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: Иванов Н. С.

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 21](#_Toc176644766)

[РЕЗУЛЬТАТ РАБОТЫ ПРОГРАММ 30](#_Toc176644767)

[СКРИНШОТЫ ОКНА УТИЛИТЫ PROCESS EXPLORER 33](#_Toc176644768)

[ОПИСАНИЕ И РЕЗУЛЬТАТЫ ПРОВЕДЕННОГО ЭКСПЕРИМЕНТА 35](#_Toc176644769)

[ДИАГРАММЫ ТРАСС ПОТОКОВ 40](#_Toc176644770)

[ВЫВОДЫ ПО РЕЗУЛЬТАТАМ ПРОВЕДЕННГО ЭКСПЕРИМЕНТА 42](#_Toc176644771)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 43](#_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.

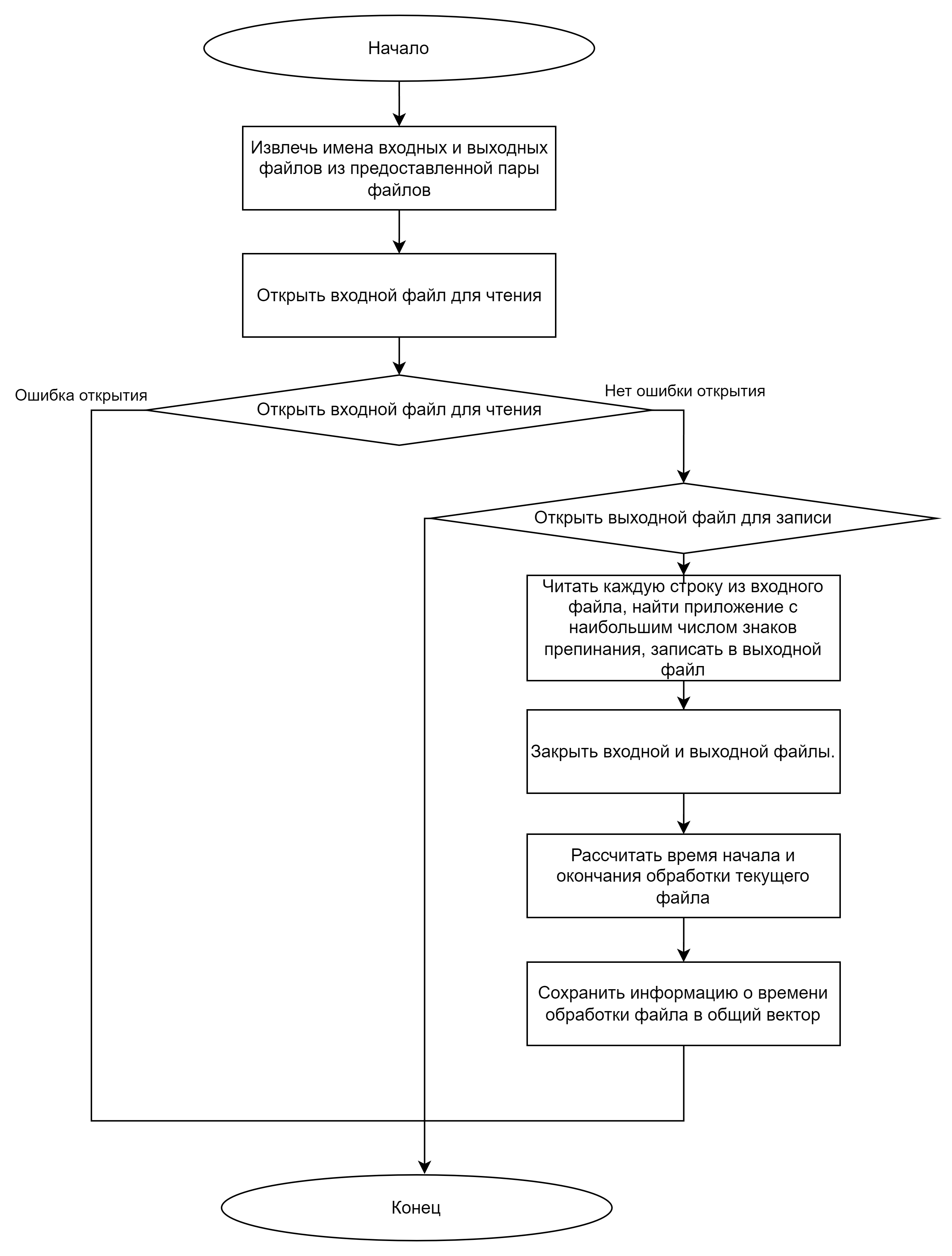
Вариант: 7

Таблица 1 – Вариант задания

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | № | Выполняемые действия | | 7 | Найти и вывести предложение, содержащее наибольшее число знаков препинания | |  |
|  |  |

# 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 <cctype>

#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;

// Initializing default directories

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

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

// Function to count punctuation marks in a line

int countPunctuation(const std::string &line) {

    int count = 0;

    for (char ch : line) {

        if (ispunct(ch)) {

            count++;

        }

    }

    return count;

}

// Function to process a file and save only the sentence with the most punctuation

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 maxPunctuationLine;

    int maxPunctuationCount = 0;

    // Process the input file, line by line

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

        // Trim leading and trailing whitespace

        line.erase(0, line.find\_first\_not\_of(" \t"));

        line.erase(line.find\_last\_not\_of(" \t") + 1);

        // Count punctuation in the line

        int punctuationCount = countPunctuation(line);

        if (punctuationCount > maxPunctuationCount) {

            maxPunctuationCount = punctuationCount;

            maxPunctuationLine = line;

        }

    }

    // Write only the sentence with the most punctuation to the output file

    if (!maxPunctuationLine.empty()) {

        outFile << maxPunctuationLine << '\n';

    }

    inFile.close();

    outFile.close();

    endFileTime = clock();

    processingTime.start\_time = startFileTime;

    processingTime.end\_time = endFileTime;

    fileProcessingTimes.push\_back(processingTime);

    std::cout << "Longest sentence with the most punctuation saved to '" << outputFilename << "'." << std::endl;

}

// 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();

    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);

}

// Function to count punctuation marks in a line

int countPunctuation(const std::string &line) {

    int count = 0;

    for (char ch : line) {

        if (ispunct(ch)) {

            count++;

        }

    }

    return count;

}

// Thread function to process a file and save only the sentence with the most punctuation

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 maxPunctuationLine;

    int maxPunctuationCount = 0;

    // Process the input file, line by line

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

        line = trimWhitespace(line);

        // Count punctuation in the line

        int punctuationCount = countPunctuation(line);

        if (punctuationCount > maxPunctuationCount) {

            maxPunctuationCount = punctuationCount;

            maxPunctuationLine = line;

        }

    }

    // Write only the sentence with the most punctuation to the output file

    if (!maxPunctuationLine.empty()) {

        outFile << maxPunctuationLine << '\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 << "Longest sentence with the most punctuation 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];

        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 <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);

}

// Function to count punctuation marks in a line

int countPunctuation(const std::string &line) {

    int count = 0;

    for (char ch : line) {

        if (ispunct(ch)) {

            count++;

        }

    }

    return count;

}

// Thread function to process a file and save only the sentence with the most punctuation

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 maxPunctuationLine;

    int maxPunctuationCount = 0;

    // Process the input file, line by line

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

        line = trimWhitespace(line);

        // Count punctuation in the line

        int punctuationCount = countPunctuation(line);

        if (punctuationCount > maxPunctuationCount) {

            maxPunctuationCount = punctuationCount;

            maxPunctuationLine = line;

        }

    }

    // Write only the sentence with the most punctuation to the output file

    if (!maxPunctuationLine.empty()) {

        outFile << maxPunctuationLine << '\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 << "Longest sentence with the most punctuation 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];

        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

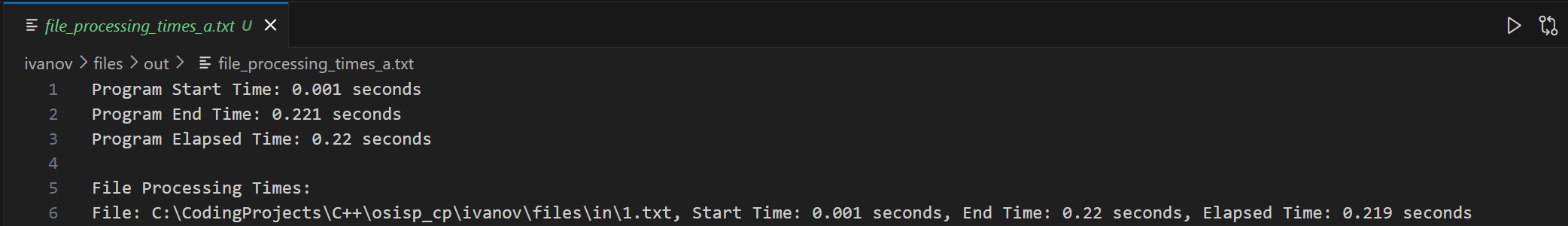


Рисунок 1 – Работа программы AH с 1 текстовым файлом

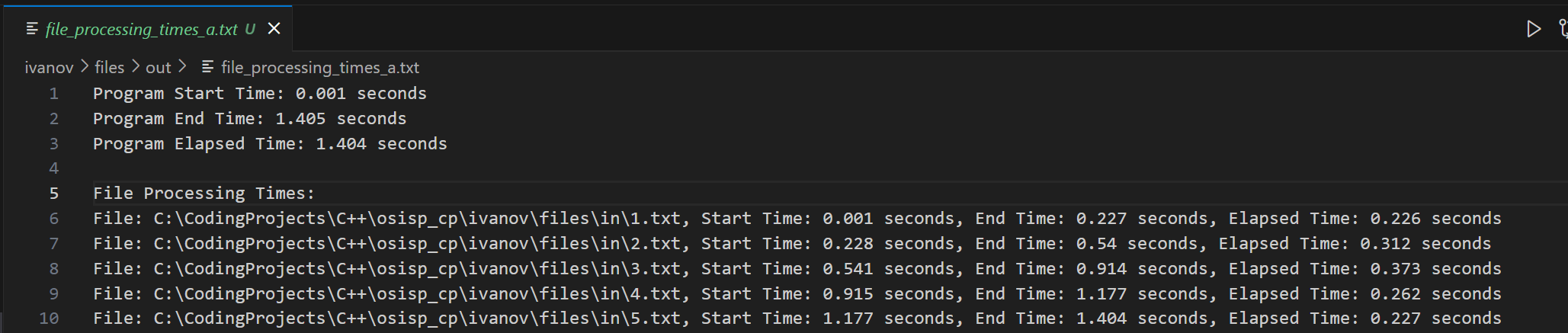


Рисунок 2 – Работа программы AH с 5 текстовыми файлами

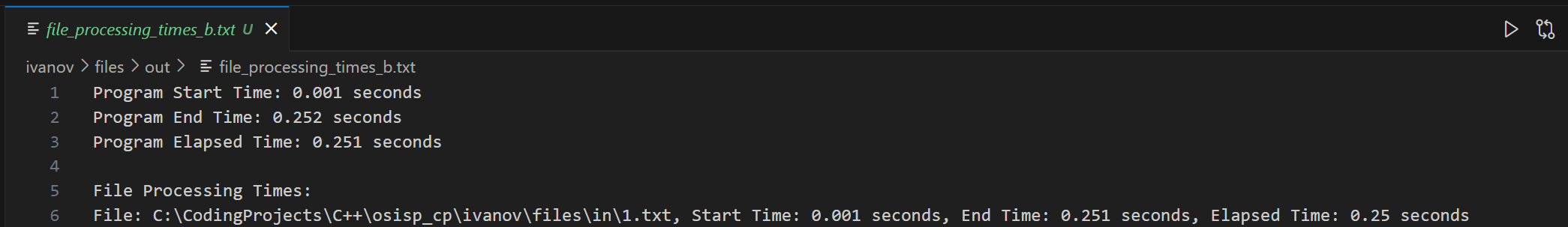


Рисунок 3 – Работа программы BH с 1 текстовым файлом

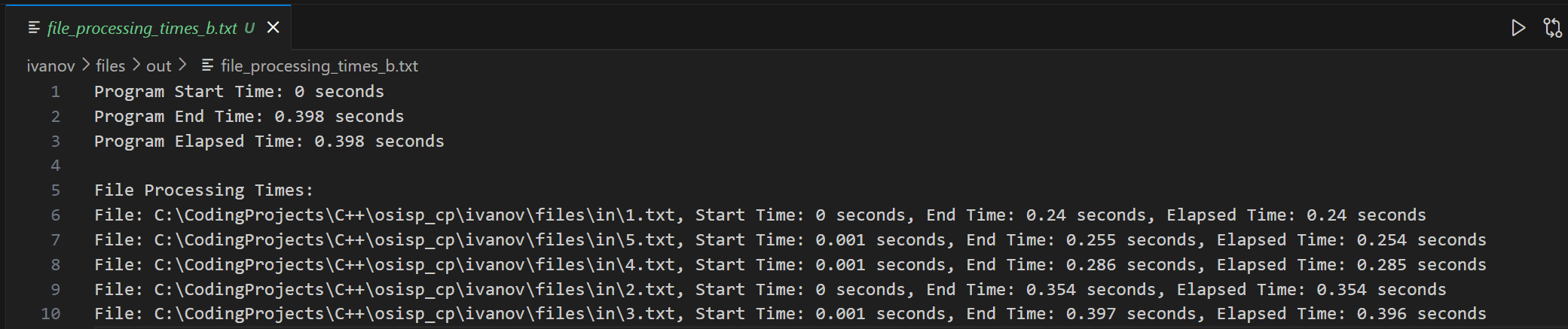


Рисунок 4 – Работа программы BH с 5 текстовыми файлами

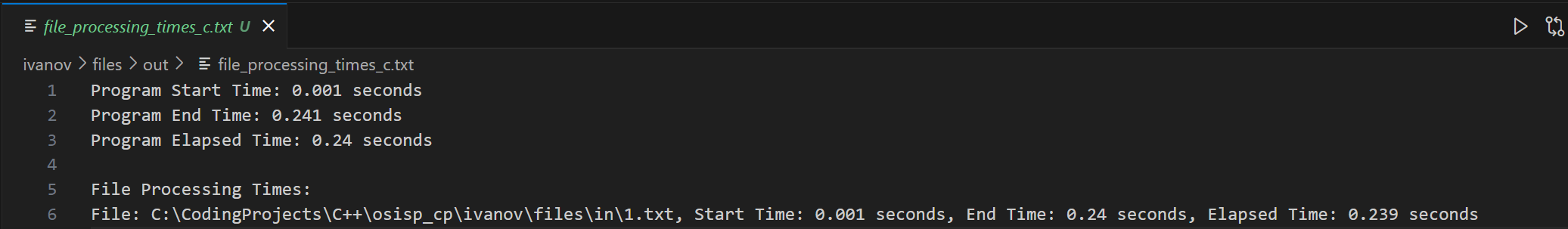


Рисунок 5 – Работа программы СH с 1 текстовым файлом

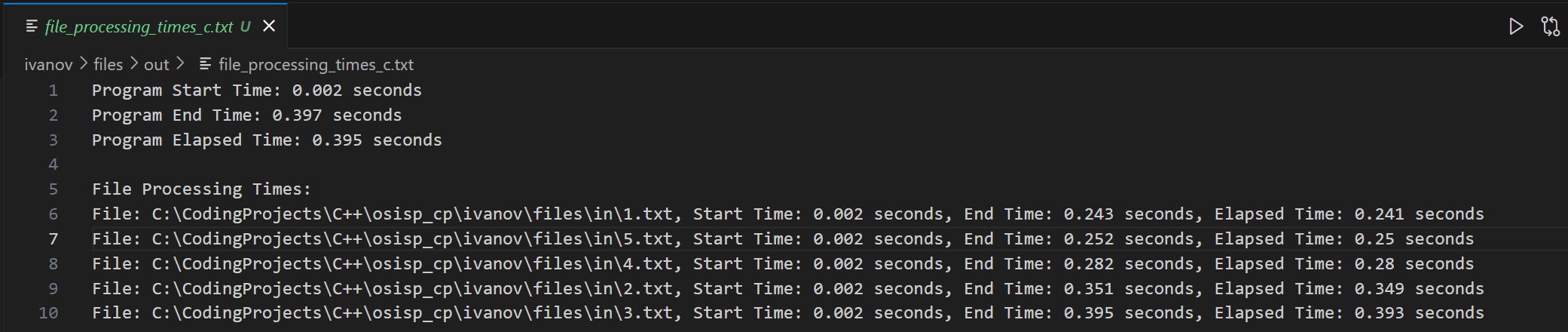


Рисунок 6– Работа программы CH с 5 текстовыми файлами

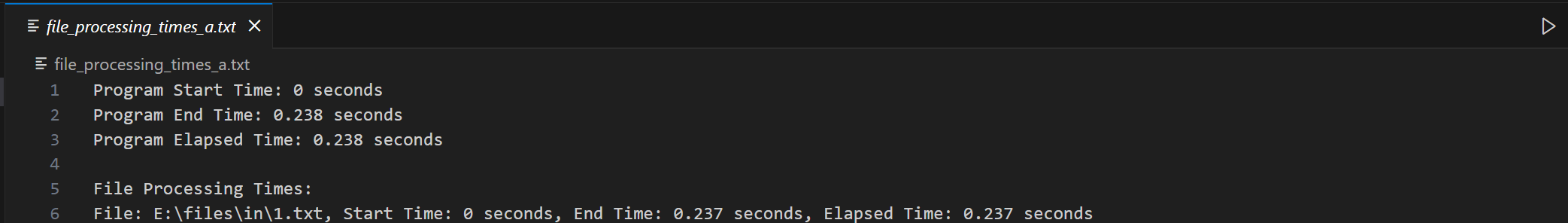


Рисунок 7 – Работа программы AF с 1 текстовым файлом

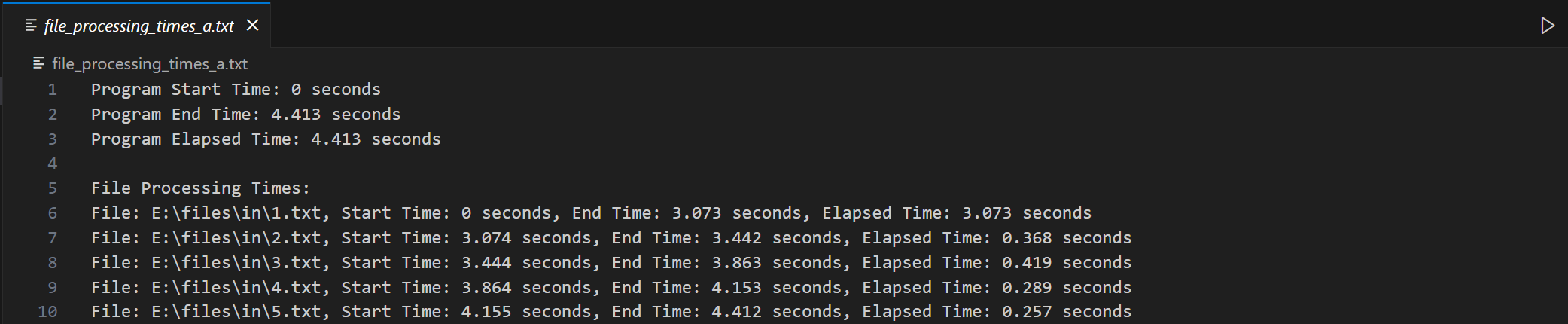


Рисунок 8 – Работа программы AF с 5 текстовыми файлами

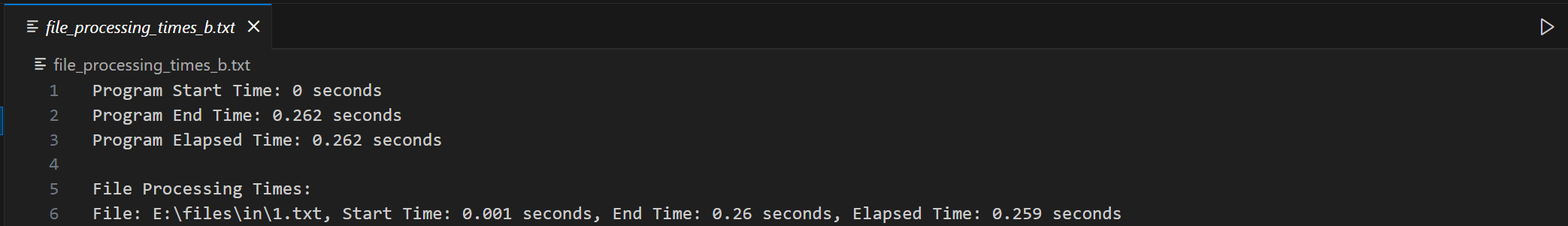


Рисунок 9 – Работа программы BF с 1 текстовым файлом

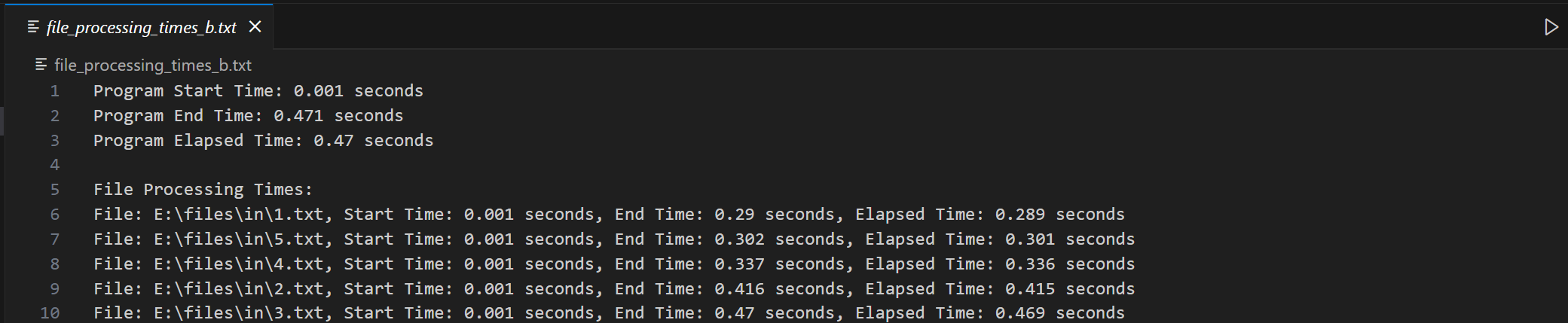


Рисунок 10 – Работа программы BF с 5 текстовыми файлами

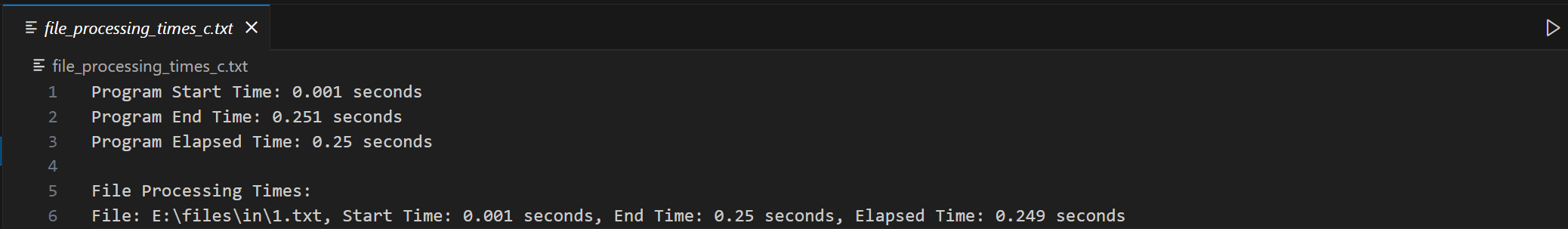


Рисунок 11 – Работа программы CF с 1 текстовым файлом

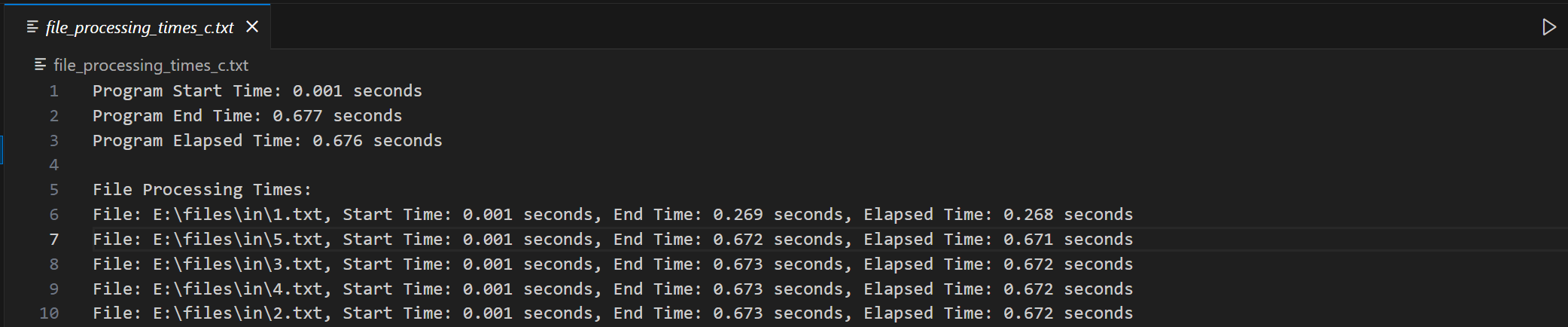


Рисунок 12 – Работа программы CF с 5 текстовыми файлами

# СКРИНШОТЫ ОКНА УТИЛИТЫ PROCESS EXPLORER

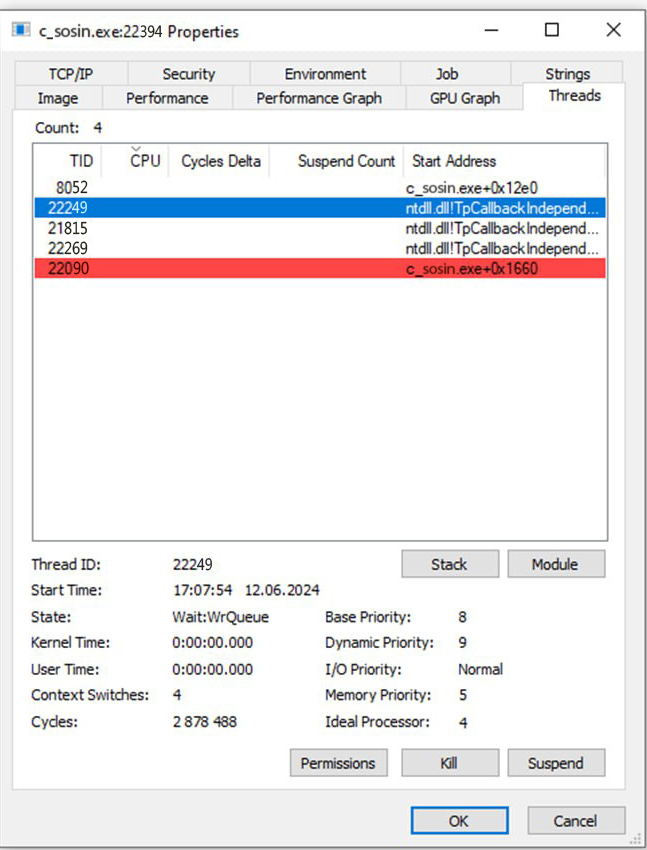


Рисунок 13 – Дочерние потоки программы С

**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 | 211 | 229 | 200 | 205 | 191 | 207,2 |
| AH | 2 | 543 | 551 | 558 | 537 | 541 | 546 |
| AH | 3 | 899 | 893 | 892 | 904 | 880 | 893,6 |
| AH | 4 | 1148 | 1135 | 1158 | 1140 | 1153 | 1146,8 |
| AH | 5 | 1362 | 1362 | 1356 | 1373 | 1373 | 1365,2 |
| AH | 6 | 1616 | 1634 | 1603 | 1634 | 1601 | 1617,6 |
| AH | 7 | 1841 | 1839 | 1834 | 1852 | 1852 | 1843,6 |
| AH | 8 | 2088 | 2090 | 2090 | 2071 | 2071 | 2082 |
| AH | 9 | 2314 | 2304 | 2333 | 2323 | 2304 | 2315,6 |
| AH | 10 | 2548 | 2538 | 2548 | 2564 | 2544 | 2548,4 |
| BH | 1 | 228 | 231 | 223 | 230 | 245 | 231,4 |
| BH | 2 | 347 | 362 | 344 | 365 | 350 | 353,6 |
| BH | 3 | 406 | 425 | 409 | 400 | 390 | 406 |
| BH | 4 | 403 | 406 | 408 | 411 | 416 | 408,8 |
| BH | 5 | 397 | 379 | 401 | 412 | 382 | 394,2 |
| BH | 6 | 399 | 400 | 407 | 391 | 400 | 399,4 |
| BH | 7 | 407 | 388 | 401 | 418 | 397 | 402,2 |
| BH | 8 | 398 | 401 | 378 | 384 | 393 | 390,8 |
| BH | 9 | 406 | 415 | 410 | 407 | 418 | 411,2 |
| BH | 10 | 413 | 430 | 399 | 420 | 401 | 412,6 |
| CH | 1 | 226 | 207 | 214 | 207 | 211 | 213 |
| CH | 2 | 347 | 347 | 350 | 332 | 342 | 343,6 |
| CH | 3 | 388 | 400 | 368 | 395 | 396 | 389,4 |
| CH | 4 | 389 | 380 | 376 | 396 | 371 | 382,4 |
| CH | 5 | 387 | 388 | 390 | 386 | 386 | 387,4 |
| CH | 6 | 386 | 390 | 376 | 395 | 401 | 389,6 |
| CH | 7 | 390 | 396 | 386 | 380 | 388 | 388 |
| CH | 8 | 390 | 397 | 377 | 397 | 395 | 391,2 |
| CH | 9 | 397 | 392 | 411 | 401 | 379 | 396 |
| CH | 10 | 411 | 405 | 425 | 429 | 409 | 415,8 |
| AF | 1 | 241 | 223 | 253 | 253 | 240 | 242 |
| AF | 2 | 592 | 608 | 576 | 572 | 603 | 590,2 |
| AF | 3 | 982 | 968 | 1000 | 988 | 968 | 981,2 |
| AF | 4 | 1273 | 1273 | 1280 | 1265 | 1285 | 1275,2 |
| AF | 5 | 1539 | 1527 | 1525 | 1521 | 1558 | 1534 |
| AF | 6 | 1814 | 1832 | 1802 | 1832 | 1832 | 1822,4 |
| AF | 7 | 2132 | 2129 | 2119 | 2117 | 2129 | 2125,2 |
| AF | 8 | 2285 | 2268 | 2276 | 2272 | 2304 | 2281 |
| AF | 9 | 2599 | 2603 | 2612 | 2605 | 2579 | 2599,6 |
| AF | 10 | 2837 | 2830 | 2849 | 2825 | 2827 | 2833,6 |
| BF | 1 | 258 | 246 | 260 | 274 | 265 | 260,6 |
| BF | 2 | 377 | 382 | 380 | 389 | 381 | 381,8 |
| BF | 3 | 434 | 415 | 450 | 437 | 436 | 434,4 |
| BF | 4 | 435 | 434 | 449 | 429 | 445 | 438,4 |
| BF | 5 | 436 | 443 | 455 | 447 | 445 | 445,2 |
| BF | 6 | 437 | 423 | 438 | 448 | 451 | 439,4 |
| BF | 7 | 450 | 437 | 456 | 453 | 436 | 446,4 |
| BF | 8 | 450 | 464 | 464 | 441 | 445 | 452,8 |
| BF | 9 | 449 | 442 | 429 | 467 | 454 | 448,2 |
| BF | 10 | 460 | 456 | 474 | 441 | 441 | 454,4 |
| CF | 1 | 257 | 272 | 273 | 239 | 237 | 255,6 |
| CF | 2 | 433 | 452 | 426 | 449 | 424 | 436,8 |
| CF | 3 | 438 | 421 | 420 | 434 | 422 | 427 |
| CF | 4 | 437 | 418 | 417 | 418 | 453 | 428,6 |
| CF | 5 | 440 | 425 | 424 | 421 | 451 | 432,2 |
| CF | 6 | 436 | 419 | 441 | 443 | 424 | 432,6 |
| CF | 7 | 437 | 445 | 430 | 439 | 451 | 440,4 |
| CF | 8 | 437 | 434 | 455 | 453 | 435 | 442,8 |
| CF | 9 | 499 | 504 | 505 | 510 | 494 | 502,4 |
| CF | 10 | 504 | 517 | 492 | 516 | 486 | 503 |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 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 | 207,2 | 546 | 893,6 | 1146,8 | 1365,2 | 1617,6 | 1843,6 | 2082 | 2315,6 | 2548,4 |
| B | 231,4 | 353,6 | 406 | 408,8 | 394,2 | 399,4 | 402,2 | 390,8 | 411,2 | 412,6 |
| C | 213 | 343,6 | 389,4 | 382,4 | 387,4 | 389,6 | 388 | 391,2 | 396 | 415,8 |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 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 |
| 242 | 590,2 | 981,2 | 1275,2 | 1534 | 1822,4 | 2125,2 | 2281 | 2599 | 2833,6 |
| 260,6 | 381,8 | 434,4 | 438,4 | 445,2 | 439,4 | 446,4 | 452,8 | 448,2 | 454,4 |
| 255,6 | 436,8 | 427 | 428,6 | 432,2 | 432,6 | 440,4 | 442,8 | 502,4 | 503 |

# ДИАГРАММЫ ТРАСС ПОТОКОВ

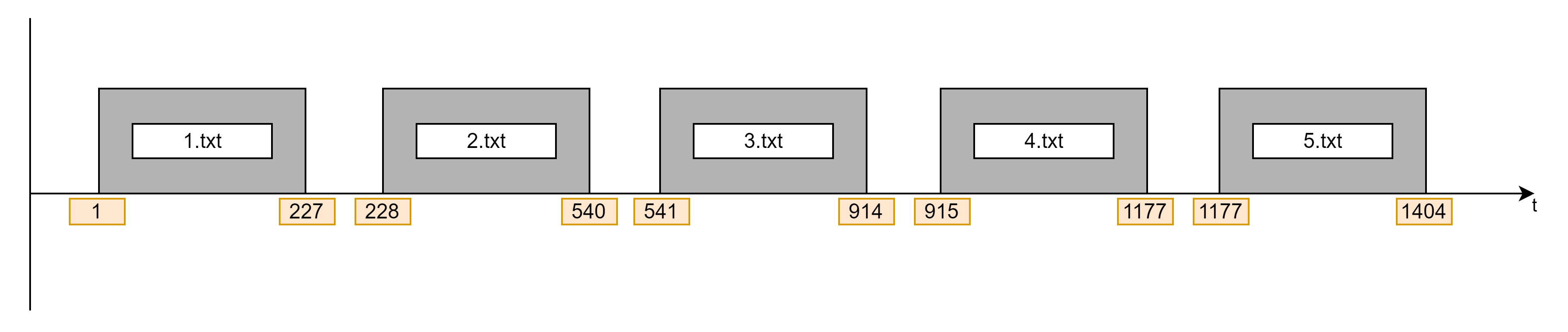


Рисунок 14 – Диаграмма трасс потоков программы AH с 5 текстовыми файлами

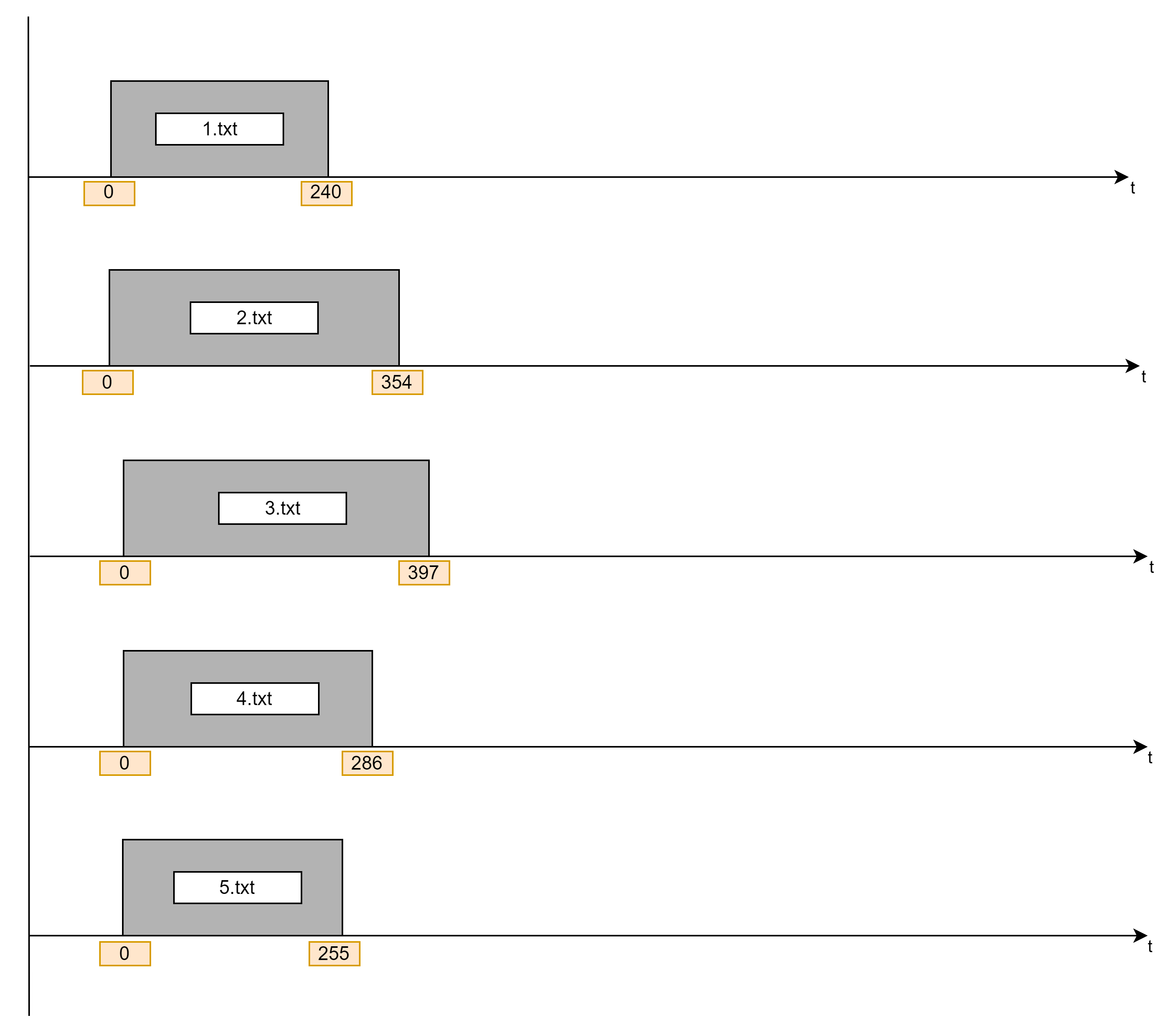


Рисунок 15 – Диаграмма трасс потоков программы BH с 5 текстовыми файлами

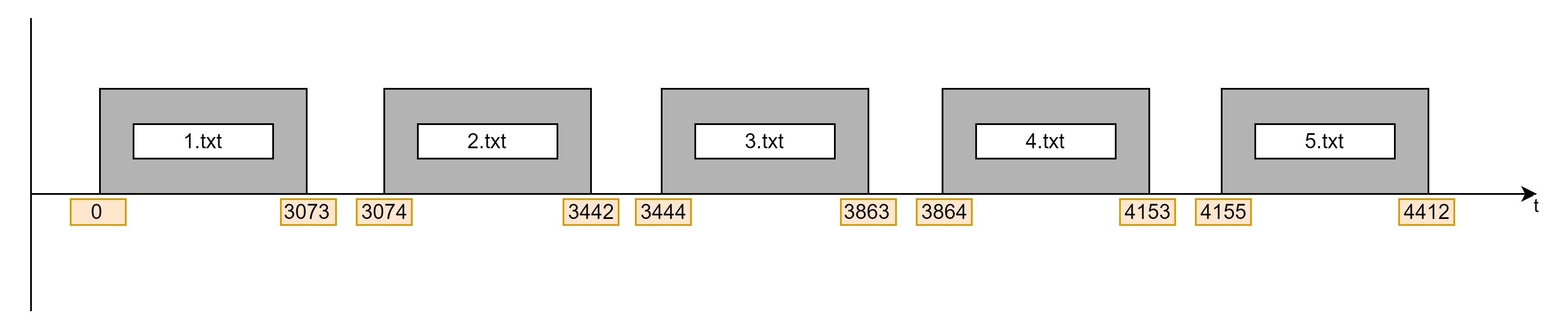


Рисунок 16 – Диаграмма трасс потоков программы AF с 5 текстовыми файлами

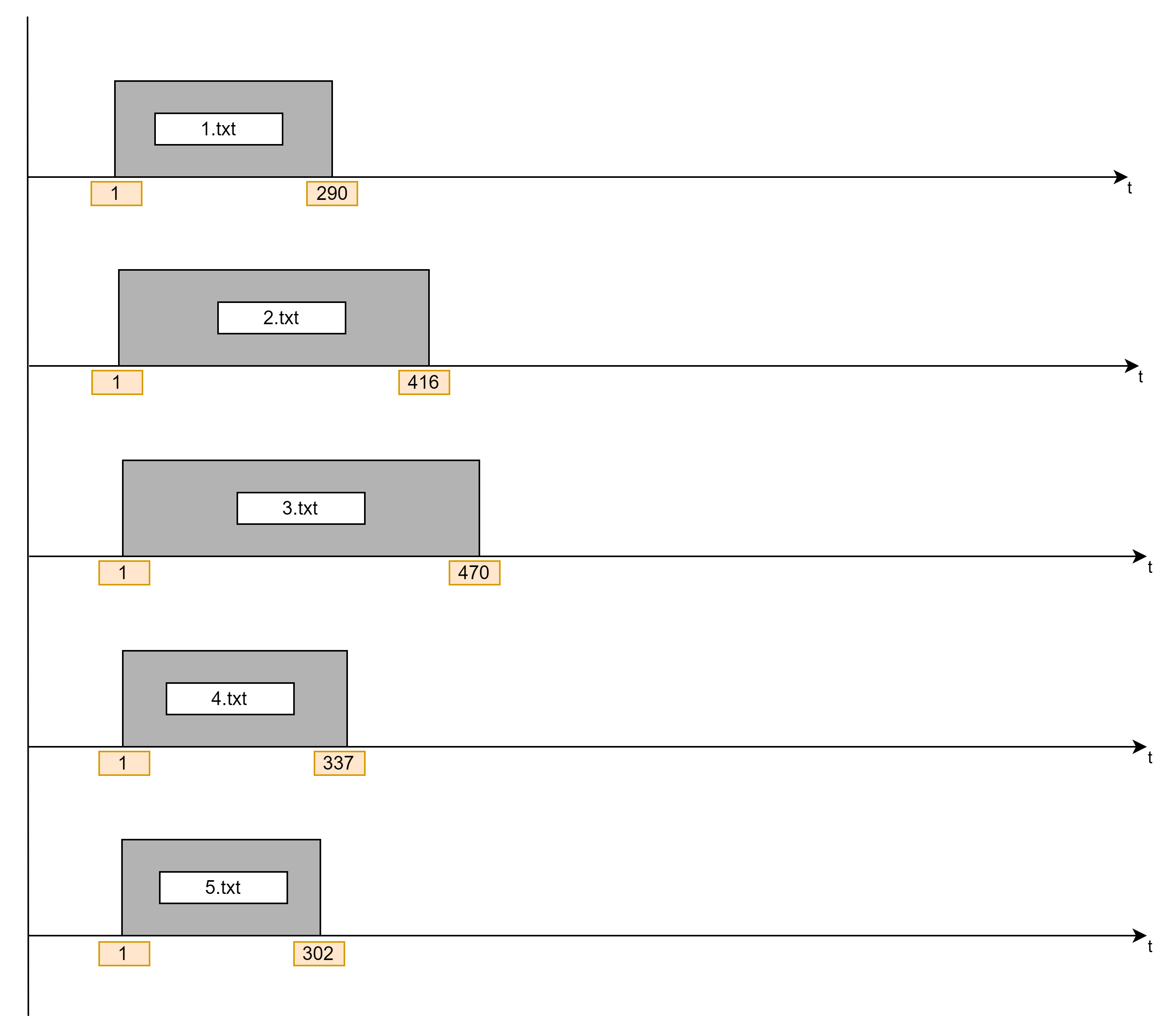


Рисунок 17 – Диаграмма трасс потоков программы BF с 5 текстовыми файлами

# ВЫВОДЫ ПО РЕЗУЛЬТАТАМ ПРОВЕДЕННГО ЭКСПЕРИМЕНТА

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

Многопоточный режим (B) показывает значительно более высокую производительность по сравнению с однопоточным режимом (A), особенно при увеличении числа файлов для обработки. Это связано с параллельной обработкой файлов, что позволяет эффективно использовать ресурсы системы.

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

На съемных носителях (F) многопоточный режим также превосходит однопоточный по времени обработки. Однако прирост производительности меньше по сравнению с жестким диском (H), что может быть связано с ограниченной пропускной способностью флеш-накопителя.

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

Диаграммы трасс потоков показывают, что многопоточность позволяет одновременно обрабатывать несколько файлов, распределяя задачи между потоками. В однопоточном режиме программа обрабатывает файлы последовательно, что увеличивает общее время обработки. Многопоточный режим, напротив, запускает несколько потоков, которые выполняются параллельно, что снижает время простоя процессора и повышает общую производительность системы. На диаграммах видно, что в многопоточном режиме процессор загружен более равномерно, что приводит к сокращению общего времени обработки файлов.

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

Повышение приоритета потоков в программе (режим C) приводит к тому, что операционная система выделяет больше процессорного времени этим потокам, особенно в условиях конкуренции за ресурсы. На практике это приводит к небольшому улучшению производительности по сравнению с многопоточным режимом с нормальным приоритетом (режим B). Повышение приоритета потоков полезно, когда необходимо обеспечить более быстрый отклик для критически важных задач, однако значительного увеличения скорости обработки не наблюдается из-за ограничений на уровне ввода/вывода файлов и ресурсов системы.

**Влияние количества ядер процессора на результаты**:

Количество ядер процессора оказывает непосредственное влияние на эффективность многопоточных приложений. Чем больше ядер в системе, тем больше потоков может выполняться одновременно, что позволяет обрабатывать файлы быстрее. В многопоточном режиме каждый поток может быть привязан к отдельному ядру, что сокращает время ожидания выполнения и уменьшает задержки, связанные с переключением контекста. На системах с небольшим количеством ядер многопоточные программы могут сталкиваться с конкуренцией за ресурсы, что снижает эффективность параллельной обработки.

**Необходимость использования средств синхронизации потоков**:

В многопоточных программах использование средств синхронизации, таких как критические секции, мьютексы или семафоры, крайне важно для корректной работы программы. Без синхронизации возможны ситуации, когда несколько потоков одновременно обращаются к разделяемым ресурсам (например, файлам или общим переменным), что может привести к некорректным результатам или повреждению данных. В проекте использовались критические секции для обеспечения потокобезопасного доступа к общим структурам данных, что позволило избежать состояний гонки и обеспечило правильное функционирование программы при одновременной работе нескольких потоков.

# 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.