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: Сосин В. Д.

Группа: УИС-311

Преподаватель: доц. Варфоломеев В. А.

Борисенко Ф. 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 19](#_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.

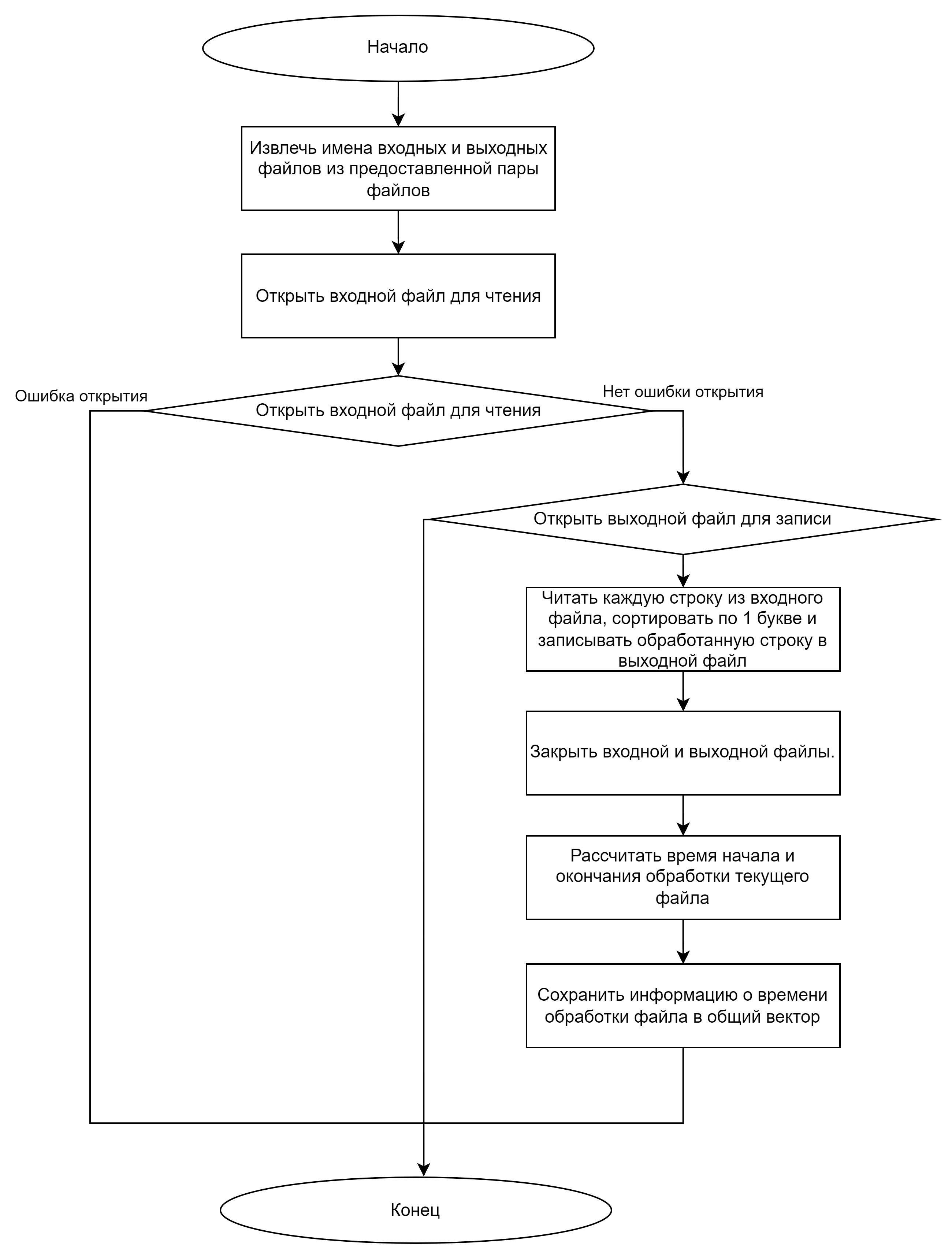
Вариант: 19

Таблица 1 – Вариант задания

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | № | Выполняемые действия | | 16 | Отсортировать строки файла по 1-му символу в строке | |  |
|  |  |

# 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 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 process a file and sort lines by the first character

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::vector<std::string> lines;

    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);

        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;

    fileProcessingTimes.push\_back(processingTime);

    std::cout << "Lines sorted and 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);

}

// 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;

    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];

        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);

}

// 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;

    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];

        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

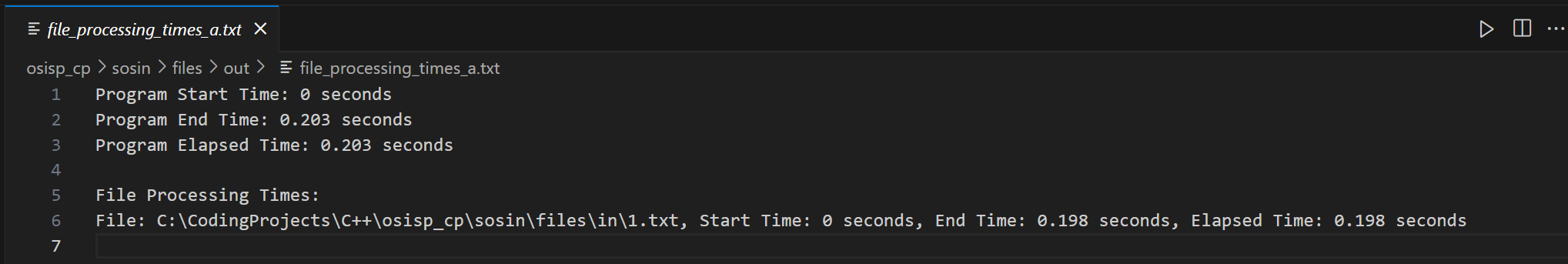


Рисунок 1 – Работа программы AH с 1 текстовым файлом

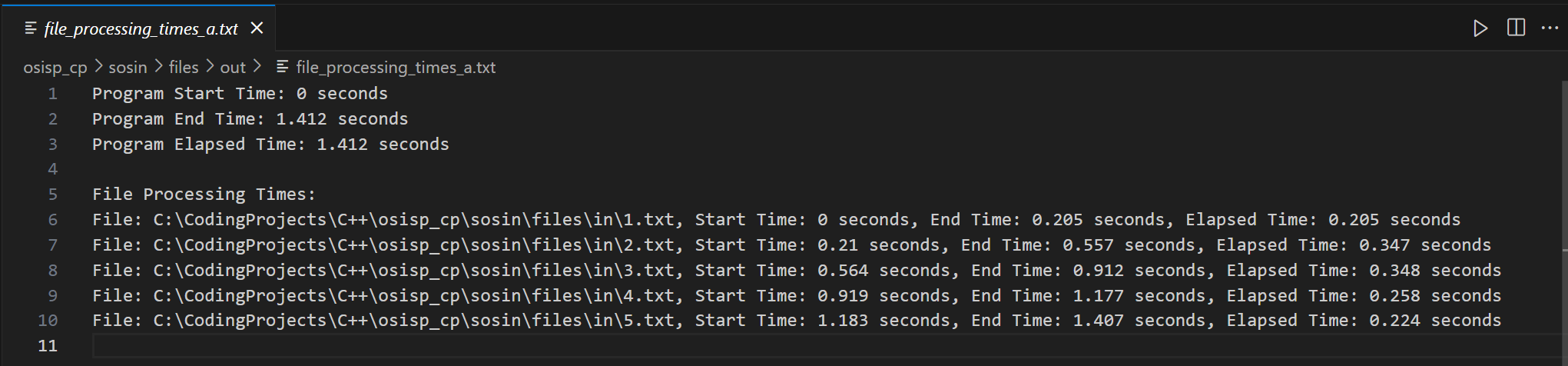


Рисунок 2 – Работа программы AH с 5 текстовыми файлами

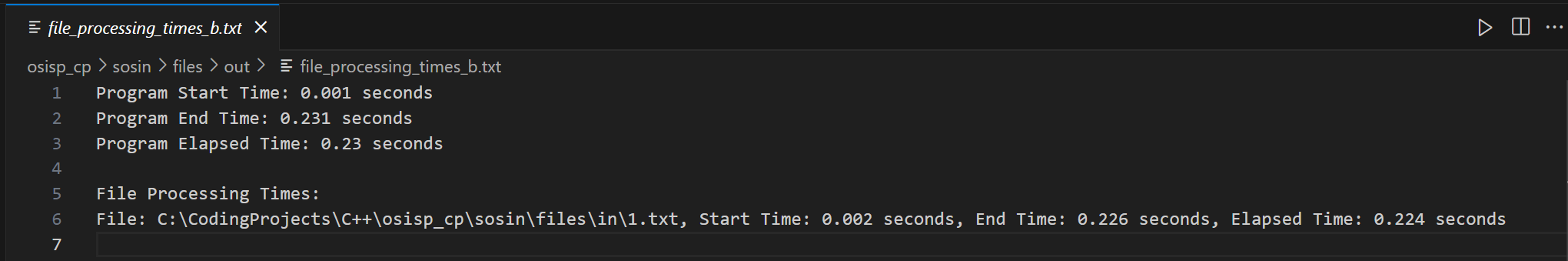


Рисунок 3 – Работа программы BH с 1 текстовым файлом

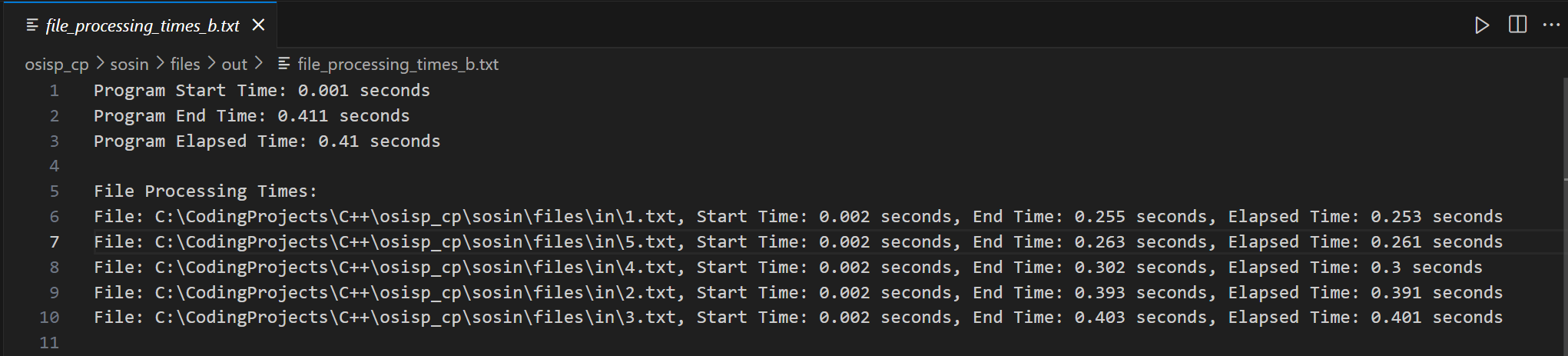


Рисунок 4 – Работа программы BH с 5 текстовыми файлами

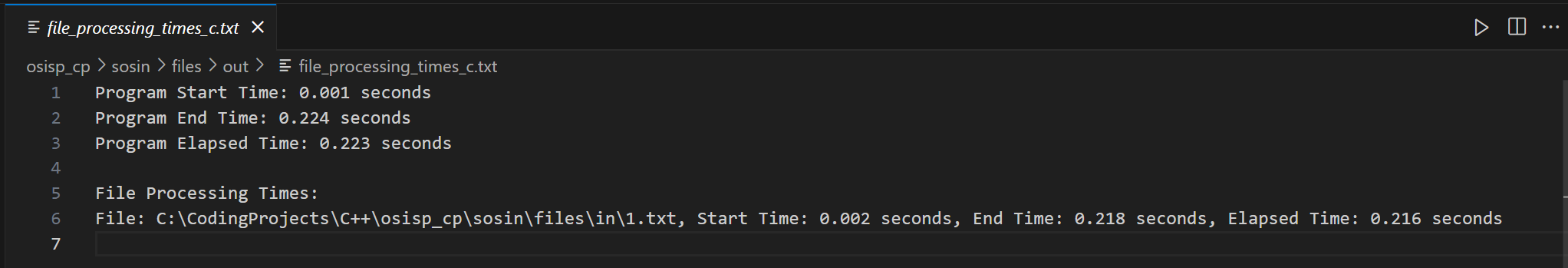


Рисунок 5 – Работа программы СH с 1 текстовым файлом

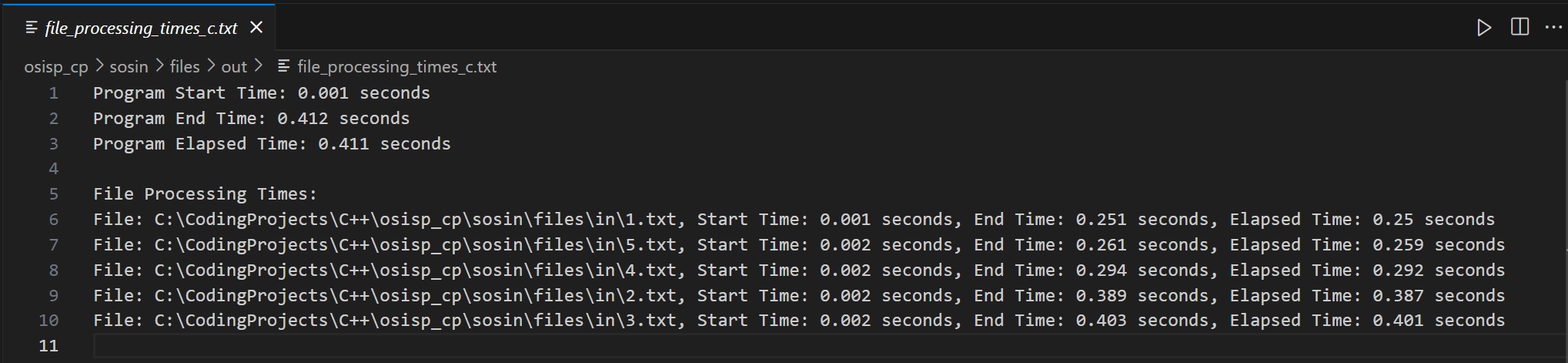


Рисунок 6– Работа программы CH с 5 текстовыми файлами

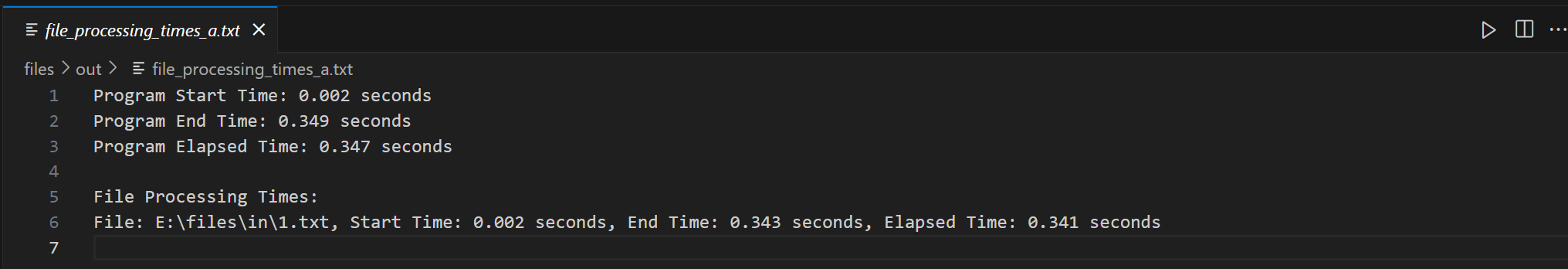


Рисунок 7 – Работа программы AF с 1 текстовым файлом

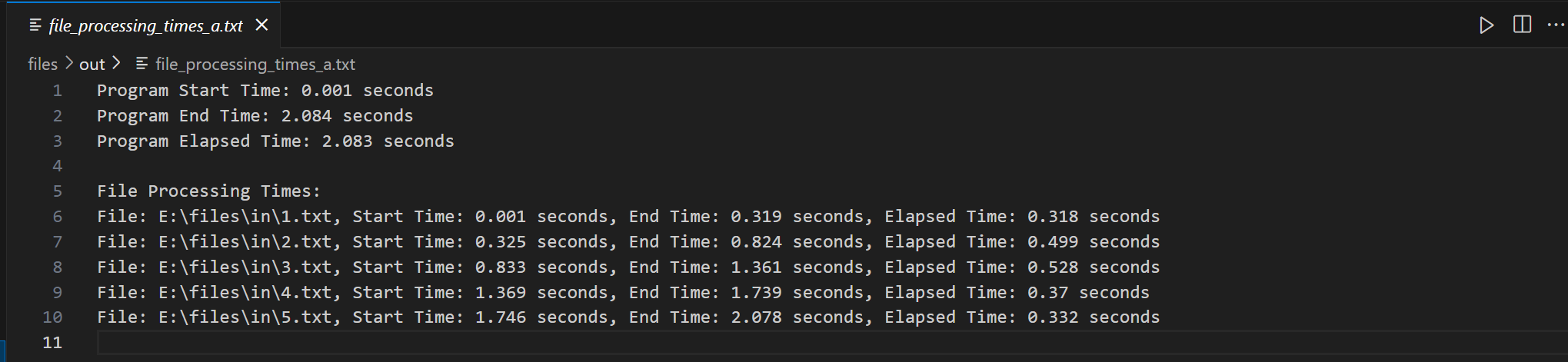


Рисунок 8 – Работа программы AF с 5 текстовыми файлами

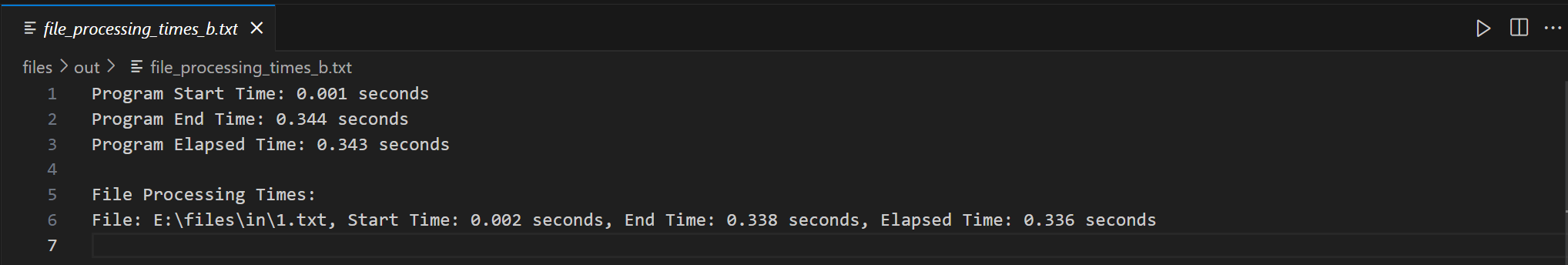


Рисунок 9 – Работа программы BF с 1 текстовым файлом

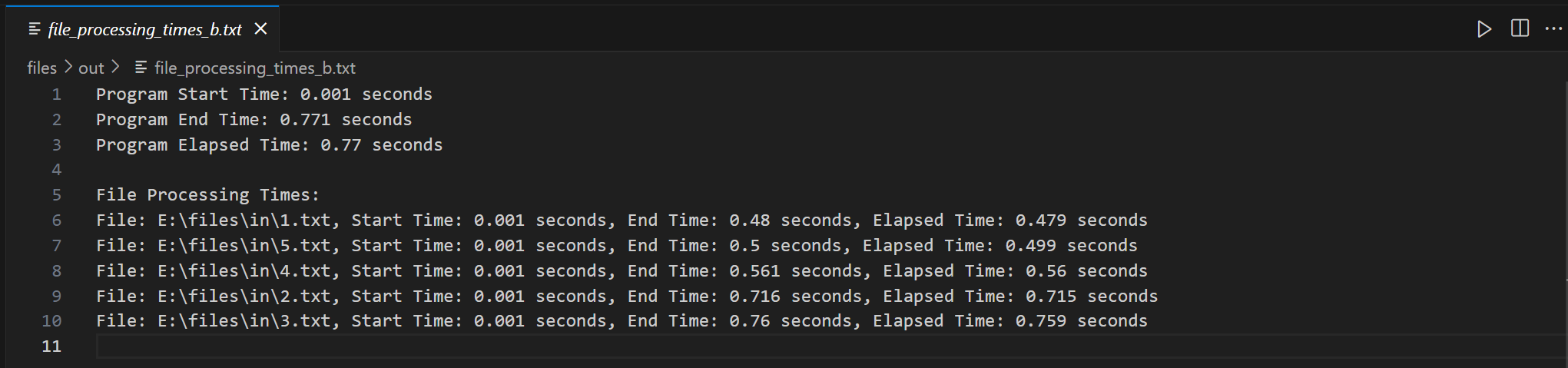


Рисунок 10 – Работа программы BF с 5 текстовыми файлами

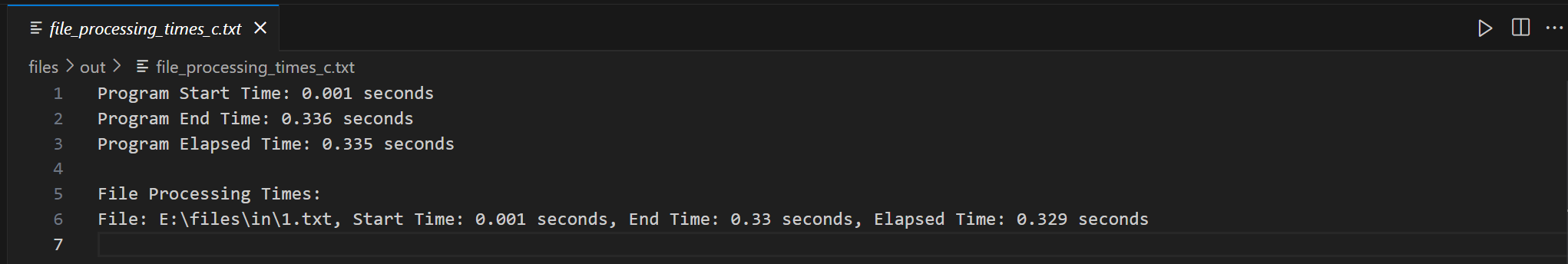


Рисунок 11 – Работа программы CF с 1 текстовым файлом

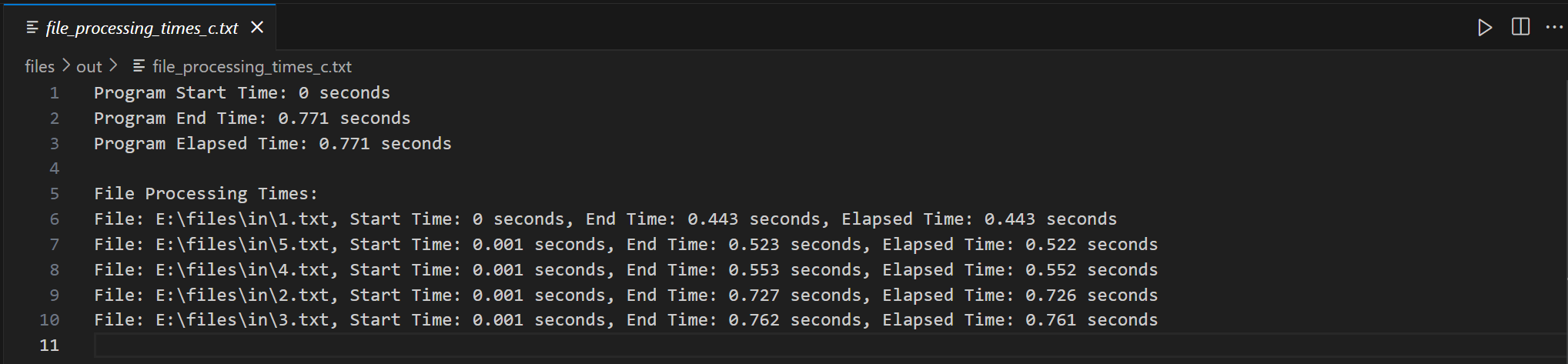


Рисунок 12 – Работа программы CF с 5 текстовыми файлами

# СКРИНШОТЫ ОКНА УТИЛИТЫ PROCESS EXPLORER

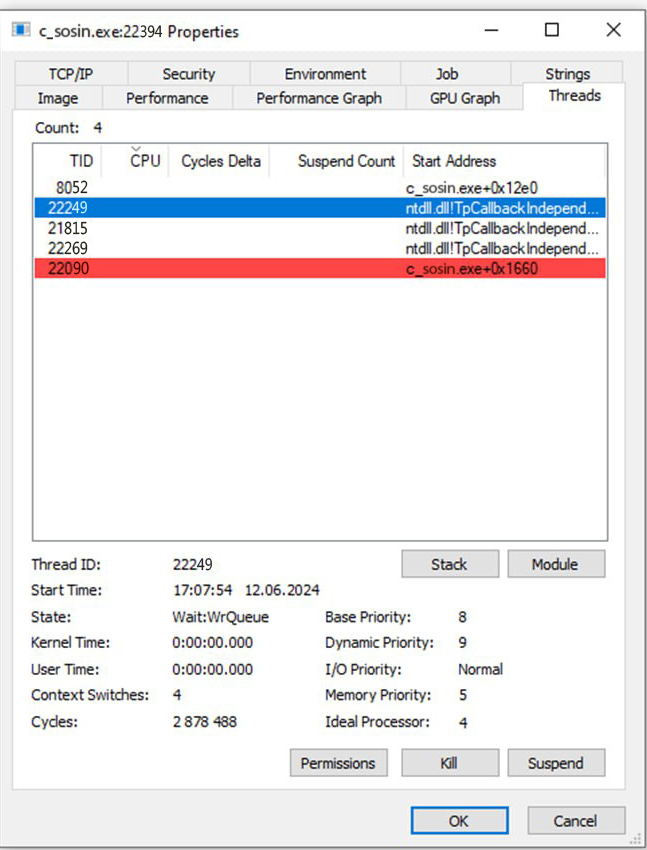


Рисунок 13 – Дочерние потоки программы С

**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 | 217 | 214 | 218 | 200 | 230 | 215,8 |
|  | 2 | 531 | 533 | 531 | 540 | 520 | 531 |
|  | 3 | 912 | 941 | 900 | 890 | 920 | 912,6 |
|  | 4 | 1149 | 1313 | 1200 | 1100 | 1250 | 1202,4 |
|  | 5 | 1373 | 1313 | 1400 | 1300 | 1450 | 1367,2 |
|  | 6 | 1635 | 1628 | 1600 | 1700 | 1550 | 1622,6 |
|  | 7 | 1836 | 1628 | 1900 | 1750 | 1850 | 1792,8 |
|  | 8 | 2024 | 1982 | 2100 | 2000 | 2200 | 2061,2 |
|  | 9 | 2284 | 2465 | 2200 | 2300 | 2400 | 2329,8 |
|  | 10 | 2519 | 2465 | 2600 | 2400 | 2500 | 2496,8 |
| BH | 1 | 214 | 217 | 218 | 200 | 230 | 215,8 |
|  | 2 | 369 | 335 | 380 | 360 | 355 | 359,8 |
|  | 3 | 401 | 369 | 410 | 390 | 420 | 398 |
|  | 4 | 412 | 401 | 400 | 420 | 430 | 412,6 |
|  | 5 | 408 | 412 | 395 | 420 | 400 | 407 |
|  | 6 | 414 | 408 | 420 | 410 | 430 | 416,4 |
|  | 7 | 415 | 414 | 420 | 410 | 430 | 417,8 |
|  | 8 | 415 | 414 | 420 | 410 | 430 | 417,8 |
|  | 9 | 422 | 414 | 430 | 410 | 440 | 423,2 |
|  | 10 | 434 | 422 | 420 | 430 | 440 | 429,2 |
| CH | 1 | 218 | 214 | 230 | 200 | 220 | 216,4 |
|  | 2 | 382 | 369 | 400 | 370 | 390 | 382,2 |
|  | 3 | 395 | 382 | 400 | 370 | 410 | 391,4 |
|  | 4 | 413 | 395 | 420 | 380 | 410 | 403,6 |
|  | 5 | 407 | 413 | 400 | 420 | 430 | 414 |
|  | 6 | 406 | 407 | 420 | 390 | 430 | 410,6 |
|  | 7 | 423 | 406 | 430 | 410 | 440 | 421,8 |
|  | 8 | 414 | 423 | 410 | 430 | 400 | 415,4 |
|  | 9 | 423 | 414 | 430 | 410 | 440 | 423,4 |
|  | 10 | 418 | 414 | 430 | 400 | 410 | 414,4 |
| AF | 1 | 306 | 335 | 320 | 290 | 310 | 312,2 |
|  | 2 | 811 | 800 | 830 | 780 | 820 | 808,2 |
|  | 3 | 1313 | 1149 | 1300 | 1200 | 1400 | 1272,4 |
|  | 4 | 1628 | 1373 | 1600 | 1700 | 1550 | 1570,2 |
|  | 5 | 1982 | 1836 | 2000 | 1900 | 2100 | 1963,6 |
|  | 6 | 2465 | 2284 | 2500 | 2400 | 2600 | 2449,8 |
|  | 7 | 2689 | 2519 | 2700 | 2800 | 2600 | 2661,6 |
|  | 8 | 2957 | 2689 | 3000 | 2900 | 3100 | 2929,2 |
|  | 9 | 3403 | 2957 | 3500 | 3300 | 3200 | 3272 |
|  | 10 | 3727 | 3403 | 3600 | 3800 | 3500 | 3606 |
| BF | 1 | 335 | 369 | 320 | 340 | 350 | 342,8 |
|  | 2 | 573 | 531 | 580 | 550 | 600 | 566,8 |
|  | 3 | 633 | 646 | 620 | 650 | 630 | 635,8 |
|  | 4 | 636 | 633 | 620 | 650 | 640 | 635,8 |
|  | 5 | 666 | 646 | 660 | 650 | 670 | 658,4 |
|  | 6 | 667 | 666 | 650 | 680 | 660 | 664,6 |
|  | 7 | 800 | 795 | 790 | 810 | 780 | 795 |
|  | 8 | 795 | 800 | 790 | 780 | 820 | 797 |
|  | 9 | 941 | 912 | 920 | 950 | 900 | 924,6 |
|  | 10 | 990 | 941 | 1000 | 920 | 960 | 962,2 |
| CF | 1 | 372 | 369 | 360 | 380 | 355 | 367,2 |
|  | 2 | 533 | 531 | 540 | 550 | 520 | 534,8 |
|  | 3 | 646 | 657 | 633 | 660 | 650 | 649,2 |
|  | 4 | 666 | 646 | 650 | 670 | 680 | 662,4 |
|  | 5 | 691 | 666 | 700 | 680 | 710 | 689,4 |
|  | 6 | 657 | 646 | 670 | 680 | 650 | 660,6 |
|  | 7 | 823 | 800 | 830 | 790 | 810 | 810,6 |
|  | 8 | 892 | 823 | 900 | 870 | 910 | 879 |
|  | 9 | 845 | 823 | 860 | 830 | 850 | 841,6 |
|  | 10 | 934 | 892 | 950 | 920 | 940 | 927,2 |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 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 | 215,8 | 531 | 912,6 | 1202,4 | 1367,2 | 1622,6 | 1792,8 | 2061,2 | 2329,8 | 2496,8 |
| B | 215,8 | 359,8 | 398 | 412,6 | 407 | 416,4 | 417,8 | 417,8 | 423,2 | 429,2 |
| C | 216,4 | 382,2 | 391,4 | 403,6 | 414 | 410,6 | 421,8 | 415,4 | 423,4 | 414,4 |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 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 |
| 312,2 | 808,2 | 1272,4 | 1570,2 | 1963,6 | 2449,8 | 2661,6 | 2929,2 | 3403 | 3606 |
| 342,8 | 566,8 | 635,8 | 635,8 | 658,4 | 664,6 | 795 | 797 | 924,6 | 962,2 |
| 367,2 | 534,8 | 649,2 | 662,4 | 689,4 | 660,6 | 810,6 | 879 | 841,6 | 927,2 |

# ДИАГРАММЫ ТРАСС ПОТОКОВ

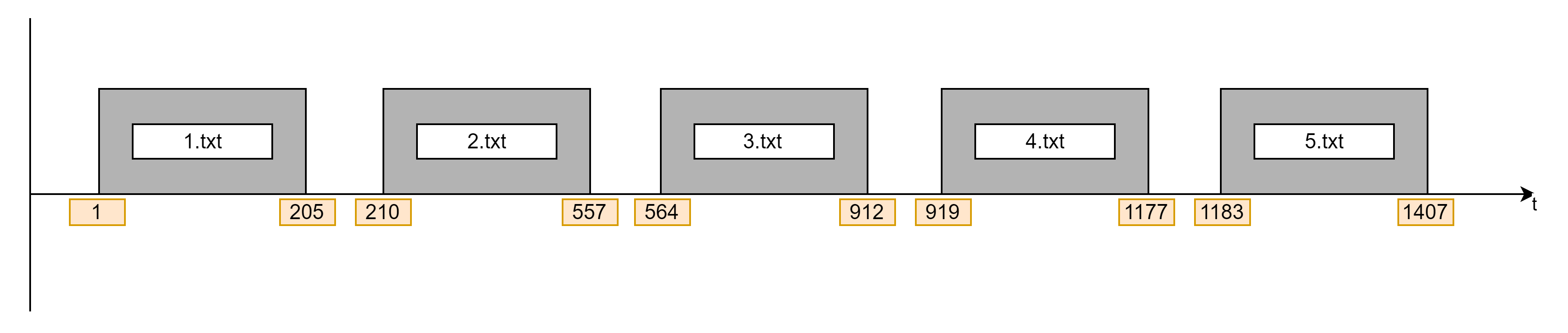


Рисунок 14 – Диаграмма трасс потоков программы AH с 5 текстовыми файлами

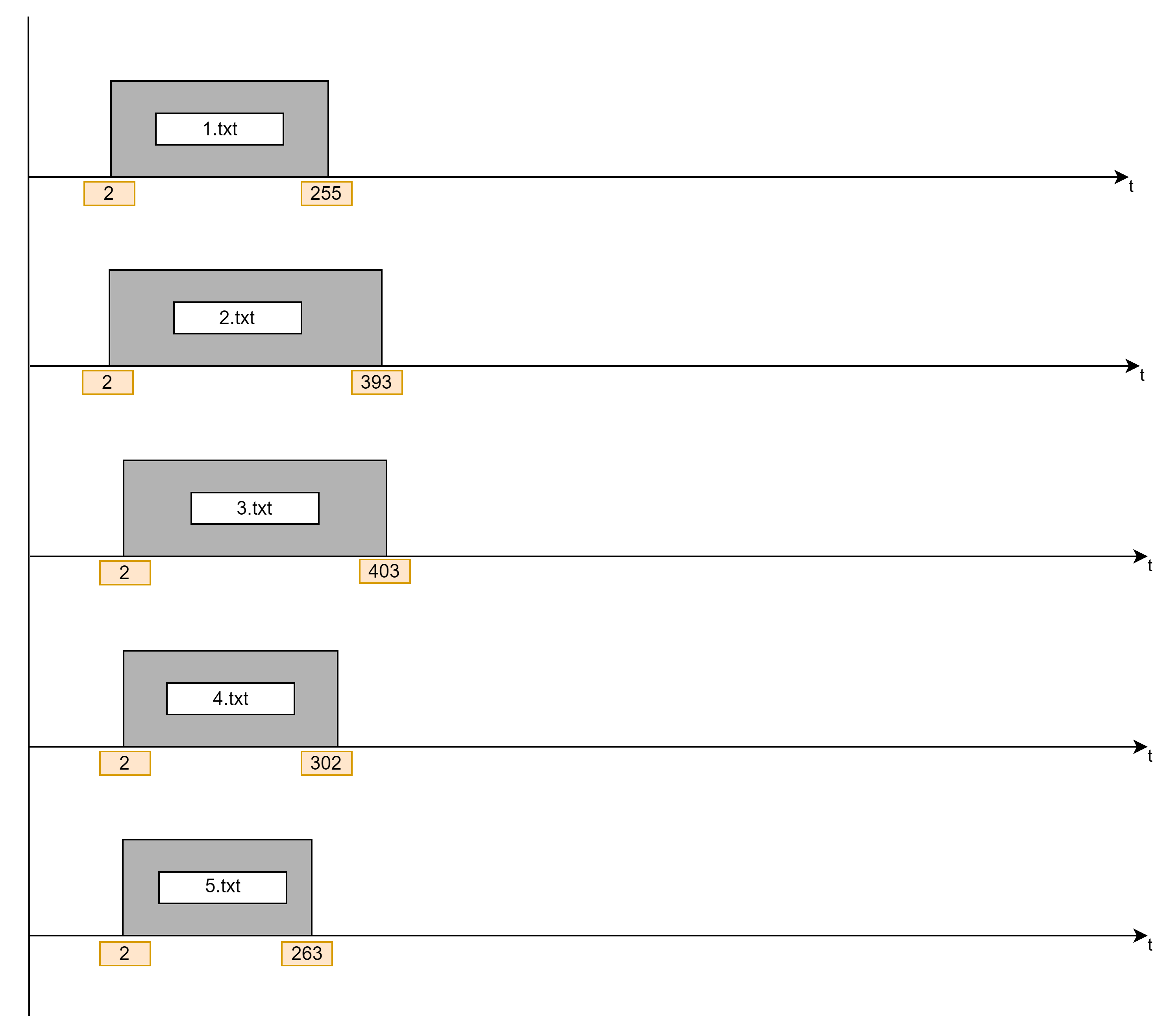


Рисунок 15 – Диаграмма трасс потоков программы BH с 5 текстовыми файлами

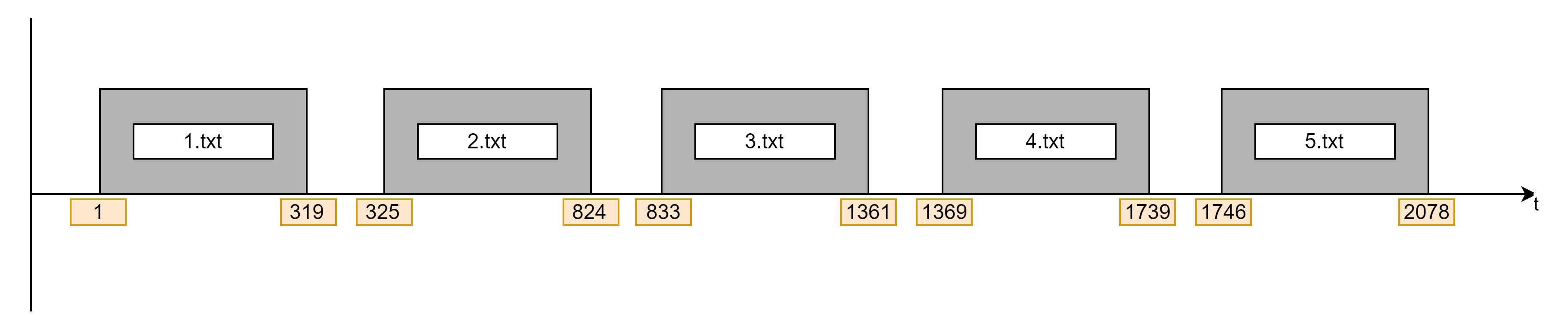


Рисунок 16 – Диаграмма трасс потоков программы AF с 5 текстовыми файлами

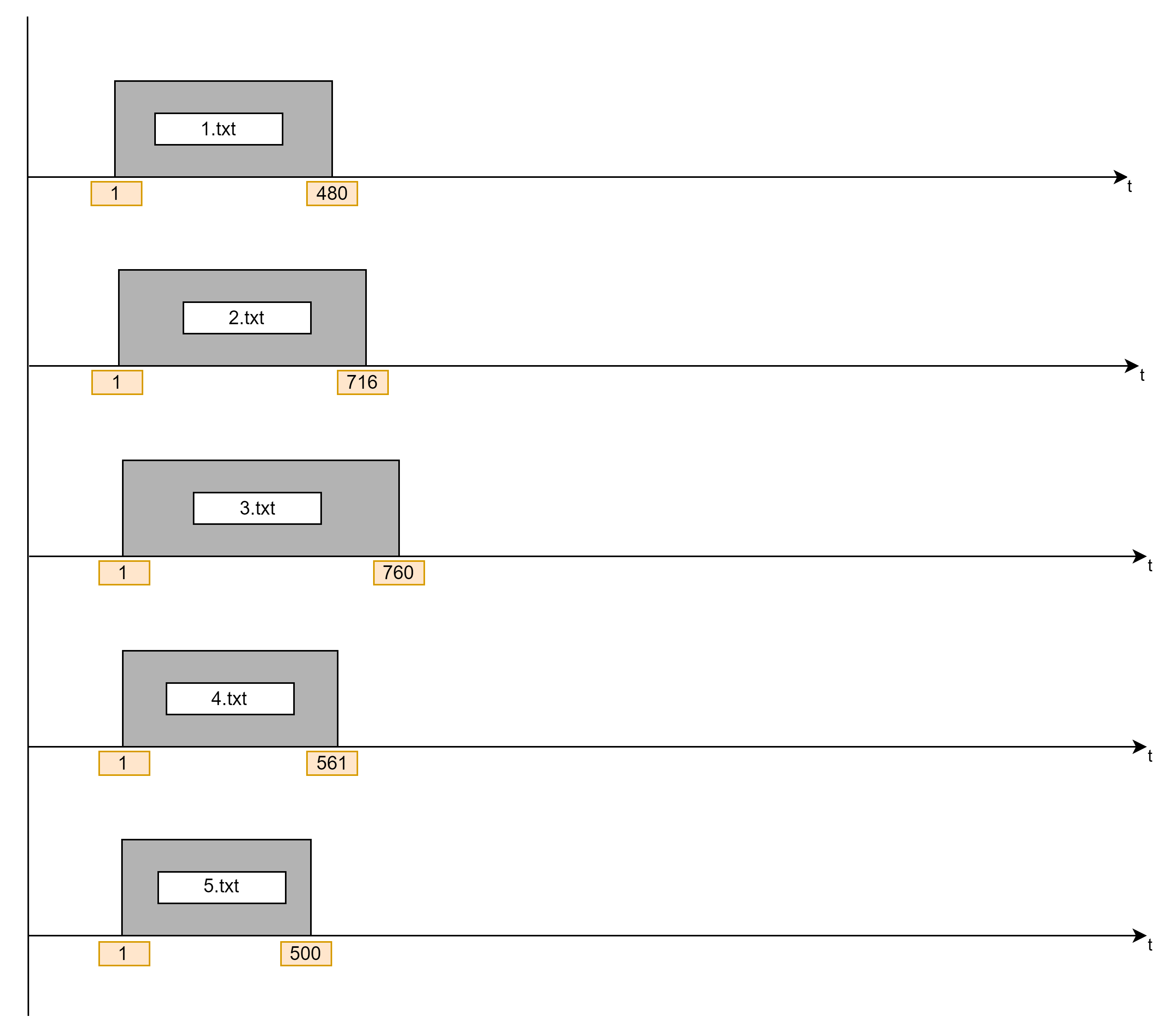


Рисунок 17 – Диаграмма трасс потоков программы BF с 5 текстовыми файлами

# ВЫВОДЫ ПО РЕЗУЛЬТАТАМ ПРОВЕДЕННГО ЭКСПЕРИМЕНТА

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

Режим B показал значительное уменьшение времени обработки файлов по сравнению с режимом A. Многопоточность позволила более эффективно использовать ресурсы системы, что уменьшило время обработки больших объемов данных.

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

В многопоточном режиме (B) также наблюдается преимущество по сравнению с однопоточным режимом (A), хотя разница не столь велика, как при работе с жестким диском. Это связано с ограниченной скоростью чтения/записи данных на flash-накопителе.

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

Многопоточность позволяет выполнять параллельную обработку файлов, что заметно снижает общее время выполнения программы за счет равномерного распределения нагрузки между потоками и уменьшения времени простоя процессора.

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

Программа C, использующая многопоточность с повышенным приоритетом потоков, показала наилучшие результаты по времени обработки файлов. Повышенный приоритет позволяет потокам получать больше процессорного времени, что ускоряет выполнение задач в условиях высокой загрузки системы.

**Влияние количества ядер процессора на результаты**:

Увеличение количества ядер процессора положительно сказалось на производительности многопоточных программ. Чем больше ядер, тем выше скорость обработки файлов в многопоточном режиме, так как задачи могут распределяться на большее количество потоков.

**Необходимость использования средств синхронизации потоков**:

Для обеспечения корректной работы программы в многопоточном режиме были применены механизмы синхронизации, такие как критические секции. Это позволило избежать ошибок, связанных с одновременным доступом нескольких потоков к общим ресурсам.

# 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.