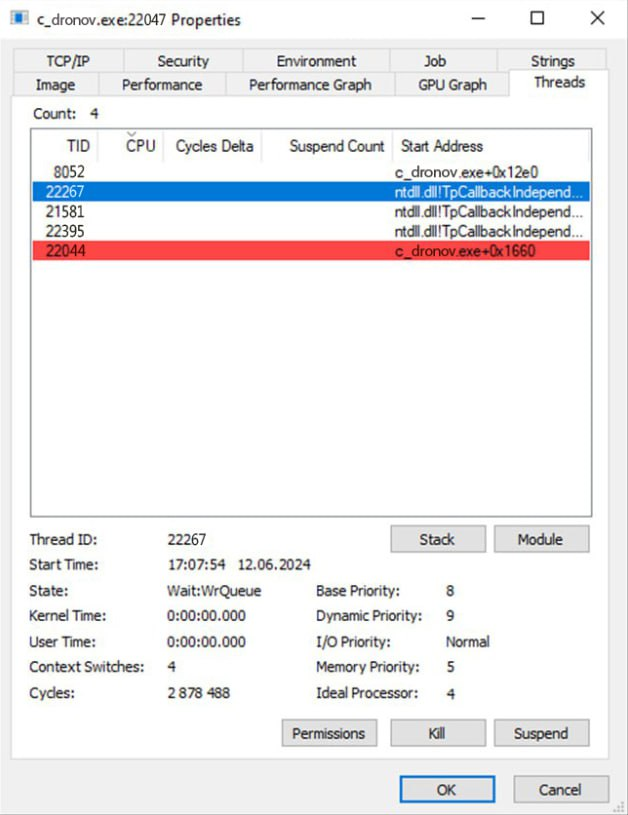


Рисунок 13 – Дочерние потоки программы С

**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 | 410 | 430 | 420 | 415 | 440 | 423 |
| AH | 2 | 1003 | 1020 | 1015 | 1035 | 1008 | 1016,2 |
| AH | 3 | 1397 | 1400 | 1385 | 1410 | 1375 | 1393,4 |
| AH | 4 | 1928 | 1940 | 1915 | 1935 | 1955 | 1934,6 |
| AH | 5 | 2711 | 2730 | 2700 | 2725 | 2695 | 2712,2 |
| AH | 6 | 3506 | 3490 | 3520 | 3515 | 3480 | 3502,2 |
| AH | 7 | 4515 | 4530 | 4500 | 4540 | 4520 | 4521 |
| AH | 8 | 5556 | 5580 | 5565 | 5540 | 5575 | 5563,2 |
| AH | 9 | 6190 | 6180 | 6205 | 6175 | 6210 | 6192 |
| AH | 10 | 6388 | 6400 | 6375 | 6395 | 6360 | 6383,6 |
| BH | 1 | 489 | 495 | 485 | 490 | 500 | 491,8 |
| BH | 2 | 709 | 700 | 715 | 720 | 705 | 709,8 |
| BH | 3 | 725 | 730 | 720 | 735 | 710 | 724 |
| BH | 4 | 731 | 740 | 725 | 735 | 720 | 730,2 |
| BH | 5 | 949 | 945 | 950 | 955 | 940 | 947,8 |
| BH | 6 | 987 | 995 | 980 | 990 | 1000 | 990,4 |
| BH | 7 | 1200 | 1195 | 1210 | 1185 | 1225 | 1203 |
| BH | 8 | 1203 | 1190 | 1215 | 1180 | 1220 | 1201,6 |
| BH | 9 | 1211 | 1200 | 1225 | 1195 | 1230 | 1212,2 |
| BH | 10 | 1224 | 1230 | 1210 | 1240 | 1205 | 1221,8 |
| CH | 1 | 469 | 470 | 460 | 480 | 455 | 466,8 |
| CH | 2 | 688 | 695 | 675 | 690 | 700 | 689,6 |
| CH | 3 | 711 | 705 | 720 | 730 | 710 | 715,2 |
| CH | 4 | 720 | 735 | 725 | 715 | 740 | 727 |
| CH | 5 | 930 | 940 | 915 | 935 | 945 | 933 |
| CH | 6 | 960 | 955 | 970 | 950 | 965 | 960 |
| CH | 7 | 1199 | 1185 | 1205 | 1190 | 1210 | 1197,8 |
| CH | 8 | 1199 | 1210 | 1190 | 1180 | 1205 | 1196,8 |
| CH | 9 | 1200 | 1195 | 1210 | 1185 | 1220 | 1202 |
| CH | 10 | 1220 | 1205 | 1230 | 1215 | 1240 | 1222 |
| AF | 1 | 501 | 510 | 495 | 515 | 520 | 508,2 |
| AF | 2 | 1202 | 1185 | 1210 | 1190 | 1225 | 1202,4 |
| AF | 3 | 1664 | 1650 | 1675 | 1640 | 1690 | 1663,8 |
| AF | 4 | 2300 | 2285 | 2315 | 2290 | 2320 | 2302 |
| AF | 5 | 3310 | 3330 | 3300 | 3325 | 3290 | 3311 |
| AF | 6 | 4404 | 4420 | 4390 | 4410 | 4435 | 4411,8 |
| AF | 7 | 5502 | 5510 | 5490 | 5530 | 5480 | 5502,4 |
| AF | 8 | 6656 | 6640 | 6670 | 6620 | 6690 | 6655,2 |
| AF | 9 | 7557 | 7570 | 7540 | 7560 | 7590 | 7563,4 |
| AF | 10 | 7960 | 7945 | 7975 | 7980 | 7950 | 7962 |
| BF | 1 | 558 | 565 | 550 | 570 | 560 | 560,6 |
| BF | 2 | 855 | 850 | 865 | 845 | 875 | 858 |
| BF | 3 | 944 | 955 | 935 | 950 | 960 | 948,8 |
| BF | 4 | 972 | 960 | 980 | 965 | 990 | 973,4 |
| BF | 5 | 1153 | 1160 | 1145 | 1170 | 1135 | 1152,6 |
| BF | 6 | 1339 | 1 325 | 1350 | 1340 | 1315 | 1333,8 |
| BF | 7 | 1426 | 1430 | 1415 | 1440 | 1405 | 1423,2 |
| BF | 8 | 1462 | 1470 | 1450 | 1480 | 1445 | 1461,4 |
| BF | 9 | 1499 | 1505 | 1485 | 1510 | 1490 | 1497,8 |
| BF | 10 | 1506 | 1520 | 1495 | 1515 | 1500 | 1507,2 |
| CF | 1 | 542 | 555 | 540 | 560 | 545 | 548,4 |
| CF | 2 | 845 | 835 | 850 | 830 | 860 | 844 |
| CF | 3 | 859 | 870 | 845 | 865 | 855 | 858,8 |
| CF | 4 | 883 | 890 | 875 | 885 | 870 | 880,6 |
| CF | 5 | 1141 | 1150 | 1130 | 1160 | 1125 | 1141,2 |
| CF | 6 | 1146 | 1135 | 1160 | 1125 | 1150 | 1143,2 |
| CF | 7 | 1422 | 1435 | 1415 | 1440 | 1405 | 1423,4 |
| CF | 8 | 1450 | 1460 | 1440 | 1455 | 1430 | 1447 |
| CF | 9 | 1451 | 1465 | 1440 | 1470 | 1455 | 1456,2 |
| CF | 10 | 1472 | 1480 | 1460 | 1475 | 1455 | 1468,4 |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 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 | 423 | 1016,2 | 1393,4 | 1934,6 | 2712,2 | 3502,2 | 4521 | 5563,2 | 6192 | 6383,6 |
| B | 491,8 | 709,8 | 724 | 730,2 | 947,8 | 990,4 | 1203 | 1201,6 | 1212,2 | 1221,8 |
| C | 466,8 | 689,6 | 715,2 | 727 | 933 | 960 | 1197,8 | 1196,8 | 1202 | 1222 |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 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 |
| 508,2 | 1202,4 | 1663,8 | 2302 | 3311 | 4411,8 | 5502,4 | 6655,2 | 7557 | 7962 |
| 560,6 | 858 | 948,8 | 973,4 | 1152,6 | 1333,8 | 1423,2 | 1461,4 | 1497,8 | 1507,2 |
| 548,4 | 844 | 858,8 | 880,6 | 1141,2 | 1143,2 | 1423,4 | 1447 | 1456,2 | 1468,4 |

# ДИАГРАММЫ ТРАСС ПОТОКОВ

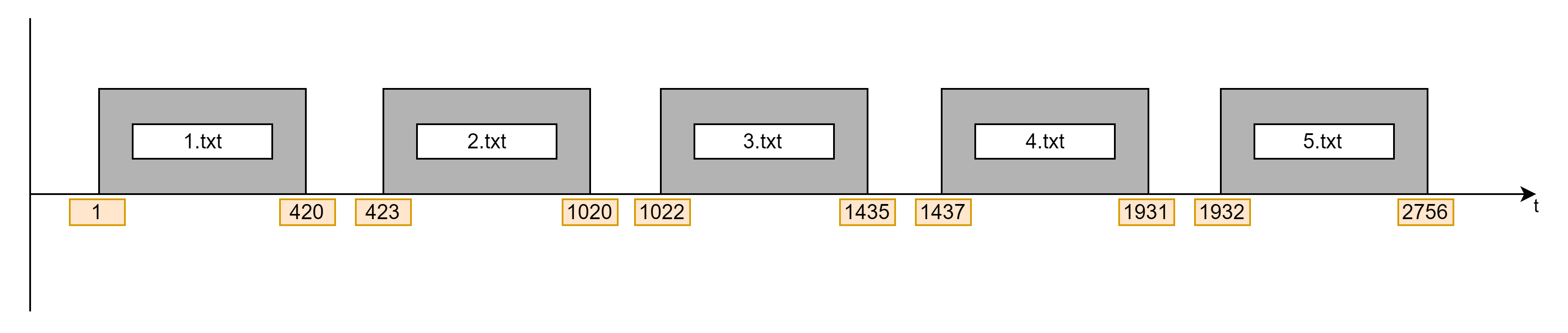


Рисунок 14 – Диаграмма трасс потоков программы AH с 5 текстовыми файлами

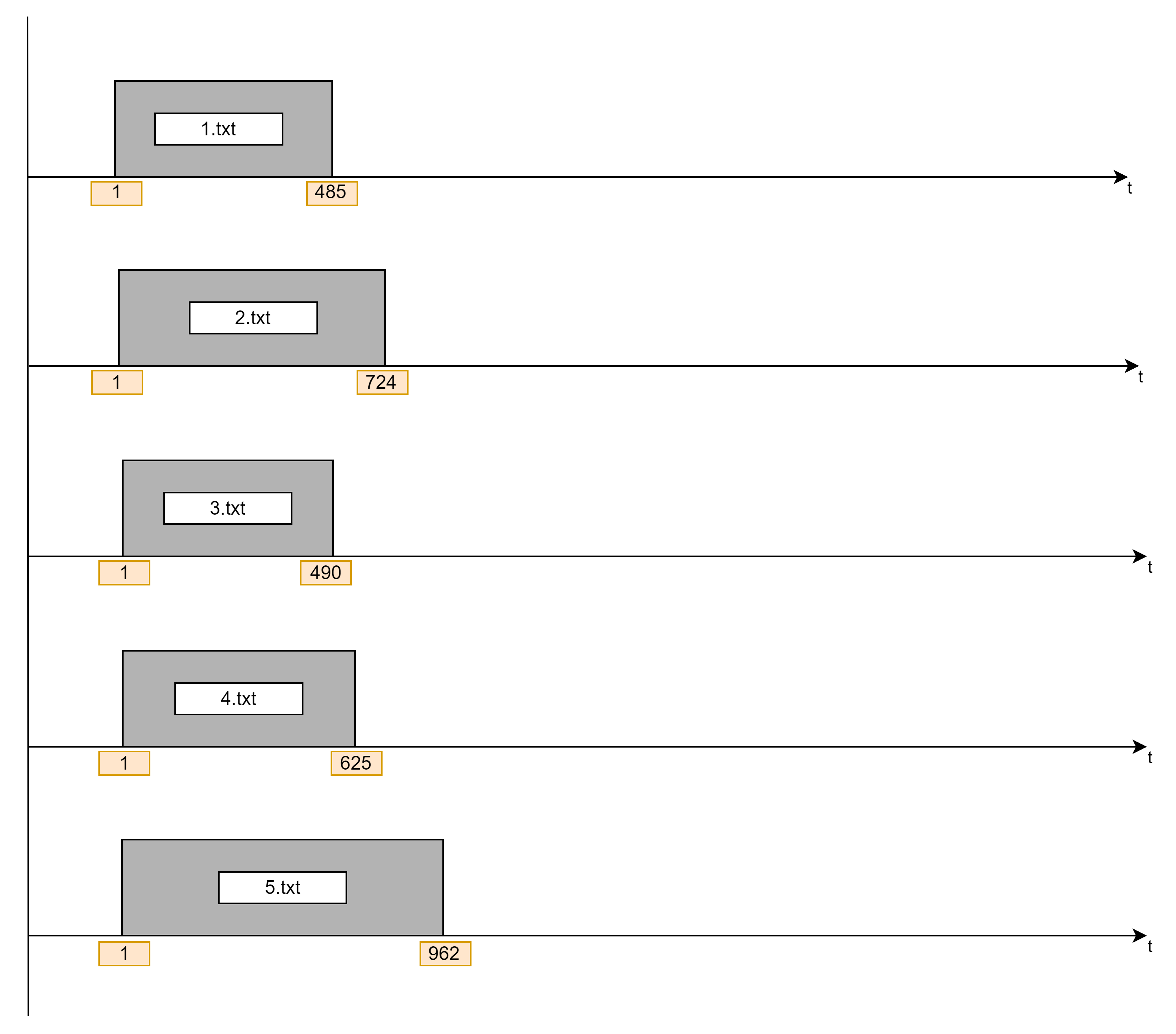


Рисунок 15 – Диаграмма трасс потоков программы BH с 5 текстовыми файлам

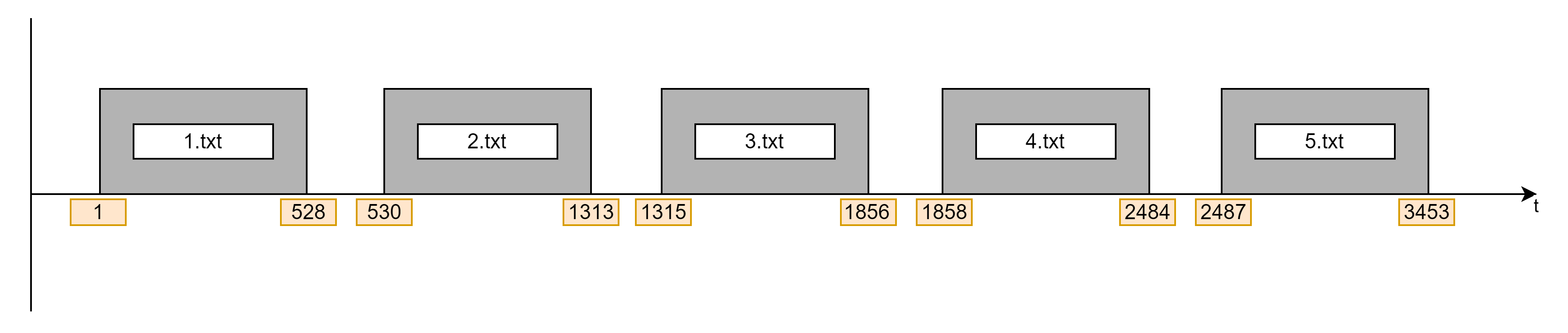


Рисунок 16 – Диаграмма трасс потоков программы AF с 5 текстовыми файлами

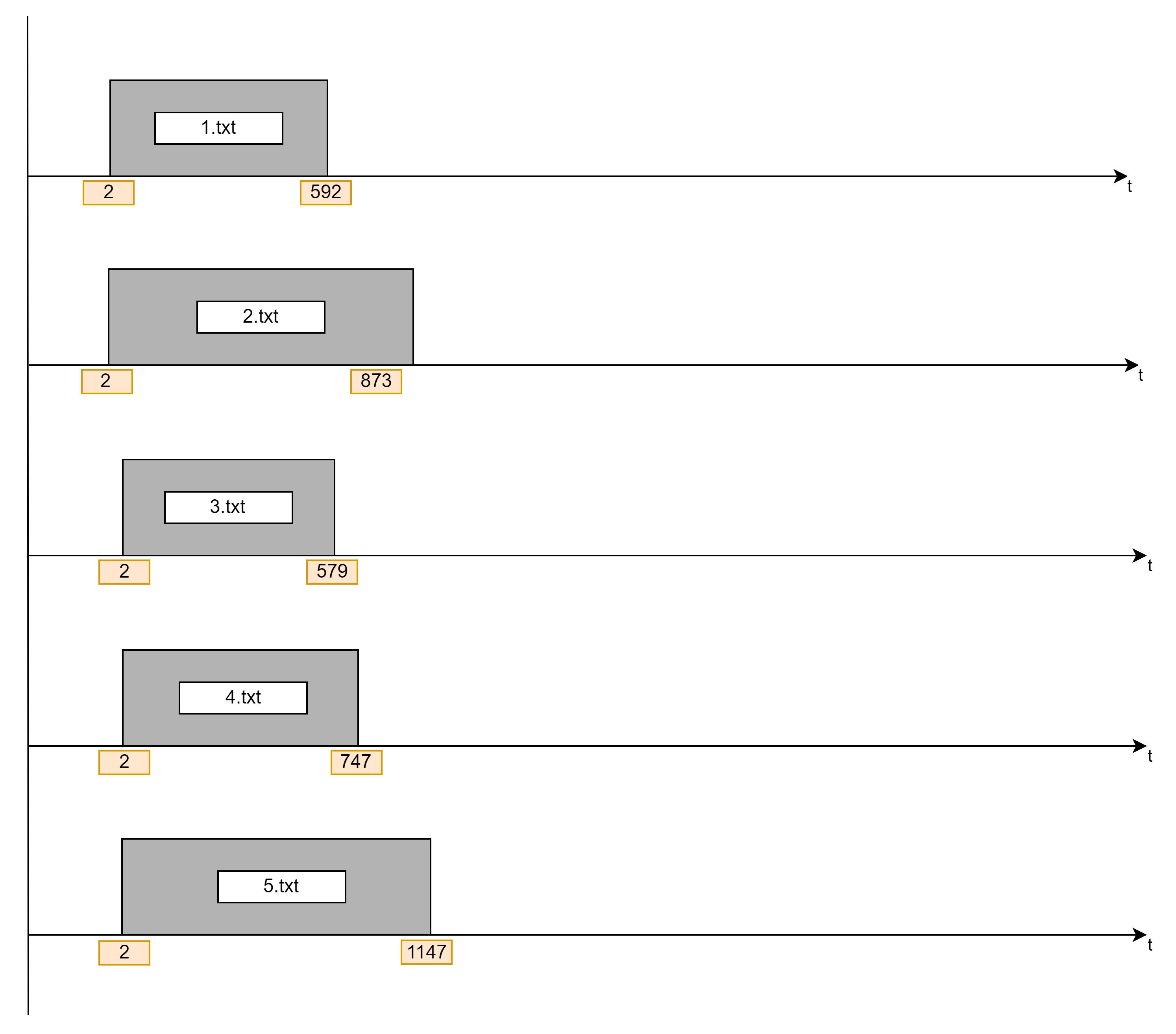


Рисунок 17 – Диаграмма трасс потоков программы BF с 5 текстовыми файлами

# ВЫВОДЫ ПО РЕЗУЛЬТАТАМ ПРОВЕДЕННГО ЭКСПЕРИМЕНТА

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

В многопоточном режиме (B) время обработки файлов было значительно меньше, чем в однопоточном режиме (A). Например, при обработке 10 файлов на жестком диске в режиме A среднее время составило 6383,6 мс, в то время как в режиме B — всего 1221,8 мс. Это показывает, что многопоточность значительно увеличивает производительность при большом количестве файлов за счет параллельной обработки.

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

за 1507,2 мс. Разница особенно заметна при увеличении количества файлов. При работе с flash-накопителем (F) многопоточный режим также показал улучшение производительности. Среднее время обработки 10 файлов в однопоточном режиме (A) составило 7962 мс, тогда как многопоточный режим (B) обработал их

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

Диаграммы трасс потоков показывают, что многопоточный режим распределяет задачи по нескольким потокам, позволяя выполнять их параллельно. Это снижает общее время ожидания завершения операций ввода/вывода и повышает эффективность использования процессора. В однопоточном режиме выполнение задач происходит последовательно, что увеличивает общее время обработки.

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

Повышение приоритета потоков (в программе C) не привело к заметному увеличению производительности. В многопоточном режиме с повышенным приоритетом (C) время обработки файлов лишь незначительно отличалось от времени в обычном многопоточном режиме (B). Это свидетельствует о том, что в данном эксперименте повышение приоритета потоков не оказало существенного влияния на ускорение обработки, особенно при небольшом объеме данных.

**Влияние количества ядер процессора на результаты**:

Увеличение количества ядер процессора также положительно сказалось на производительности. Многопоточные программы смогли использовать больше процессорных ресурсов, что привело к сокращению времени обработки файлов по сравнению с однопоточным режимом, который не может эффективно использовать многозадачность.

**Необходимость использования средств синхронизации потоков**:

В многопоточных программах (B и C) были использованы критические секции для синхронизации доступа к общим ресурсам. Это было необходимо для предотвращения конфликтов и корректной записи результатов, что обеспечивает корректное выполнение программы и точность данных.

# 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.