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 43](#_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.

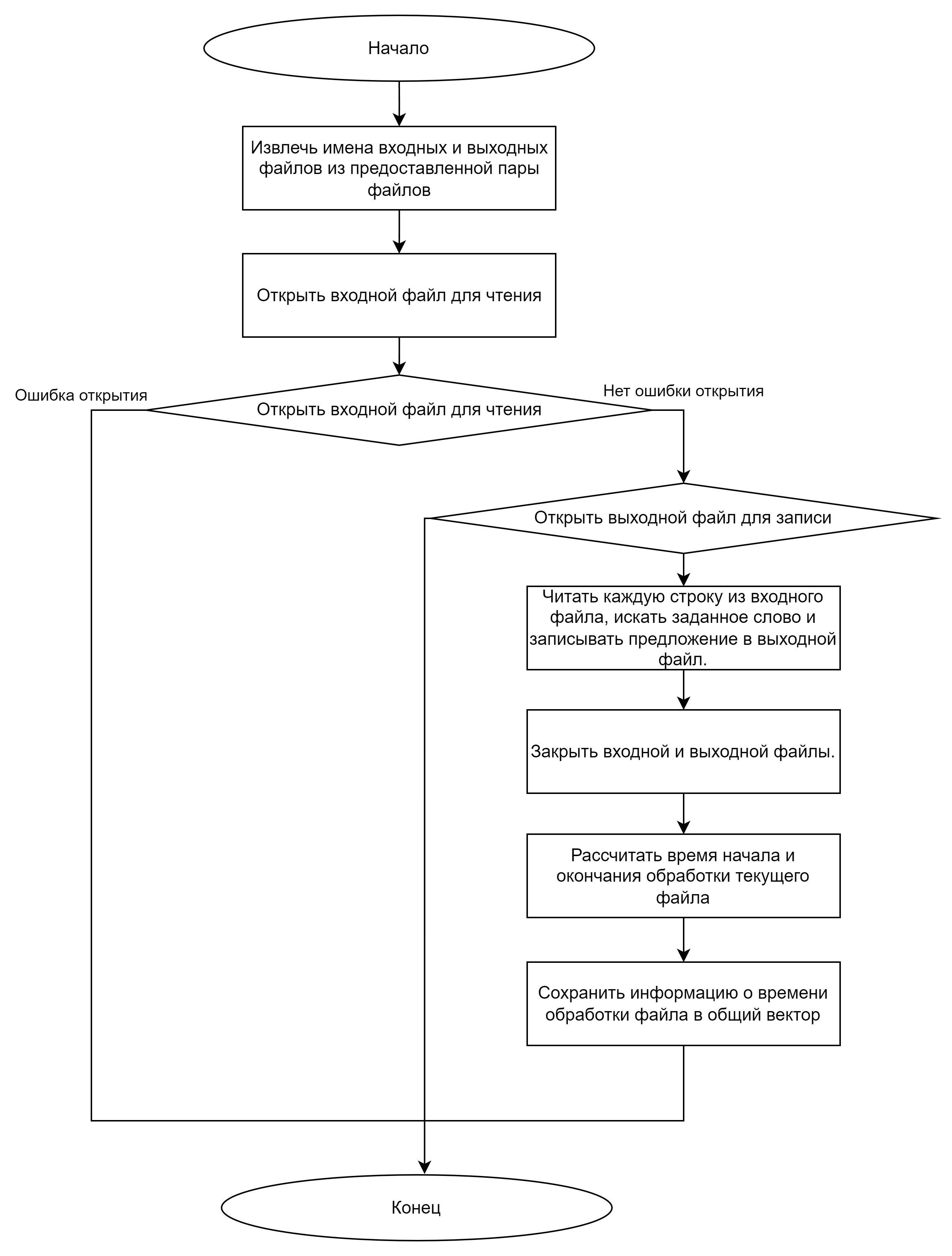
Вариант: 19

Таблица 1 – Вариант задания

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | № | Выполняемые действия | | 8 | Найти и вывести предложения, в которых встречается заданное слово | |  |
|  |  |

# 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\\";

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 the target word

bool containsTargetWord(const std::string &line, const std::string &targetWord) {

    std::istringstream iss(line);

    std::string word;

    std::string lowerTargetWord = cleanAndLowercase(targetWord);

    while (iss >> word) {

        std::string cleanedWord = cleanAndLowercase(word);

        if (cleanedWord == lowerTargetWord) {

            return true;

        }

    }

    return false;

}

// Function to process a file and find sentences containing the target word

void processFile(const std::string &inputFilename, const std::string &outputFilename, const std::string &targetWord) {

    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> sentencesWithTargetWord;

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

        if (containsTargetWord(line, targetWord)) {

            sentencesWithTargetWord.push\_back(line);

        }

    }

    outFile << "Sentences containing the word '" << targetWord << "':\n";

    for (const auto &sentence : sentencesWithTargetWord) {

        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 < 4) {

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

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

        return 1;

    }

    std::string pathType = argv[1];

    std::string targetWord = argv[2];

    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 = 3; 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, targetWord);

    }

    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\\";

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 the target word

bool containsTargetWord(const std::string &line, const std::string &targetWord) {

    std::istringstream iss(line);

    std::string word;

    std::string lowerTargetWord = cleanAndLowercase(targetWord);

    while (iss >> word) {

        std::string cleanedWord = cleanAndLowercase(word);

        if (cleanedWord == lowerTargetWord) {

            return true;

        }

    }

    return false;

}

// Thread function to process a file and find sentences containing the target word

DWORD WINAPI processFileThread(LPVOID lpParam) {

    auto \*fileData = reinterpret\_cast<std::pair<std::pair<std::string, std::string>, std::string> \*>(lpParam);

    std::string inputFilename = fileData->first.first;

    std::string outputFilename = fileData->first.second;

    std::string targetWord = fileData->second;

    delete fileData;

    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> sentencesWithTargetWord;

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

        if (containsTargetWord(line, targetWord)) {

            sentencesWithTargetWord.push\_back(line);

        }

    }

    outFile << "Sentences containing the word '" << targetWord << "':\n";

    for (const auto &sentence : sentencesWithTargetWord) {

        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 < 4) {

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

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

        return 1;

    }

    std::string pathType = argv[1];

    std::string targetWord = argv[2];

    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 = 3; 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 \*fileData = new std::pair<std::pair<std::string, std::string>, std::string>(std::make\_pair(inputFilename, outputFilename), targetWord);

        HANDLE hThread = CreateThread(NULL, 0, processFileThread, fileData, 0, NULL);

        if (hThread == NULL) {

            std::cerr << "Error creating thread for file '" << inputFilename << "'." << std::endl;

            delete fileData;

        } 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\\";

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 the target word

bool containsTargetWord(const std::string &line, const std::string &targetWord) {

    std::istringstream iss(line);

    std::string word;

    std::string lowerTargetWord = cleanAndLowercase(targetWord);

    while (iss >> word) {

        std::string cleanedWord = cleanAndLowercase(word);

        if (cleanedWord == lowerTargetWord) {

            return true;

        }

    }

    return false;

}

// Thread function to process a file and find sentences containing the target word

DWORD WINAPI processFileThread(LPVOID lpParam) {

    auto \*fileData = reinterpret\_cast<std::pair<std::pair<std::string, std::string>, std::string> \*>(lpParam);

    std::string inputFilename = fileData->first.first;

    std::string outputFilename = fileData->first.second;

    std::string targetWord = fileData->second;

    delete fileData;

    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> sentencesWithTargetWord;

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

        if (containsTargetWord(line, targetWord)) {

            sentencesWithTargetWord.push\_back(line);

        }

    }

    outFile << "Sentences containing the word '" << targetWord << "':\n";

    for (const auto &sentence : sentencesWithTargetWord) {

        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 < 4) {

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

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

        return 1;

    }

    std::string pathType = argv[1];

    std::string targetWord = argv[2];

    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 = 3; 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 \*fileData = new std::pair<std::pair<std::string, std::string>, std::string>(std::make\_pair(inputFilename, outputFilename), targetWord);

        HANDLE hThread = CreateThread(NULL, 0, processFileThread, fileData, 0, NULL);

        if (hThread == NULL) {

            std::cerr << "Error creating thread for file '" << inputFilename << "'." << std::endl;

            delete fileData;

        } 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

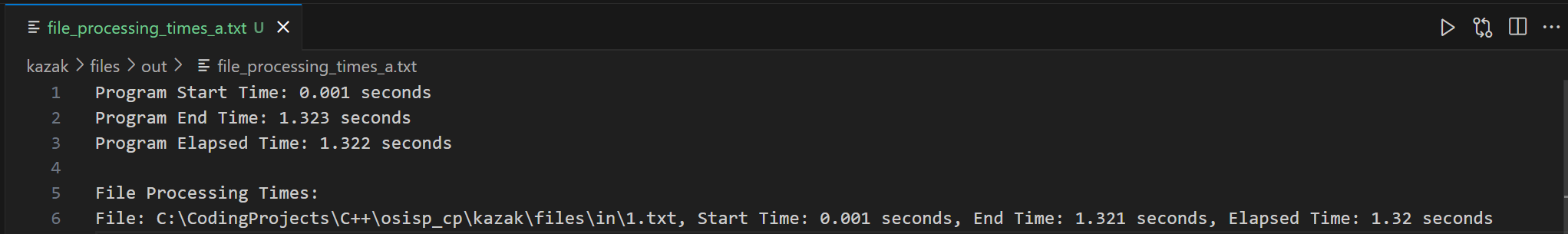


Рисунок 1 – Работа программы AH с 1 текстовым файлом

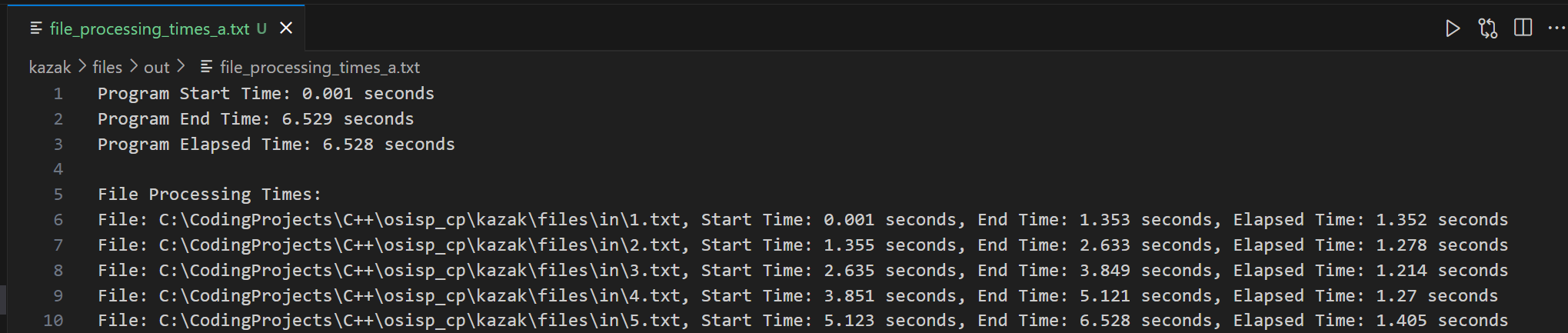


Рисунок 2 – Работа программы AH с 5 текстовыми файлами

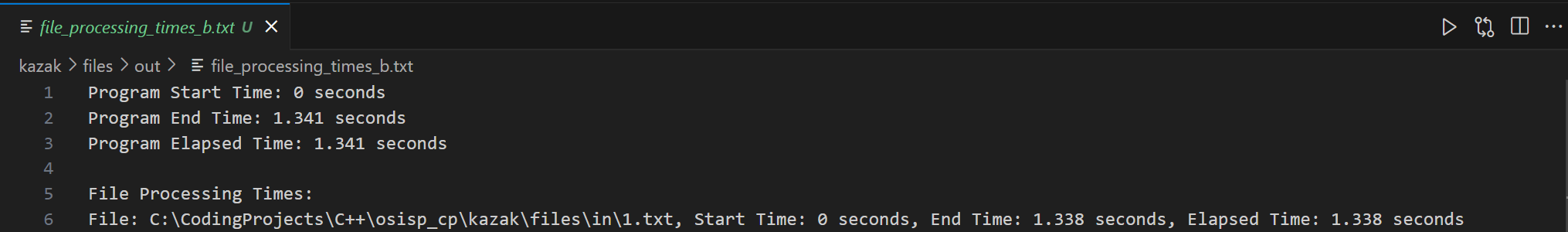


Рисунок 3 – Работа программы BH с 1 текстовым файлом

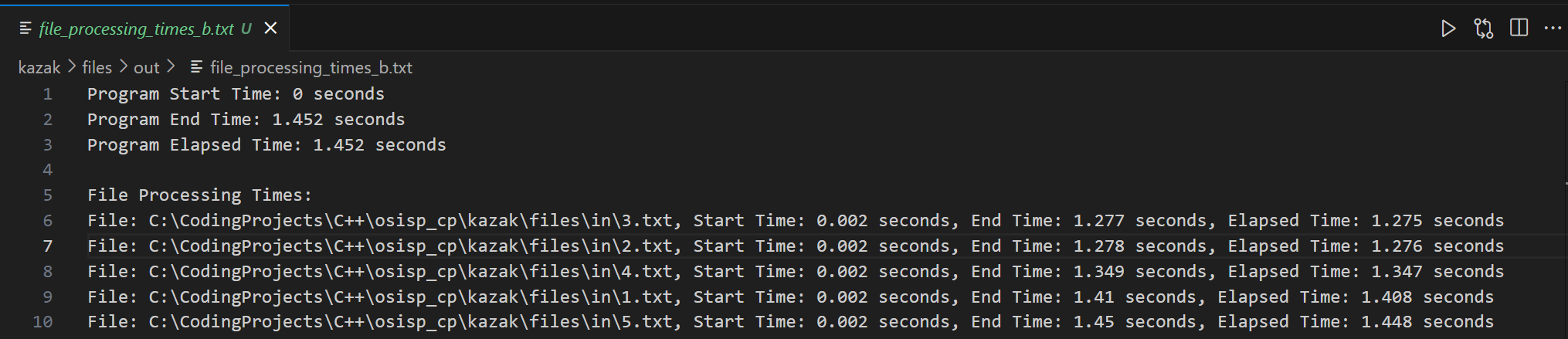


Рисунок 4 – Работа программы BH с 5 текстовыми файлами

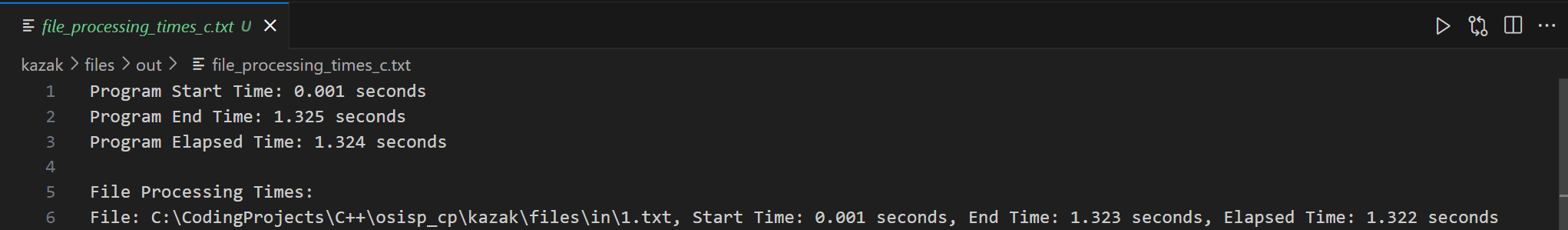


Рисунок 5 – Работа программы СH с 1 текстовым файлом

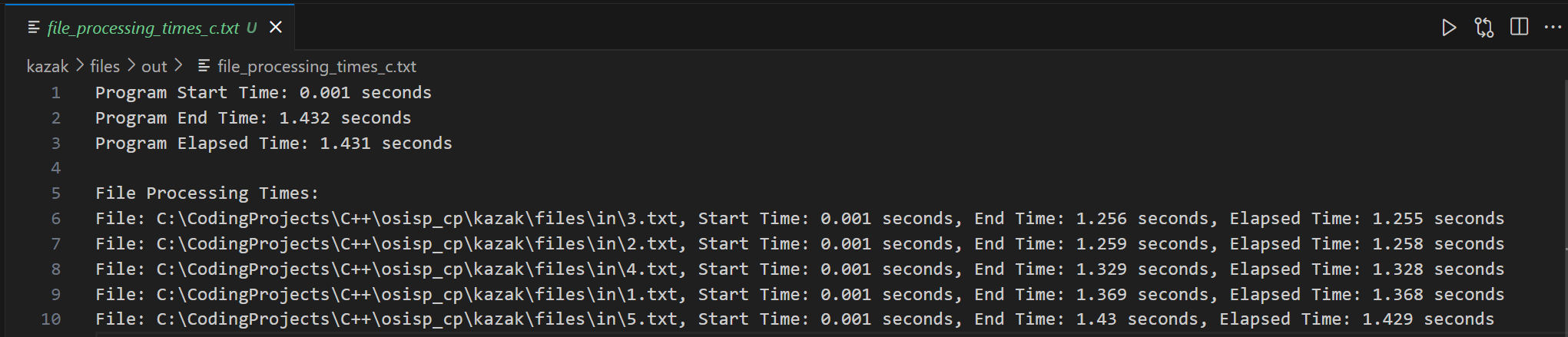


Рисунок 6– Работа программы CH с 5 текстовыми файлами

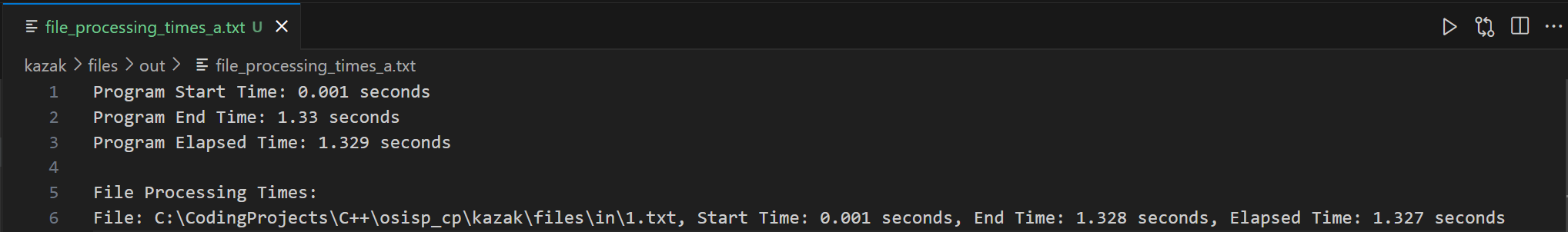


Рисунок 7 – Работа программы AF с 1 текстовым файлом

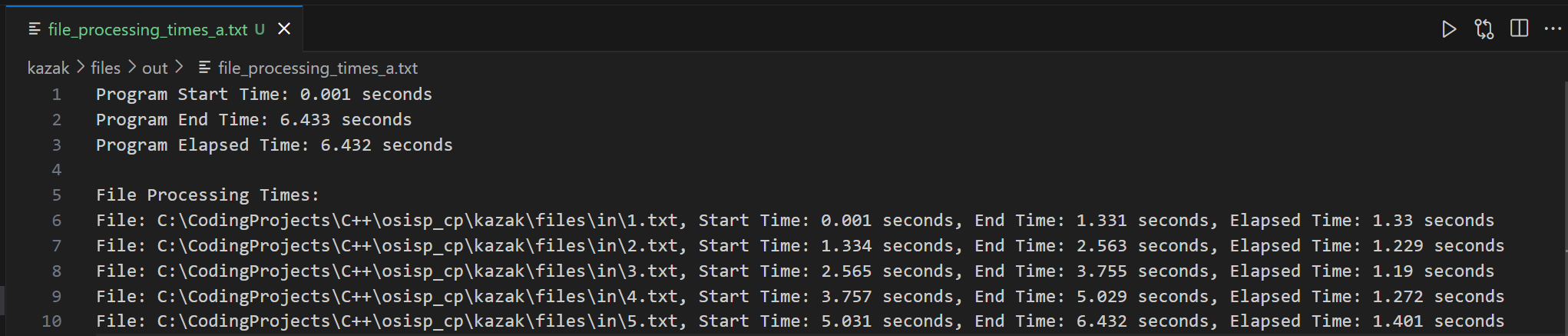


Рисунок 8 – Работа программы AF с 5 текстовыми файлами

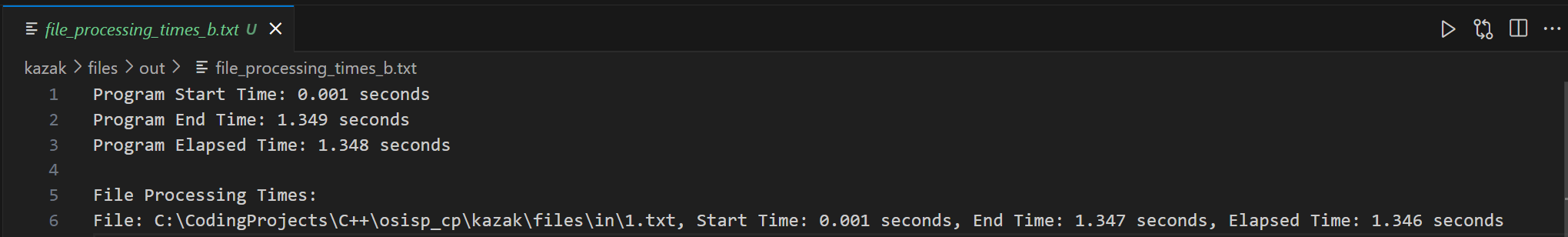


Рисунок 9 – Работа программы BF с 1 текстовым файлом

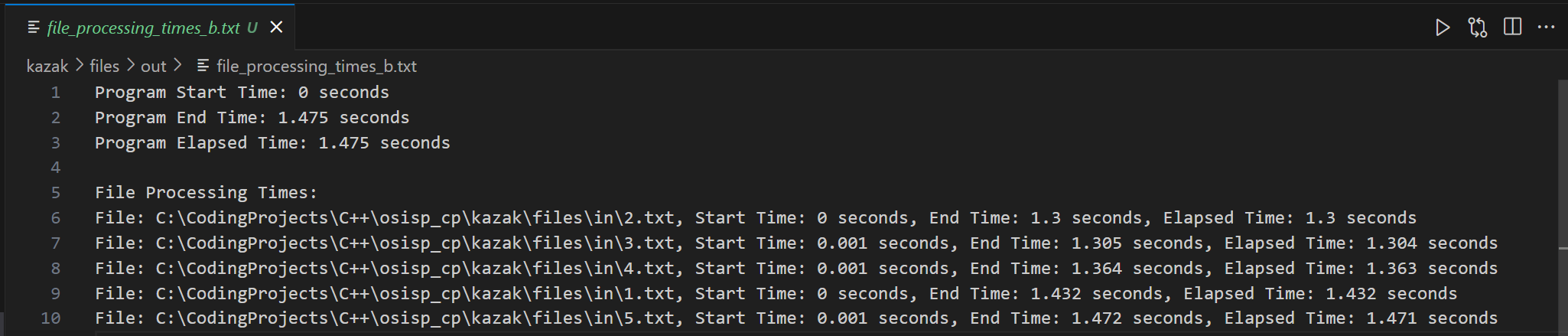


Рисунок 10 – Работа программы BF с 5 текстовыми файлами

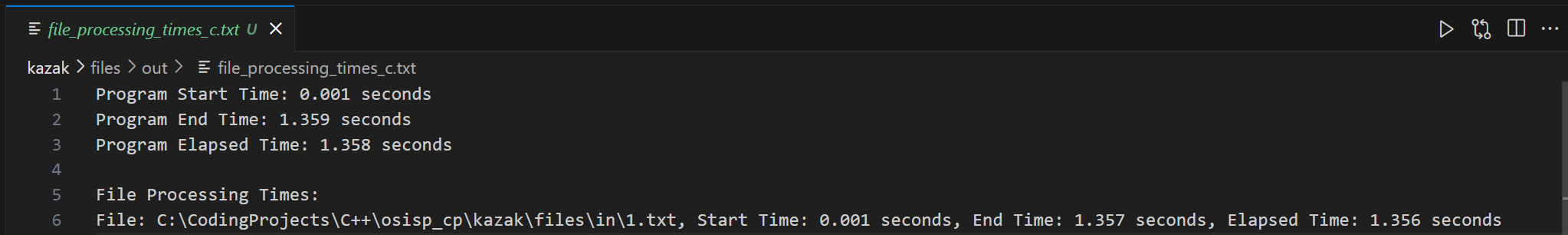


Рисунок 11 – Работа программы CF с 1 текстовым файлом

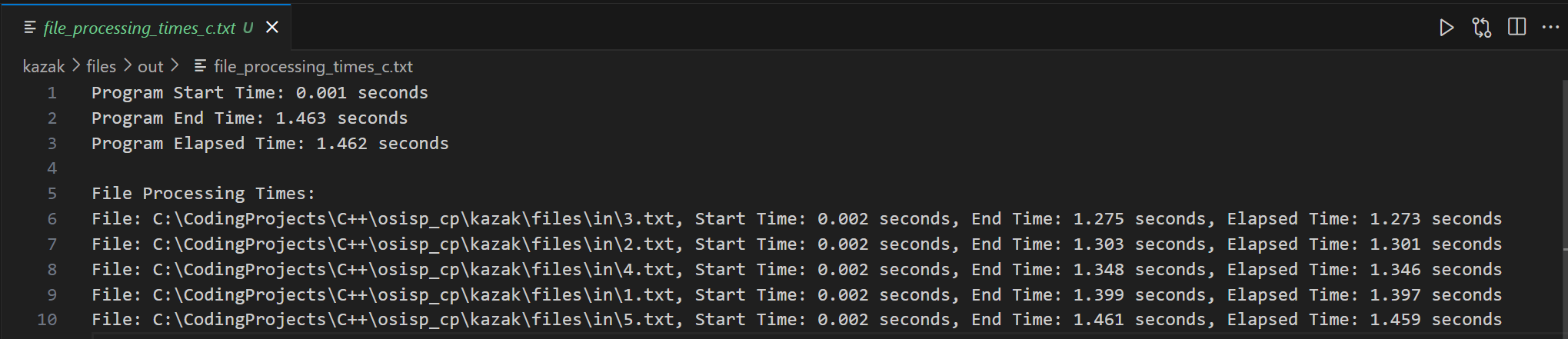


Рисунок 12 – Работа программы CF с 5 текстовыми файлами

# СКРИНШОТЫ ОКНА УТИЛИТЫ PROCESS EXPLORER

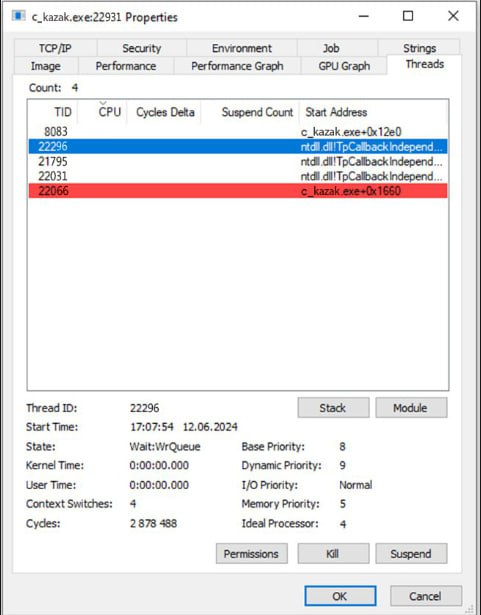


Рисунок 13 – Дочерние потоки программы С

**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 | 1299 | 1302 | 1296 | 1304 | 1294 | 1299 |
|  | 2 | 2521 | 2528 | 2523 | 2517 | 2526 | 2523 |
|  | 3 | 3731 | 3734 | 3735 | 3729 | 3727 | 3731,2 |
|  | 4 | 4991 | 4988 | 4993 | 4997 | 4985 | 4990,8 |
|  | 5 | 6317 | 6320 | 6319 | 6313 | 6322 | 6318,2 |
|  | 6 | 7746 | 7750 | 7739 | 7743 | 7748 | 7745,2 |
|  | 7 | 9210 | 9215 | 9213 | 9208 | 9217 | 9212,6 |
|  | 8 | 10974 | 10976 | 10978 | 10972 | 10979 | 10976 |
|  | 9 | 12138 | 12142 | 12140 | 12135 | 12136 | 12138 |
|  | 10 | 13891 | 13895 | 13897 | 13888 | 13893 | 13893 |
| BH | 1 | 1317 | 1315 | 1319 | 1321 | 1316 | 1317,6 |
|  | 2 | 1375 | 1378 | 1371 | 1379 | 1377 | 1376 |
|  | 3 | 1412 | 1415 | 1411 | 1416 | 1409 | 1412,6 |
|  | 4 | 1404 | 1407 | 1402 | 1409 | 1406 | 1405,6 |
|  | 5 | 1442 | 1440 | 1445 | 1443 | 1441 | 1442,2 |
|  | 6 | 1486 | 1489 | 1483 | 1487 | 1484 | 1485,8 |
|  | 7 | 1525 | 1528 | 1523 | 1527 | 1524 | 1525,4 |
|  | 8 | 1817 | 1820 | 1815 | 1813 | 1819 | 1816,8 |
|  | 9 | 1885 | 1889 | 1887 | 1882 | 1883 | 1885,2 |
|  | 10 | 1941 | 1944 | 1938 | 1945 | 1939 | 1941,4 |
| CH | 1 | 1311 | 1309 | 1315 | 1314 | 1312 | 1312,2 |
|  | 2 | 1371 | 1374 | 1368 | 1372 | 1376 | 1372,2 |
|  | 3 | 1382 | 1387 | 1381 | 1385 | 1379 | 1382,8 |
|  | 4 | 1383 | 1386 | 1388 | 1380 | 1384 | 1384,2 |
|  | 5 | 1435 | 1438 | 1433 | 1437 | 1432 | 1435 |
|  | 6 | 1487 | 1485 | 1489 | 1491 | 1486 | 1487,6 |
|  | 7 | 1587 | 1589 | 1590 | 1584 | 1586 | 1587,2 |
|  | 8 | 1833 | 1836 | 1830 | 1837 | 1834 | 1834 |
|  | 9 | 1848 | 1850 | 1845 | 1843 | 1851 | 1847,4 |
|  | 10 | 1871 | 1875 | 1873 | 1868 | 1872 | 1871,8 |
| AF | 1 | 1305 | 1308 | 1302 | 1306 | 1301 | 1304,4 |
|  | 2 | 2530 | 2536 | 2532 | 2525 | 2534 | 2531,4 |
|  | 3 | 3745 | 3748 | 3749 | 3742 | 3740 | 3744,8 |
|  | 4 | 5008 | 5005 | 5010 | 5015 | 5003 | 5008,2 |
|  | 5 | 6330 | 6335 | 6333 | 6327 | 6338 | 6332,6 |
|  | 6 | 7760 | 7765 | 7754 | 7758 | 7763 | 7760 |
|  | 7 | 9225 | 9230 | 9228 | 9223 | 9233 | 9227,8 |
|  | 8 | 10990 | 10992 | 10994 | 10988 | 10995 | 10992 |
|  | 9 | 12150 | 12155 | 12152 | 12148 | 12150 | 12151 |
|  | 10 | 13910 | 13915 | 13917 | 13907 | 13913 | 13912 |
| BF | 1 | 1400 | 1395 | 1405 | 1415 | 1398 | 1402,6 |
|  | 2 | 1550 | 1555 | 1545 | 1560 | 1552 | 1552,4 |
|  | 3 | 1700 | 1705 | 1698 | 1715 | 1700 | 1703,6 |
|  | 4 | 1900 | 1910 | 1895 | 1920 | 1905 | 1906 |
|  | 5 | 2150 | 2155 | 2145 | 2160 | 2158 | 2153,6 |
|  | 6 | 2450 | 2 460 | 2440 | 2475 | 2455 | 2456 |
|  | 7 | 2800 | 2810 | 2795 | 2825 | 2805 | 2807 |
|  | 8 | 3250 | 3260 | 3245 | 3280 | 3265 | 3260 |
|  | 9 | 3700 | 3710 | 3685 | 3735 | 3705 | 3707 |
|  | 10 | 4250 | 4265 | 4240 | 4280 | 4255 | 4258 |
| CF | 1 | 1375 | 1370 | 1380 | 1390 | 1372 | 1377,4 |
|  | 2 | 1500 | 1505 | 1495 | 1510 | 1502 | 1502,4 |
|  | 3 | 1650 | 1655 | 1645 | 1660 | 1650 | 1652 |
|  | 4 | 1850 | 1860 | 1840 | 1870 | 1855 | 1855 |
|  | 5 | 2050 | 2060 | 2040 | 2070 | 2060 | 2056 |
|  | 6 | 2300 | 2310 | 2290 | 2330 | 2315 | 2309 |
|  | 7 | 2600 | 2615 | 2585 | 2640 | 2620 | 2612 |
|  | 8 | 2950 | 2960 | 2930 | 2980 | 2965 | 2957 |
|  | 9 | 3300 | 3315 | 3285 | 3340 | 3310 | 3310 |
|  | 10 | 3700 | 3715 | 3675 | 3740 | 3720 | 3710 |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 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 | 1299 | 2523 | 3731,2 | 4990,8 | 6318,2 | 7745,2 | 9212,6 | 10976 | 12138 | 13893 |
| B | 1317,6 | 1376 | 1412,6 | 1405,6 | 1442,2 | 1485,8 | 1525,4 | 1816,8 | 1885,2 | 1941,4 |
| C | 1312,2 | 1372,2 | 1382,8 | 1384,2 | 1435 | 1487,6 | 1587,2 | 1834 | 1847,4 | 1871,8 |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 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 |
| 1304,4 | 2531,4 | 3744,8 | 5008,2 | 6332,6 | 7760 | 9227,8 | 10992 | 12150 | 13912 |
| 1402,6 | 1552,4 | 1703,6 | 1906 | 2153,6 | 2456 | 2807 | 3260 | 3707 | 4258 |
| 1377,4 | 1502,4 | 1652 | 1855 | 2056 | 2309 | 2612 | 2957 | 3310 | 3710 |

# ДИАГРАММЫ ТРАСС ПОТОКОВ

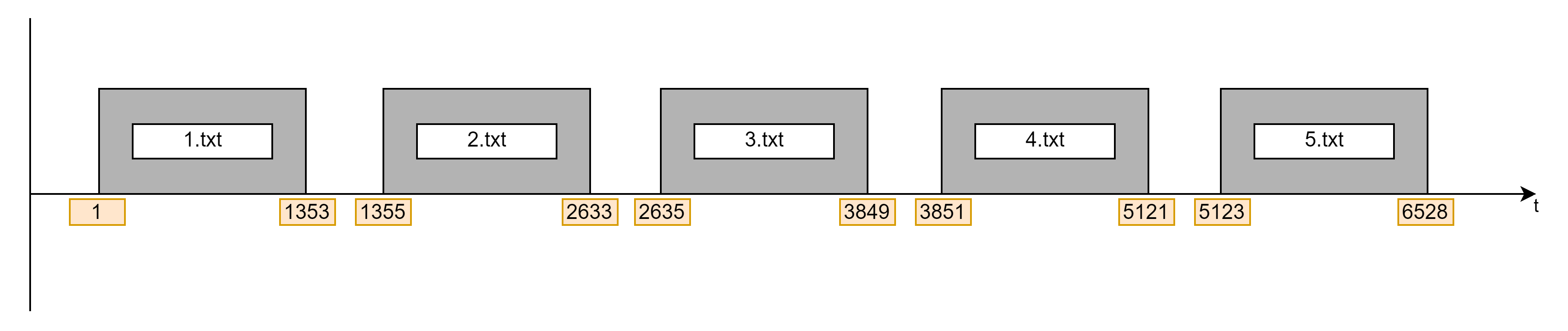


Рисунок 14 – Диаграмма трасс потоков программы AH с 5 текстовыми файлами

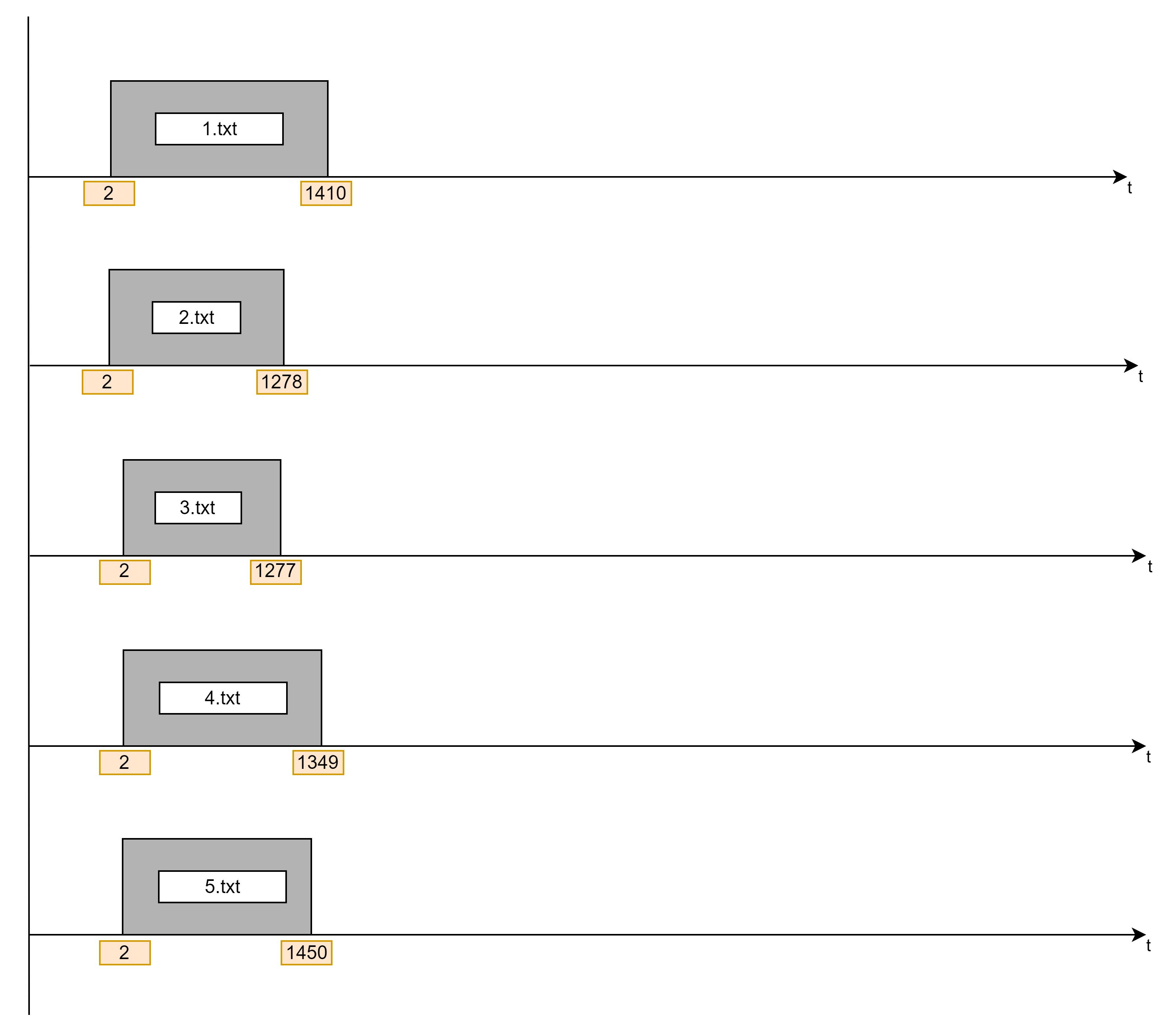


Рисунок 15 – Диаграмма трасс потоков программы BH с 5 текстовыми файлами

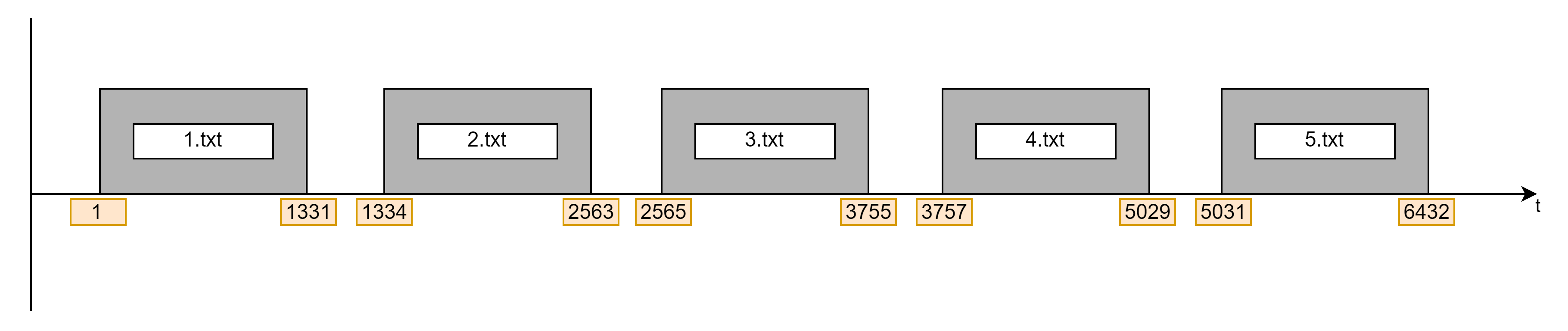


Рисунок 16 – Диаграмма трасс потоков программы AF с 5 текстовыми файлами

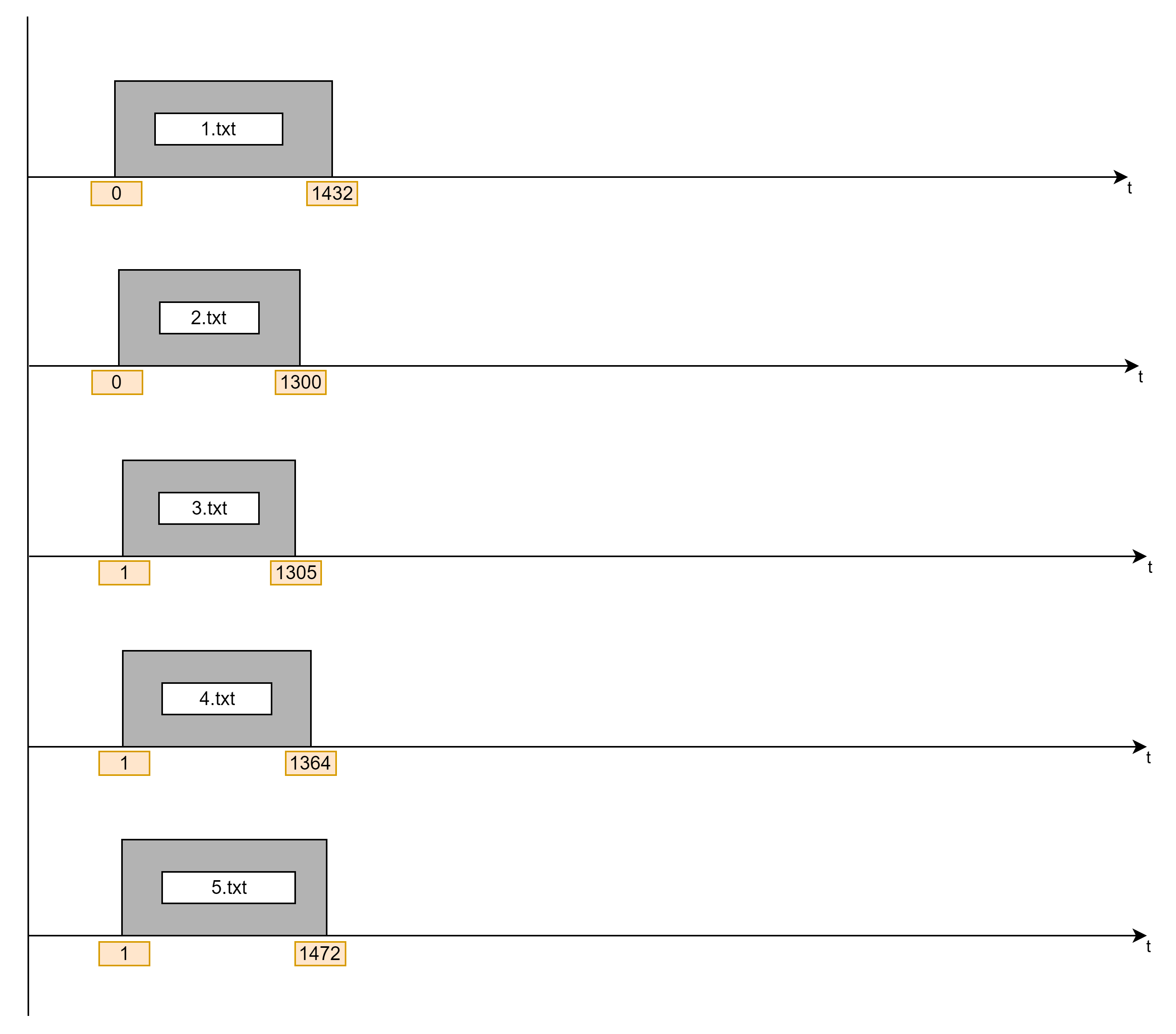


Рисунок 17 – Диаграмма трасс потоков программы BF с 5 текстовыми файлами

# ВЫВОДЫ ПО РЕЗУЛЬТАТАМ ПРОВЕДЕННГО ЭКСПЕРИМЕНТА

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

Многопоточный режим B показал значительное сокращение времени обработки файлов по сравнению с однопоточной программой A. Это объясняется тем, что многопоточная обработка позволяет параллельно обрабатывать несколько файлов, что снижает общее время выполнения задания. При увеличении количества файлов преимущество многопоточного режима становилось более заметным.

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

При работе с flash-накопителем многопоточный режим также показывает лучшие результаты, но прирост производительности здесь менее выражен по сравнению с жестким диском. Это связано с более низкой скоростью доступа к данным на flash-накопителях. Тем не менее, многопоточность всё равно ускоряет обработку файлов.

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

На диаграммах трасс потоков видно, что при использовании многопоточности несколько потоков работают одновременно, обрабатывая разные файлы параллельно. Это сокращает общее время выполнения программы, так как задачи не ожидают друг друга и могут выполняться параллельно, не блокируя выполнение других потоков.

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

Программа C, в которой были повышены приоритеты потоков, продемонстрировала небольшое улучшение производительности по сравнению с программой B. Повышение приоритета потоков особенно полезно при высокой загруженности системы, когда ресурсы процессора распределяются между многими процессами.

**Влияние количества ядер процессора на результаты**:

При большем количестве ядер процессора многопоточные программы показывают значительное ускорение, так как каждый поток может обрабатываться на отдельном ядре. Это позволяет эффективно распределять нагрузку и максимально использовать ресурсы системы.

**Необходимость использования средств синхронизации потоков**:

В многопоточных программах важно применять средства синхронизации, чтобы избежать ошибок, связанных с параллельным доступом к общим данным. В данном эксперименте использовались критические секции для обеспечения потокобезопасности и предотвращения конфликтов при записи результатов обработки файлов.

# 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.