Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет   
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Операционные среды и системное программирование

ОТЧЁТ

к лабораторной работе №2

на тему

**РАБОТА С ФАЙЛАМИ**

Выполнил: студент гр.253505 Гимпель К.А.

Проверил: ассистент кафедры информатики Гриценко Н.Ю.

Минск 2024

**СОДЕРЖАНИЕ**

[1 Формулировка задачи 3](#_Toc178780699)

[2 Краткие теоритические сведения 4](#_Toc178780700)

[3 Результат запуска программы 5](#_Toc178780701)

[3.1 Выбор процесса для запуска 5](#_Toc178780702)

[3.2 Иллюстрация запуска и работы программы 5](#_Toc178780703)

Заключение 7

Список использованных источников 8

Приложение А (обязательное) Исходный код программы 9

# 1 ФОРМУЛИРОВКА ЗАДАЧИ

Целью выполнения лабораторной работы является расширенные возможности и механизмы работы с файлами и организации ввода-вывода: неблокирующие и асинхронные операции, мультиплексирование ввода-вывода, отображение файлов в память.

Задача заключается в реализации асинхронной обработки данных из файла, которая должна выполнять следующие функции:

– асинхронное чтение содержимого файла;

– сортировка прочитанных данных в памяти с использованием функции обработки;

– асинхронная запись обработанных данных в выходной файл;

– отображение возникающих ошибок.

Специальных требований к приложению не предъявляется; в частности, оно может быть не обязательно оконными, но также и консольным.

2 КРАТКИЕ ТЕОРИТИЧЕСКИЕ СВЕДЕНИЯ

Файловый ввод-вывод в операционной системе Windows основывается на использовании API функций, которые предоставляют гибкий и эффективный способ работы с файлами. Одним из основных элементов является дескриптор файла (Handle), который представляет собой ссылку на объект файла, созданную системой после вызова функции CreateFile(). Этот механизм обеспечивает возможность выполнять операции чтения, записи и управления файлами, а завершение работы с дескриптором осуществляется с помощью функции CloseHandle().

Асинхронный ввод-вывод, также известный как перекрывающийся, представляет собой модель, при которой операции чтения или записи выполняются без блокировки основного потока. В Windows этот подход реализован с помощью структуры OVERLAPPED, которая позволяет выполнять несколько операций одновременно. Функции ReadFile() и WriteFile() при использовании этой структуры возвращают управление сразу после начала операции, а контроль завершения осуществляется с помощью таких функций, как GetOverlappedResult() или механизма ожидания (WaitForSingleObject() или WaitForMultipleObjects()).

Примером применения асинхронного ввода-вывода является обработка больших объемов данных или работа с несколькими файлами одновременно. Это позволяет существенно повысить производительность системы за счет оптимального использования времени процессора.

Эти механизмы подробно описаны в книге Таненбаума Э.С. «Современные операционные системы» [1], где рассматриваются особенности реализации файловых систем и ввода-вывода в различных операционных системах. Более практический подход к изучению ввода-вывода в Windows, включая использование асинхронного API, изложен в книге Сильбершаца А. «Операционные системы: принципы и практика» [2].

3 РЕЗУЛЬТАТ ЗАПУСКА ПРОГРАММЫ

## 3.1 Шаги запуска программы

Программа для асинхронной обработки данных из файла была успешно разработана и запущена с использованием среды разработки Visual Studio. Ниже описаны основные шаги запуска:

– Открываем проект в Visual Studio.

– Компилируем код через встроенную систему сборки Visual Studio.

– Запускаем программу, нажимая сочетание клавиш «Ctrl+F5».

В процессе работы программа автоматически выполняет следующие задачи:

– Асинхронное чтение содержимого входного файла.

– Сортировку данных в памяти с помощью пользовательской функции обработки.

– Асинхронную запись обработанных данных в выходной файл.

– Отображение сообщений об ошибках в случае возникновения.

## 3.2 Иллюстрация запуска и работы программы

На рисунке 3.1 представлен пример содержимого входного файла input.dat, который содержит строки с данными для обработки. На рисунке 3.2 показан результат работы программы, записанный в выходной файл output.dat, где данные отсортированы согласно заданной логике. Консольный вывод подтверждает успешное выполнение всех операций и отображает общее время выполнения (рисунок 3.3).

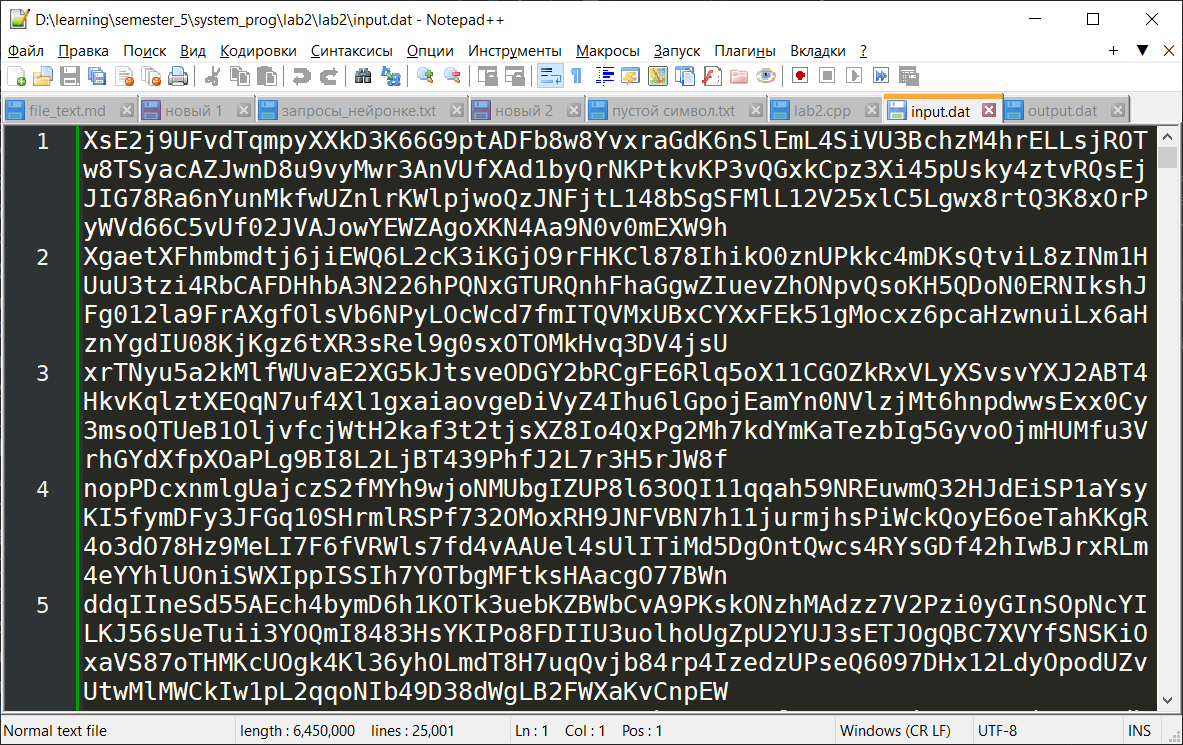


Рисунок 3.1 – Содержимое входного файла input.dat

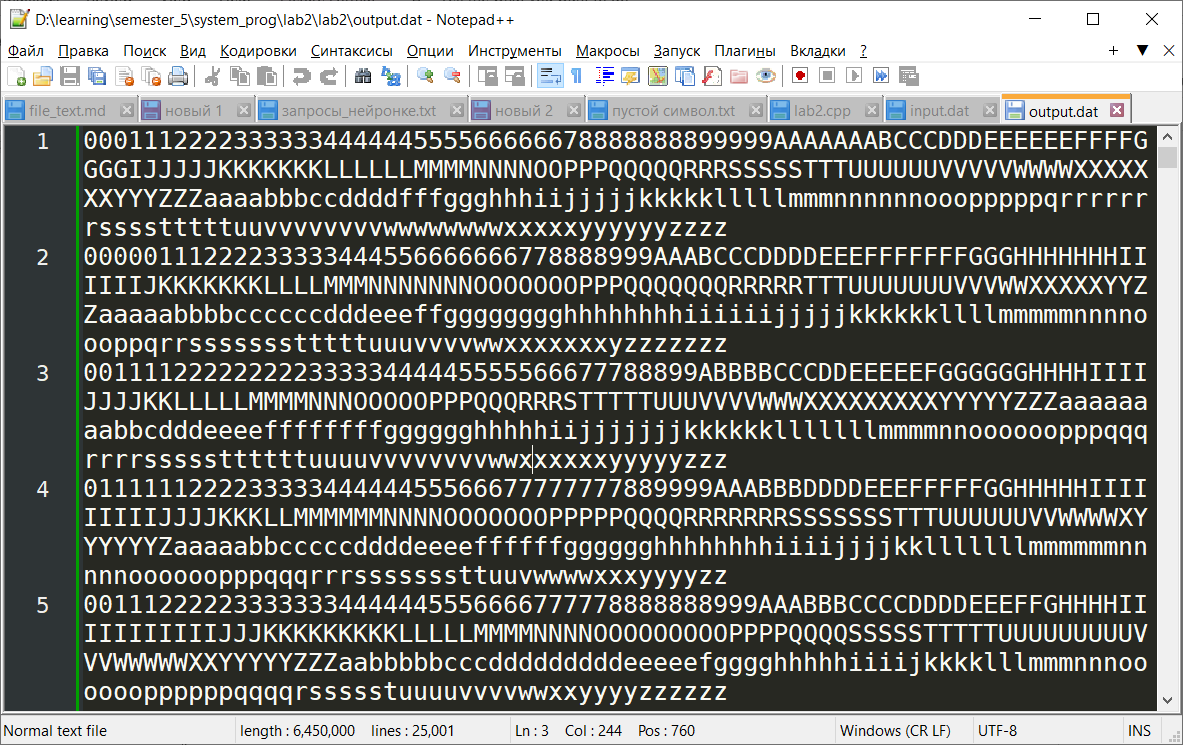


Рисунок 3.2 – Содержимое выходного файла output.dat

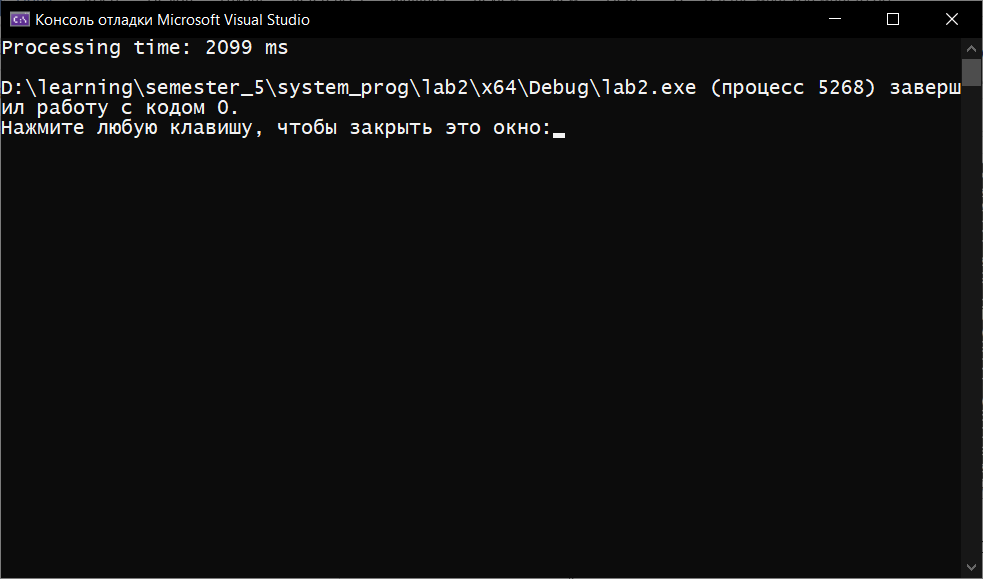


Рисунок 3.3 – Результат работы программы

ЗАКЛЮЧЕНИЕ

В ходе выполнения данной лабораторной работы по дисциплине «Операционные системы и системное программирование» была разработана программа для асинхронной обработки данных из файла. Основная задача состояла в чтении, сортировке и записи данных с использованием асинхронных операций.

В процессе выполнения работы были изучены и применены механизмы асинхронного ввода-вывода на платформе Windows. Дополнительно была использована технология пула потоков для распределения задач, что продемонстрировало возможности эффективного использования многопоточности в приложениях.

Успешное выполнение задачи по реализации асинхронного чтения и записи, а также сортировки данных внутри программы, продемонстрировало понимание принципов работы асинхронных операций и многопоточных приложений.

Лабораторная работа не только углубила теоретические знания о механизмах ввода-вывода в Windows, но и предоставила ценные практические навыки, которые являются основой для разработки высокопроизводительных приложений в среде многозадачности.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Таненбаум Э. С. Современные операционные системы. — 4-е изд. — СПб.: Питер, 2018. — 1120 с.

[2] Сильбершац А., Галвин П., Гейдж Г. Операционные системы: принципы и практика. — 9-е изд. — М.: Вильямс, 2015. — 1056 с.

ПРИЛОЖЕНИЕ А

(обязательное)

**Исходный код программы**

#include <windows.h>

#include <iostream>

#include <vector>

#include <string>

#include <algorithm>

#include <chrono>

#include "thread\_pool.h"

void processData(std::string& data) {

std::sort(data.begin(), data.end()); // Сортировка символов

}

void asyncReadWriteImproved(HANDLE hInputFile, HANDLE hOutputFile, DWORD bufferSize,

int numOperations, DWORD numberStr, DWORD sizeStr)

{

std::vector<OVERLAPPED> overlapped(numOperations);

std::vector<char\*> buffers(numOperations, nullptr);

std::vector<HANDLE> events(numOperations);

DWORD fileSize = GetFileSize(hInputFile, NULL);

DWORD totalBytesProcessed = 0;

for (int i = 0; i < numOperations; ++i) {

buffers[i] = new char[bufferSize + 1] {};

events[i] = CreateEvent(NULL, TRUE, FALSE, NULL);

if (!events[i]) {

std::cerr << "Failed to create event: " << GetLastError() << std::endl;

return;

}

}

DWORD startOffset = (numberStr - 1) \* (sizeStr + 2);

for (int i = 0; i < numOperations && totalBytesProcessed < fileSize; ++i) {

ZeroMemory(&overlapped[i], sizeof(OVERLAPPED));

overlapped[i].Offset = startOffset + totalBytesProcessed;

overlapped[i].hEvent = events[i];

DWORD bytesRead = 0;

if (!ReadFile(hInputFile, buffers[i], bufferSize, NULL, &overlapped[i])) {

DWORD error = GetLastError();

if (error != ERROR\_IO\_PENDING) {

std::cerr << "ReadFile failed: " << error << std::endl;

break;

}

}

totalBytesProcessed += bufferSize + 2;

}

for (int completedCount = 0; completedCount < numOperations; ++completedCount) {

// Ожидание завершения операций

DWORD waitResult = WaitForMultipleObjects(numOperations, events.data(), FALSE, INFINITE);

int completedIndex = waitResult - WAIT\_OBJECT\_0;

DWORD bytesTransferred = 0;

if (!GetOverlappedResult(hInputFile, &overlapped[completedIndex], &bytesTransferred, FALSE)) {

std::cerr << "GetOverlappedResult failed: " << GetLastError() << std::endl;

break;

}

// Обработка данных

std::string data(buffers[completedIndex], bytesTransferred);

processData(data);

data += "\r\n";

if (!WriteFile(hOutputFile, data.c\_str(), bytesTransferred + 2, NULL, &overlapped[completedIndex])) {

DWORD error = GetLastError();

if (error != ERROR\_IO\_PENDING) {

std::cerr << "WriteFile failed: " << error << std::endl;

break;

}

}

if (!GetOverlappedResult(hOutputFile, &overlapped[completedIndex], &bytesTransferred, TRUE)) {

std::cerr << "GetOverlappedResult (write) failed: " << GetLastError() << std::endl;

break;

}

// Сброс события

ResetEvent(events[completedIndex]);

}

for (int i = 0; i < numOperations; ++i) {

delete[] buffers[i];

CloseHandle(events[i]);

}

}

int main() {

const std::wstring inputFileName = L"input.dat";

const std::wstring outputFileName = L"output.dat";

const DWORD sizeStr = 256;

const DWORD bufferSize = sizeStr;

ThreadPool tpool(5);

// Открытие файлов

HANDLE hInputFile = CreateFile(inputFileName.c\_str(), GENERIC\_READ, 0, NULL, OPEN\_EXISTING, FILE\_FLAG\_OVERLAPPED, NULL);

HANDLE hOutputFile = CreateFile(outputFileName.c\_str(), GENERIC\_WRITE, 0, NULL, CREATE\_ALWAYS, FILE\_FLAG\_OVERLAPPED, NULL);

if (hInputFile == INVALID\_HANDLE\_VALUE || hOutputFile == INVALID\_HANDLE\_VALUE) {

std::cerr << "Failed to open files: " << GetLastError() << std::endl;

return 1;

}

DWORD fileSize = GetFileSize(hInputFile, NULL);

DWORD countStr = fileSize / (sizeStr + 2);

auto start = std::chrono::high\_resolution\_clock::now();

tpool.start();

// здесь задачи добавляются в пул потоков

for (int i = 1; i <= countStr / 2; ++i) {

tpool.push\_task(asyncReadWriteImproved, hInputFile, hOutputFile, bufferSize, 2, 2\*i - 1, sizeStr);

}

if (countStr % 2 != 0) {

tpool.push\_task(asyncReadWriteImproved, hInputFile, hOutputFile, bufferSize, 1, countStr, sizeStr);

}

tpool.stop();

auto end = std::chrono::high\_resolution\_clock::now();

std::cout << "Processing time: "

<< std::chrono::duration\_cast<std::chrono::milliseconds>(end - start).count()

<< " ms" << std::endl;

CloseHandle(hInputFile);

CloseHandle(hOutputFile);

return 0;

}