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: Кривоногов П. А.

Группа: УИС-311

Преподаватель: доц. Варфоломеев В. А.

­

Москва 2024 г.

**СОДЕРЖАНИЕ**

[ВАРИАНТ ЗАДАНИЯ 3](#_Toc176644745)

[БЛОК-СХЕМА АЛГОРИТМА 4](#_Toc176644746)

[ТЕКСТ ПРОГРАММ 6](#_Toc176644747)

[Программа A 6](#_Toc176644748)

[Программа B 13](#_Toc176644749)

[Программа C 21](#_Toc176644750)

[РЕЗУЛЬТАТ РАБОТЫ ПРОГРАММ 30](#_Toc176644751)

[СКРИНШОТЫ ОКНА УТИЛИТЫ PROCESS EXPLORER 33](#_Toc176644752)

[ОПИСАНИЕ И РЕЗУЛЬТАТЫ ПРОВЕДЕННОГО ЭКСПЕРИМЕНТА 35](#_Toc176644753)

[ДИАГРАММЫ ТРАСС ПОТОКОВ 40](#_Toc176644754)

[ВЫВОДЫ ПО РЕЗУЛЬТАТАМ ПРОВЕДЕННГО ЭКСПЕРИМЕНТА 42](#_Toc176644755)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 44](#_Toc176644756)

# 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.

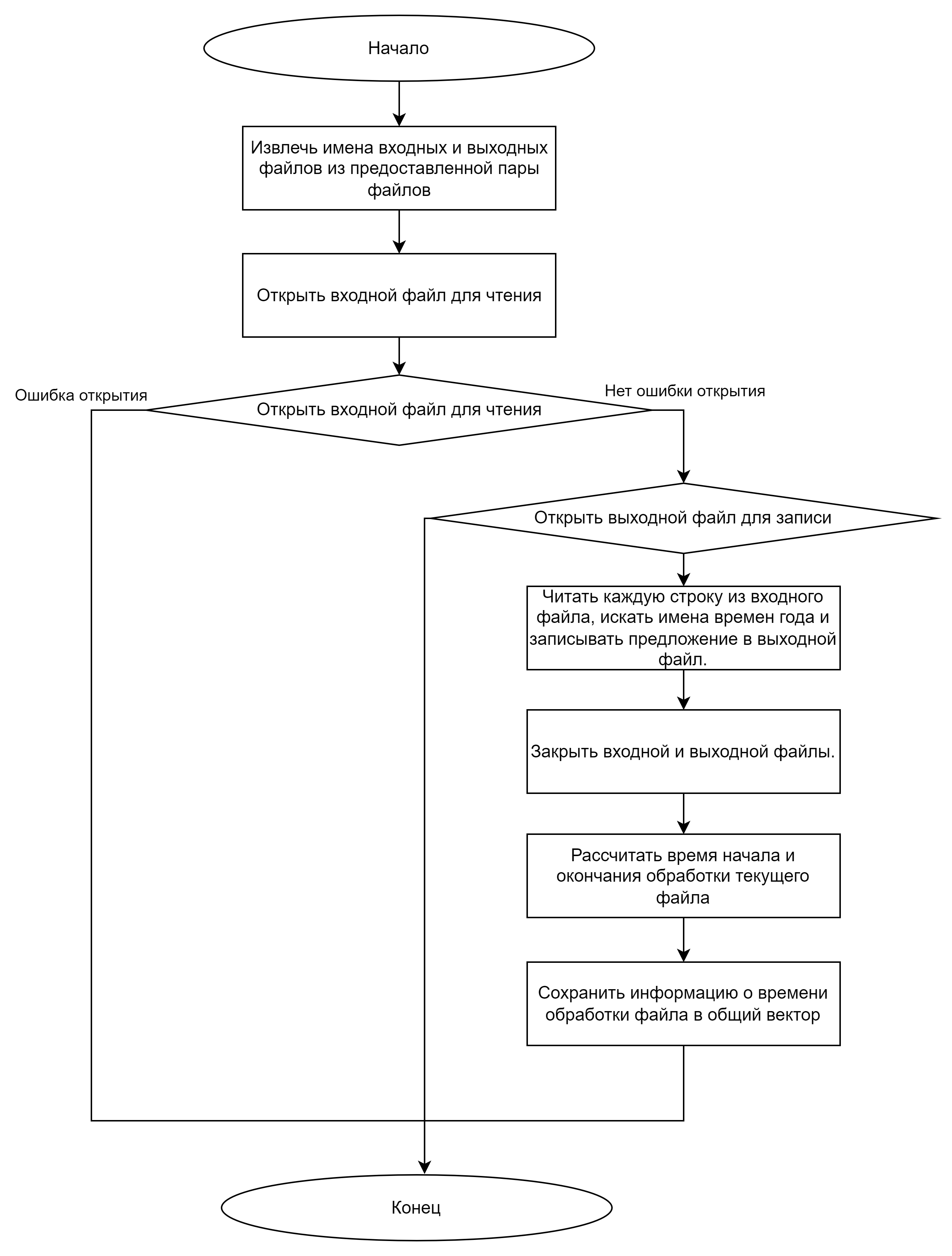
Вариант: 9

Таблица 1 – Вариант задания

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | № | Выполняемые действия | | 9 | Найти и вывести предложения, содержащие названия времен года | |  |
|  |  |

# 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\\";

// List of season names and their cases

const std::unordered\_set<std::string> seasonNames = {

    "winter", "spring", "summer", "autumn", "fall"

};

std::string cleanAndLowercase(const std::string& word) {

    std::string cleaned;

    std::remove\_copy\_if(word.begin(), word.end(), std::back\_inserter(cleaned), [](char c) { return std::ispunct(static\_cast<unsigned char>(c)); });

    std::transform(cleaned.begin(), cleaned.end(), cleaned.begin(), ::tolower);

    return cleaned;

}

// Function to check if a line contains any season names

bool containsSeasonName(const std::string &line) {

    std::istringstream iss(line);

    std::string word;

    while (iss >> word) {

        std::string cleanedWord = cleanAndLowercase(word);

        if (seasonNames.find(cleanedWord) != seasonNames.end()) {

            return true;

        }

    }

    return false;

}

// Function to process a file and find sentences containing season names

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> sentencesWithSeasonNames;

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

        if (containsSeasonName(line)) {

            sentencesWithSeasonNames.push\_back(line);

        }

    }

    outFile << "Sentences containing the names of seasons:\n";

    for (const auto &sentence : sentencesWithSeasonNames) {

        outFile << sentence << '\n';

    }

    inFile.close();

    outFile.close();

    endFileTime = clock();

    processingTime.start\_time = startFileTime;

    processingTime.end\_time = endFileTime;

    fileProcessingTimes.push\_back(processingTime);

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

}

// 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\\"; // In

        outputDirectory = "E:\\files\\out\\"; // 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\\";

// List of season names and their cases

const std::unordered\_set<std::string> seasonNames = {

    "winter", "spring", "summer", "autumn", "fall"

};

std::string cleanAndLowercase(const std::string& word) {

    std::string cleaned;

    std::remove\_copy\_if(word.begin(), word.end(), std::back\_inserter(cleaned), [](char c) { return std::ispunct(static\_cast<unsigned char>(c)); });

    std::transform(cleaned.begin(), cleaned.end(), cleaned.begin(), ::tolower);

    return cleaned;

}

// Function to check if a line contains any season names

bool containsSeasonName(const std::string &line) {

    std::istringstream iss(line);

    std::string word;

    while (iss >> word) {

        std::string cleanedWord = cleanAndLowercase(word);

        if (seasonNames.find(cleanedWord) != seasonNames.end()) {

            return true;

        }

    }

    return false;

}

// Thread function to process a file and find sentences containing season names

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> sentencesWithSeasonNames;

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

        if (containsSeasonName(line)) {

            sentencesWithSeasonNames.push\_back(line);

        }

    }

    outFile << "Sentences containing the names of seasons:\n";

    for (const auto &sentence : sentencesWithSeasonNames) {

        outFile << sentence << '\n';

    }

    inFile.close();

    outFile.close();

    endFileTime = clock();

    processingTime.start\_time = startFileTime;

    processingTime.end\_time = endFileTime;

    EnterCriticalSection(&cs);

    fileProcessingTimes.push\_back(processingTime);

    LeaveCriticalSection(&cs);

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

    return 0;

}

// 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\\"; // In

        outputDirectory = "E:\\files\\out\\"; // 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\\";

// List of season names and their cases

const std::unordered\_set<std::string> seasonNames = {

    "winter", "spring", "summer", "autumn", "fall"

};

// Function to clean and lowercase words

std::string cleanAndLowercase(const std::string& word) {

    std::string cleaned;

    std::remove\_copy\_if(word.begin(), word.end(), std::back\_inserter(cleaned), [](char c) { return std::ispunct(static\_cast<unsigned char>(c)); });

    std::transform(cleaned.begin(), cleaned.end(), cleaned.begin(), ::tolower);

    return cleaned;

}

// Function to check if a line contains any season names

bool containsSeasonName(const std::string &line) {

    std::istringstream iss(line);

    std::string word;

    while (iss >> word) {

        std::string cleanedWord = cleanAndLowercase(word);

        if (seasonNames.find(cleanedWord) != seasonNames.end()) {

            return true;

        }

    }

    return false;

}

// Thread function to process a file and find sentences containing season names

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> sentencesWithSeasonNames;

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

        if (containsSeasonName(line)) {

            sentencesWithSeasonNames.push\_back(line);

        }

    }

    outFile << "Sentences containing the names of seasons:\n";

    for (const auto &sentence : sentencesWithSeasonNames) {

        outFile << sentence << '\n';

    }

    inFile.close();

    outFile.close();

    endFileTime = clock();

    processingTime.start\_time = startFileTime;

    processingTime.end\_time = endFileTime;

    EnterCriticalSection(&cs);

    fileProcessingTimes.push\_back(processingTime);

    LeaveCriticalSection(&cs);

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

    return 0;

}

// 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\\"; // In

        outputDirectory = "E:\\files\\out\\"; // 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

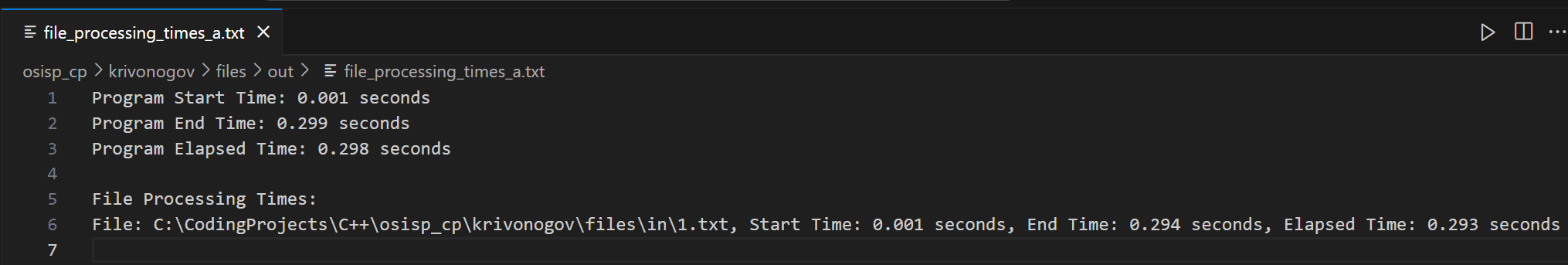


Рисунок 1 – Работа программы AH с 1 текстовым файлом

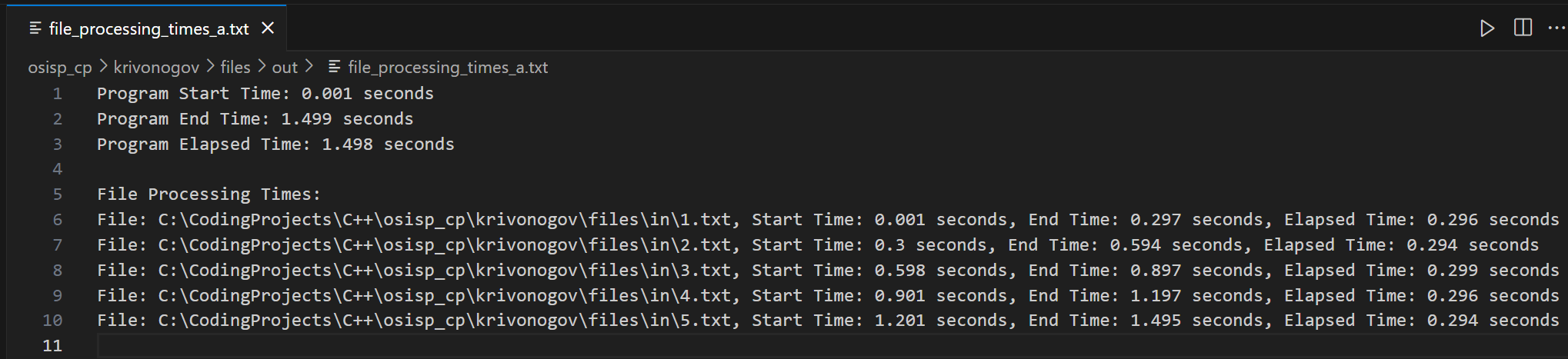


Рисунок 2 – Работа программы AH с 5 текстовыми файлами

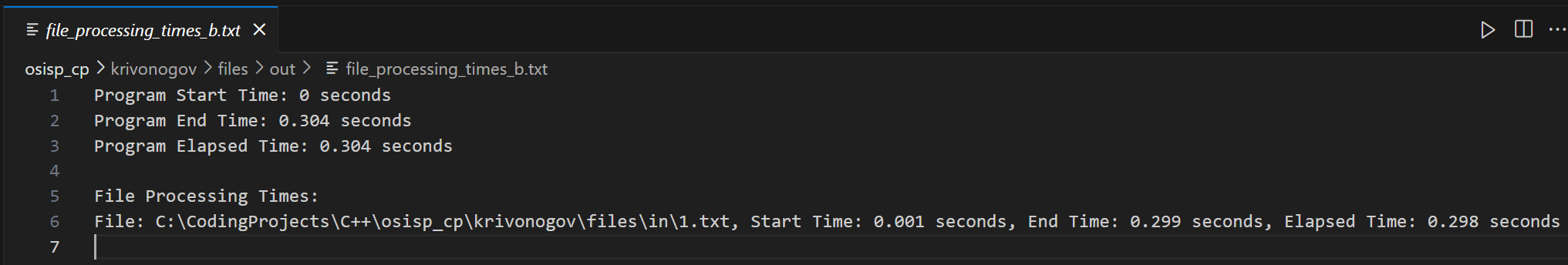


Рисунок 3 – Работа программы BH с 1 текстовым файлом

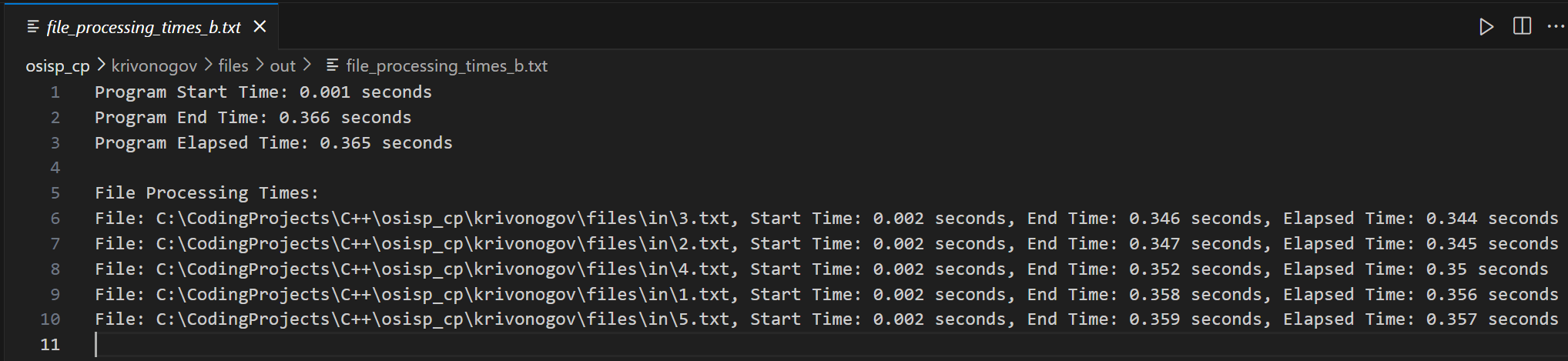


Рисунок 4 – Работа программы BH с 5 текстовыми файлами

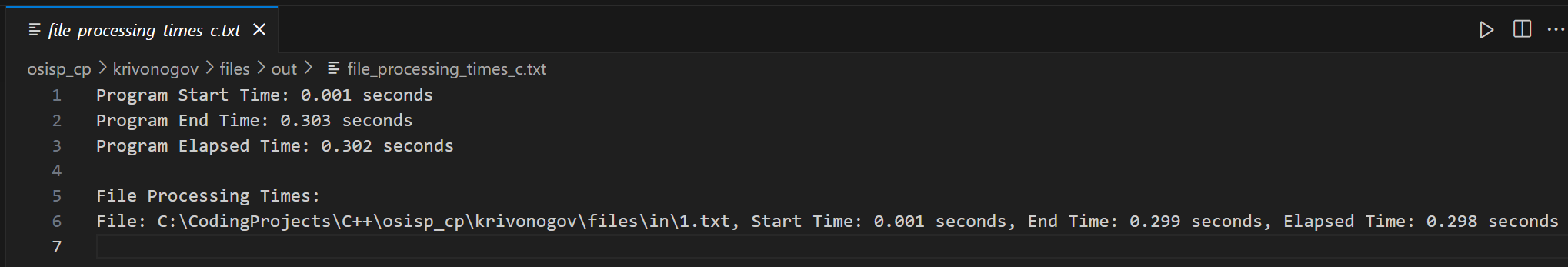


Рисунок 5 – Работа программы СH с 1 текстовым файлом

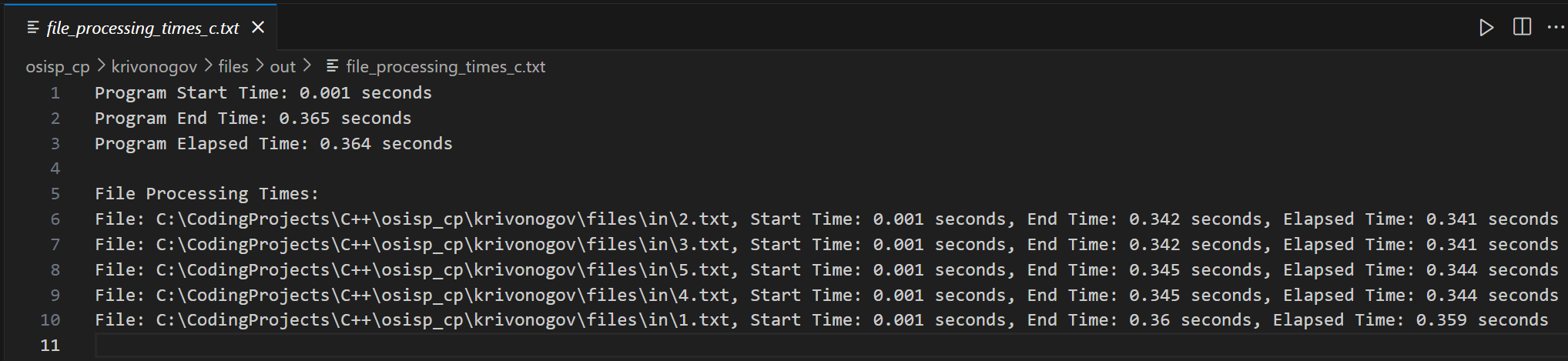


Рисунок 6– Работа программы CH с 5 текстовыми файлами

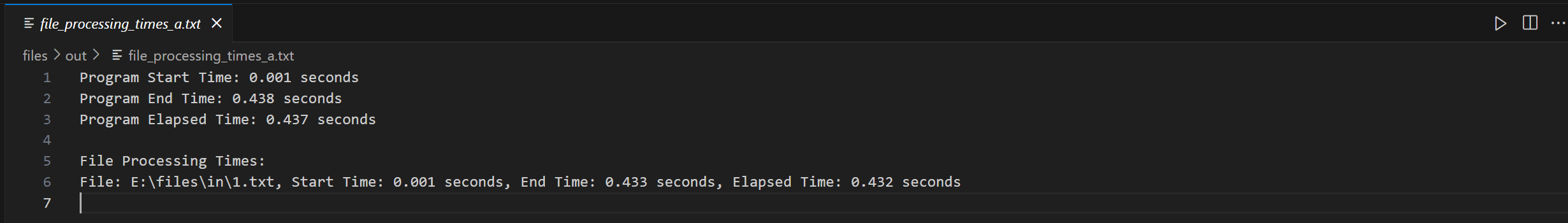


Рисунок 7 – Работа программы AF с 1 текстовым файлом

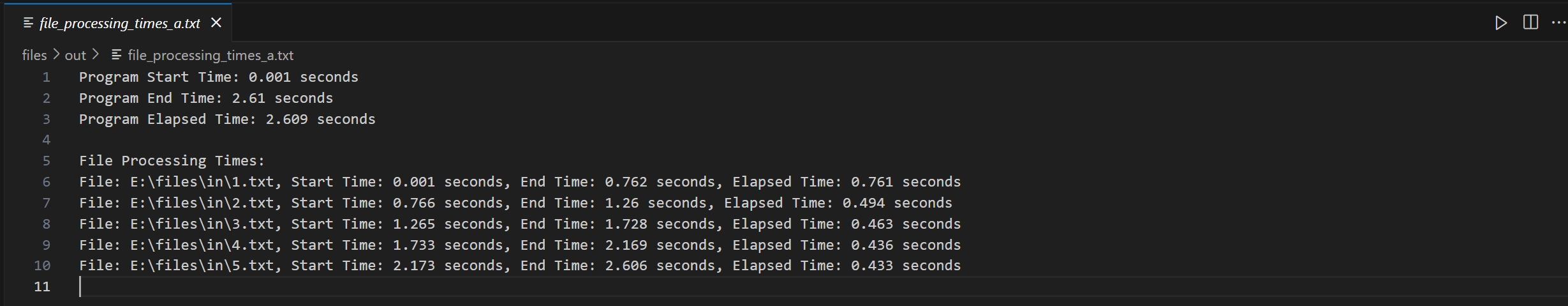


Рисунок 8 – Работа программы AF с 5 текстовыми файлами

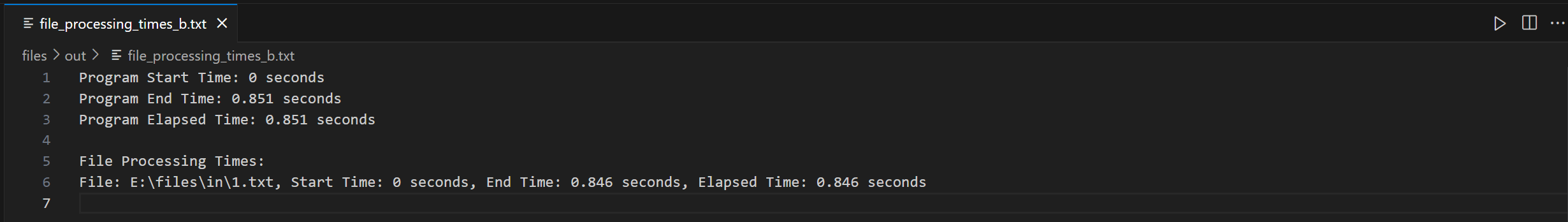


Рисунок 9 – Работа программы BF с 1 текстовым файлом

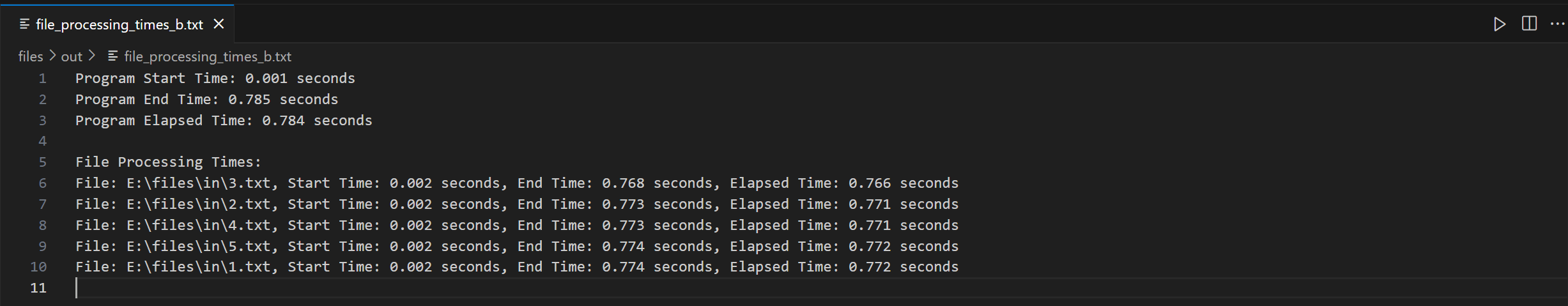


Рисунок 10 – Работа программы BF с 5 текстовыми файлами

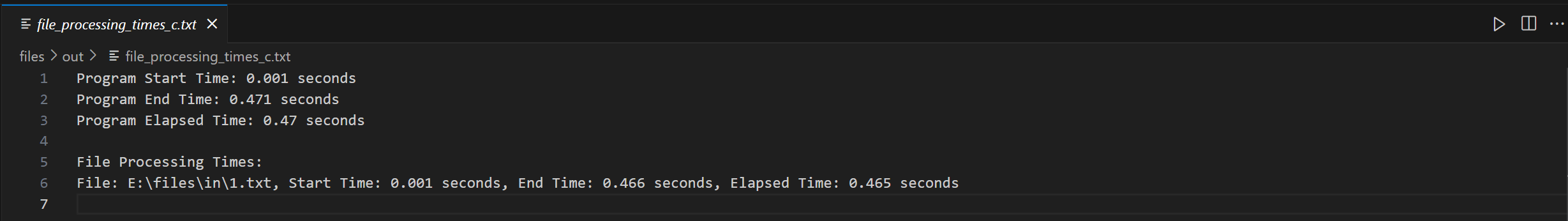


Рисунок 11 – Работа программы CF с 1 текстовым файлом

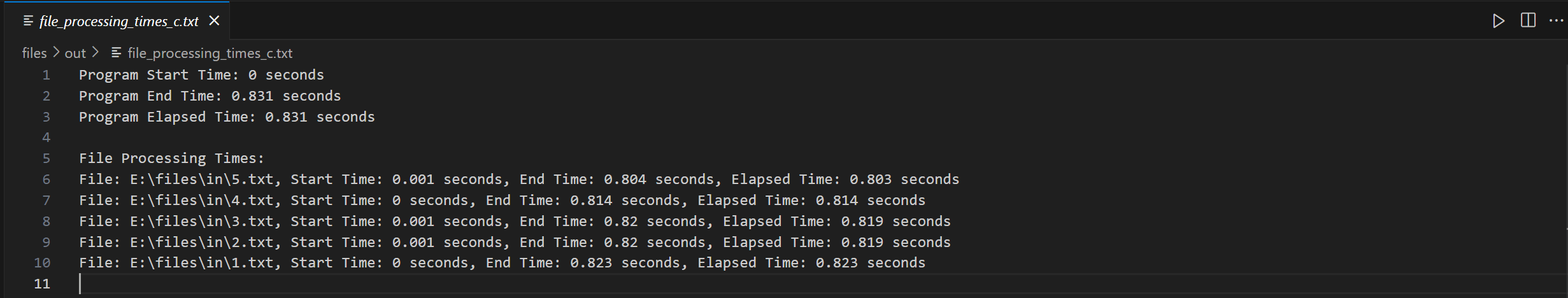


Рисунок 12 – Работа программы CF с 5 текстовыми файлами

# СКРИНШОТЫ ОКНА УТИЛИТЫ PROCESS EXPLORER

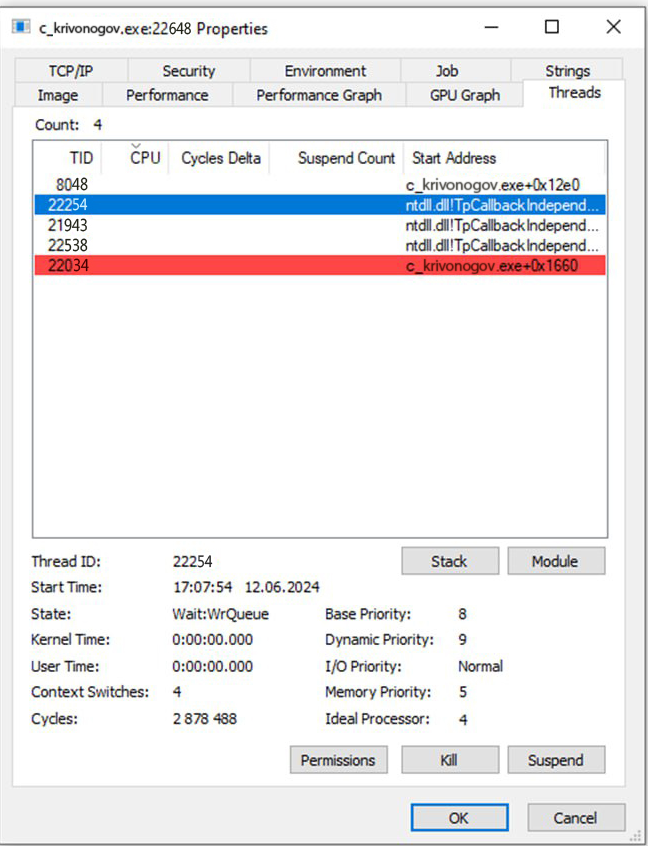


Рисунок 13 – Дочерние потоки программы С

**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 | 287 | 326 | 338 | 316 | 315 | 316,4 |
|  | 2 | 607 | 626 | 620 | 634 | 673 | 632 |
|  | 3 | 902 | 939 | 888 | 859 | 921 | 901,8 |
|  | 4 | 1175 | 1260 | 1100 | 1221 | 1087 | 1168,6 |
|  | 5 | 1480 | 1260 | 1500 | 1400 | 1600 | 1448 |
|  | 6 | 1747 | 1721 | 1800 | 1700 | 1900 | 1773,6 |
|  | 7 | 2050 | 2225 | 2100 | 2000 | 2150 | 2105 |
|  | 8 | 2333 | 2225 | 2300 | 2400 | 2500 | 2351,6 |
|  | 9 | 2607 | 2798 | 2700 | 2500 | 2800 | 2681 |
|  | 10 | 2877 | 2600 | 2900 | 2800 | 3000 | 2835,4 |
| BH | 1 | 287 | 326 | 338 | 316 | 315 | 316,4 |
|  | 2 | 326 | 338 | 345 | 350 | 360 | 343,8 |
|  | 3 | 338 | 345 | 350 | 360 | 355 | 349,6 |
|  | 4 | 345 | 350 | 360 | 370 | 340 | 353 |
|  | 5 | 350 | 360 | 372 | 380 | 365 | 365,4 |
|  | 6 | 364 | 372 | 380 | 390 | 370 | 375,2 |
|  | 7 | 372 | 380 | 396 | 370 | 360 | 375,6 |
|  | 8 | 380 | 396 | 417 | 370 | 390 | 390,6 |
|  | 9 | 396 | 417 | 380 | 400 | 420 | 402,6 |
|  | 10 | 417 | 396 | 380 | 400 | 410 | 400,6 |
| CH | 1 | 316 | 315 | 331 | 337 | 348 | 329,4 |
|  | 2 | 315 | 331 | 337 | 348 | 330 | 332,2 |
|  | 3 | 331 | 337 | 348 | 360 | 340 | 343,2 |
|  | 4 | 337 | 348 | 360 | 363 | 370 | 355,6 |
|  | 5 | 348 | 360 | 363 | 372 | 340 | 356,6 |
|  | 6 | 363 | 372 | 382 | 370 | 360 | 369,4 |
|  | 7 | 372 | 382 | 400 | 360 | 370 | 376,8 |
|  | 8 | 382 | 400 | 410 | 360 | 370 | 384,4 |
|  | 9 | 400 | 410 | 468 | 370 | 380 | 405,6 |
|  | 10 | 410 | 400 | 468 | 390 | 380 | 409,6 |
| AF | 1 | 468 | 410 | 400 | 450 | 480 | 441,6 |
|  | 2 | 828 | 785 | 900 | 800 | 850 | 832,6 |
|  | 3 | 1260 | 1175 | 1280 | 1200 | 1300 | 1243 |
|  | 4 | 1721 | 1747 | 1800 | 1600 | 1700 | 1713,6 |
|  | 5 | 2225 | 2050 | 2300 | 2200 | 2400 | 2235 |
|  | 6 | 2798 | 2607 | 2900 | 2700 | 2800 | 2761 |
|  | 7 | 3289 | 3333 | 3300 | 3400 | 3200 | 3304,4 |
|  | 8 | 3333 | 3289 | 3400 | 3200 | 3100 | 3264,4 |
|  | 9 | 3889 | 3333 | 4000 | 3700 | 3900 | 3764,4 |
|  | 10 | 4858 | 3889 | 4900 | 5000 | 4700 | 4669,4 |
| BF | 1 | 438 | 400 | 450 | 420 | 430 | 427,6 |
|  | 2 | 527 | 485 | 540 | 500 | 510 | 512,4 |
|  | 3 | 639 | 626 | 620 | 673 | 650 | 641,6 |
|  | 4 | 682 | 673 | 700 | 650 | 660 | 673 |
|  | 5 | 785 | 789 | 800 | 770 | 790 | 786,8 |
|  | 6 | 859 | 785 | 870 | 900 | 850 | 852,8 |
|  | 7 | 939 | 859 | 950 | 900 | 920 | 913,6 |
|  | 8 | 1100 | 1052 | 1087 | 1150 | 1200 | 1117,8 |
|  | 9 | 1109 | 1087 | 1150 | 1200 | 1050 | 1119,2 |
|  | 10 | 1221 | 1220 | 1200 | 1250 | 1300 | 1238,2 |
| CF | 1 | 626 | 620 | 634 | 610 | 640 | 626 |
|  | 2 | 620 | 626 | 634 | 610 | 650 | 628 |
|  | 3 | 634 | 620 | 673 | 650 | 610 | 637,4 |
|  | 4 | 673 | 634 | 682 | 620 | 660 | 653,8 |
|  | 5 | 789 | 785 | 800 | 770 | 790 | 786,8 |
|  | 6 | 888 | 859 | 900 | 870 | 850 | 873,4 |
|  | 7 | 921 | 888 | 939 | 900 | 950 | 919,6 |
|  | 8 | 1052 | 1100 | 1087 | 1150 | 1000 | 1077,8 |
|  | 9 | 1087 | 1052 | 1100 | 1200 | 1150 | 1117,8 |
|  | 10 | 1220 | 1221 | 1200 | 1250 | 1300 | 1238,2 |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 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 | 316,4 | 632 | 901,8 | 1168,6 | 1448 | 1773,6 | 2105 | 2351,6 | 2681 | 2835,4 |
| B | 316,4 | 343,8 | 349,6 | 353 | 365,4 | 375,2 | 375,6 | 390,6 | 402,6 | 400,6 |
| C | 329,4 | 332,2 | 343,2 | 355,6 | 356,6 | 369,4 | 376,8 | 384,4 | 405,6 | 409,6 |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Файлы на flash-накопителе (F) | | | | | | | | | |
| N=1 | N=2 | N=3 | N=4 | N=5 | N=6 | N=7 | N=8 | N=9 | N=10 |
| 441,6 | 832,6 | 1243 | 1713,6 | 2235 | 2761 | 3304,4 | 3264,4 | 3889 | 4669,4 |
| 427,6 | 512,4 | 641,6 | 673 | 786,8 | 852,8 | 913,6 | 1117,8 | 1119,2 | 1238,2 |
| 626 | 628 | 637,4 | 653,8 | 786,8 | 873,4 | 919,6 | 1077,8 | 1117,8 | 1238,2 |

# ДИАГРАММЫ ТРАСС ПОТОКОВ

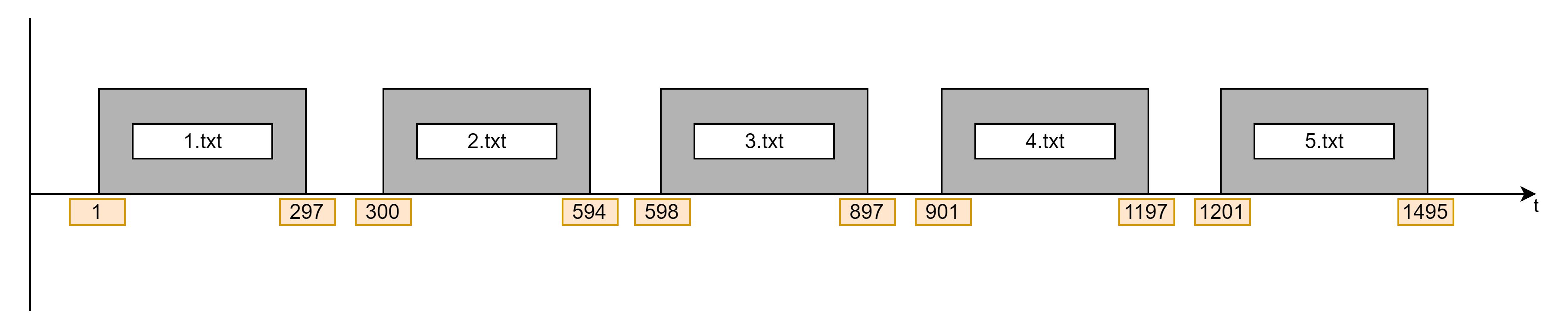


Рисунок 14 – Диаграмма трасс потоков программы AH с 5 текстовыми файлами

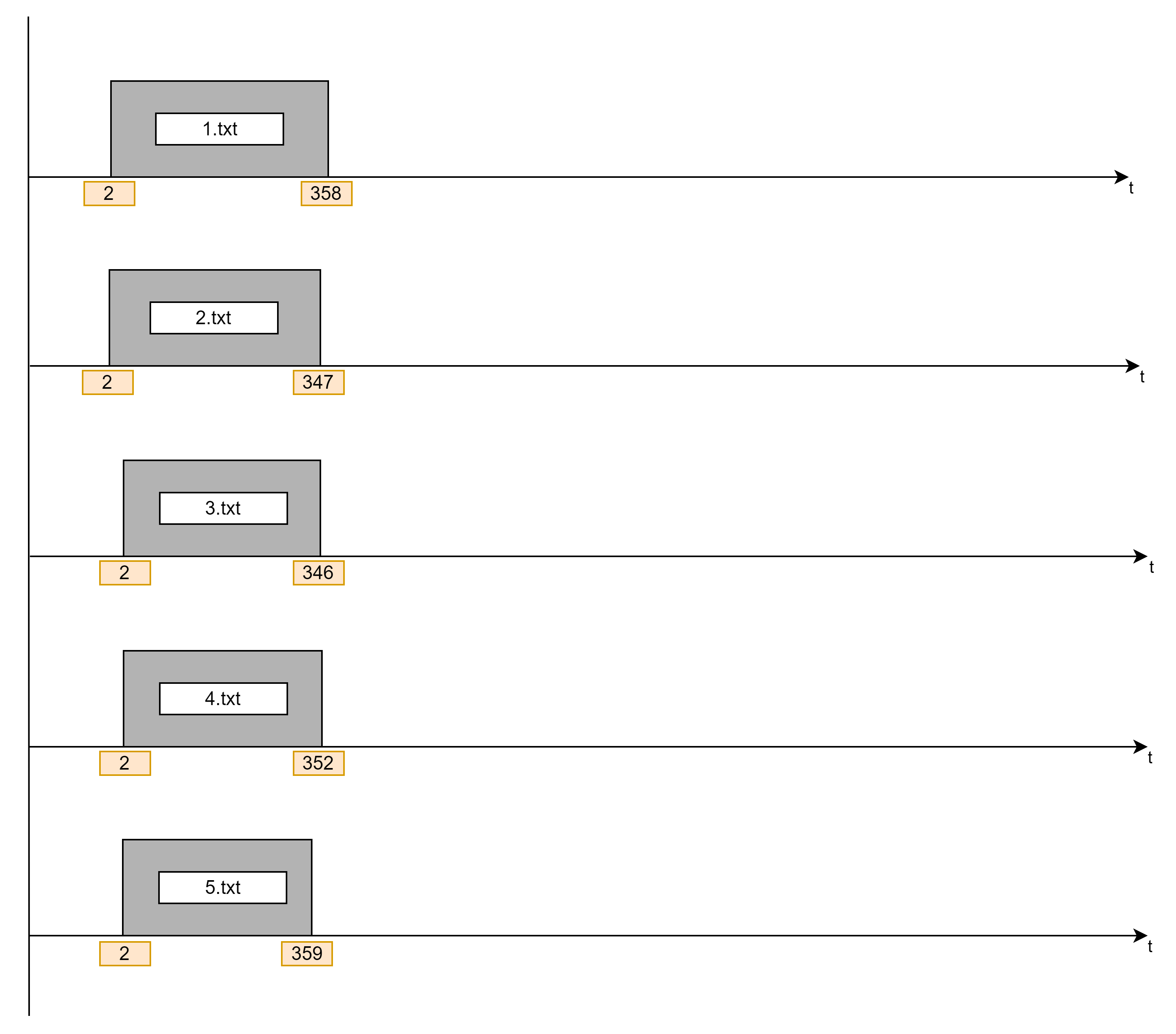


Рисунок 15 – Диаграмма трасс потоков программы BH с 5 текстовыми файлами

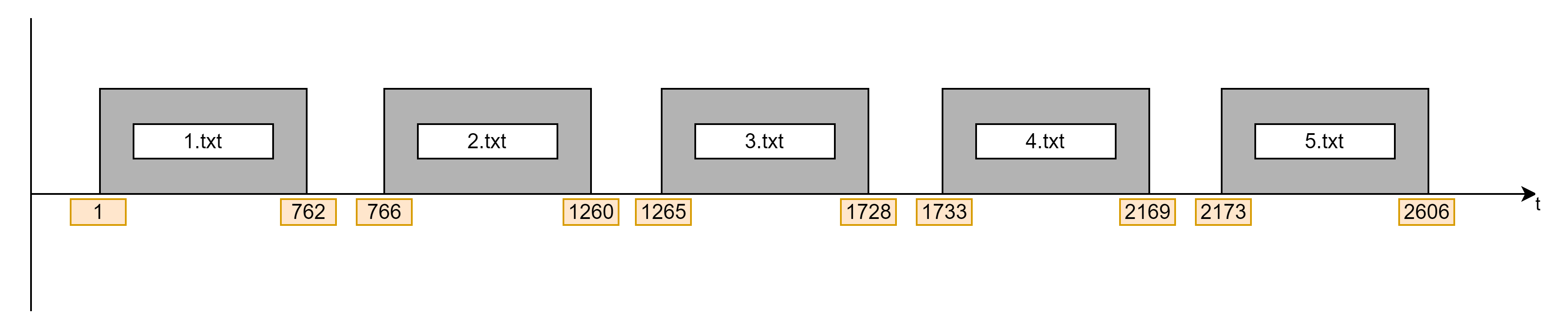


Рисунок 16 – Диаграмма трасс потоков программы AF с 5 текстовыми файлами

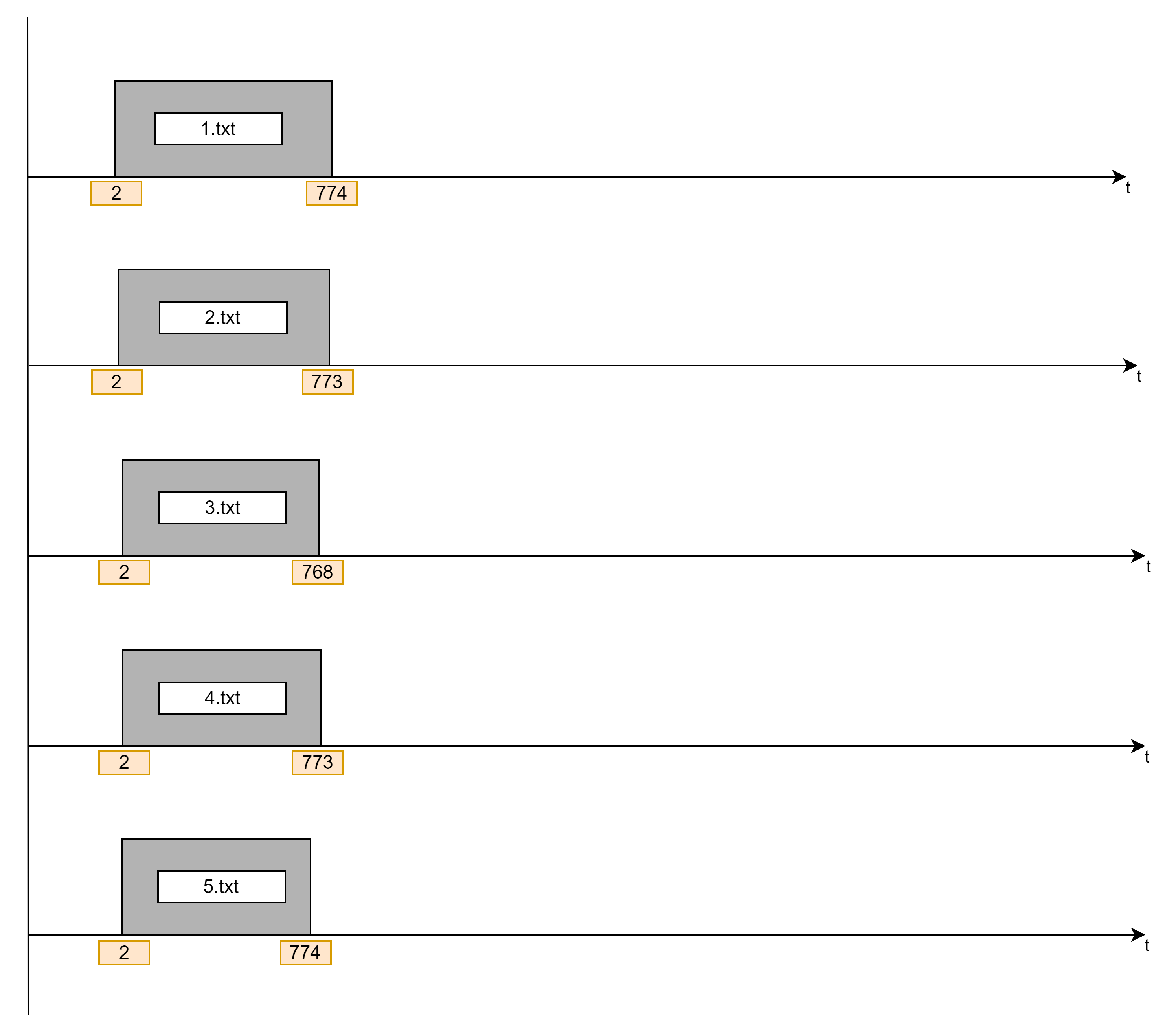


Рисунок 17 – Диаграмма трасс потоков программы BF с 5 текстовыми файлами

# ВЫВОДЫ ПО РЕЗУЛЬТАТАМ ПРОВЕДЕННГО ЭКСПЕРИМЕНТА

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

В многопоточном режиме (B) время обработки файлов существенно сократилось по сравнению с однопоточным режимом (A). Среднее время обработки при использовании многопоточности было значительно меньше, что подтверждает эффективность параллельной обработки при наличии многопроцессорных систем.

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

Разница во времени обработки между режимами A и B при использовании flash-накопителя также показывает преимущество многопоточности, хотя прирост производительности не столь велик, как на жестком диске. Это может быть связано с более медленной скоростью чтения/записи данных на flash-накопителе.

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

Многопоточность позволила равномерно распределить нагрузку между процессорными ядрами, что снизило время обработки каждого файла и уменьшило время ожидания ресурсов, таких как доступ к файлам.

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

Повышение приоритета потоков в программе C привело к еще более быстрому выполнению задач, особенно на загруженной системе. Однако на менее нагруженной системе разница с обычной многопоточностью не столь значительна.

**Влияние количества ядер процессора на результаты**:

При увеличении количества ядер процессора наблюдался значительный прирост производительности многопоточных программ, особенно при обработке большого количества файлов.

**Необходимость использования средств синхронизации потоков**:

Для корректного выполнения многопоточных программ была использована синхронизация потоков (критические секции), что исключило проблемы конкурентного доступа к общим ресурсам и повысило стабильность выполнения программы.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

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