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 Шпаковский А.В

Проверил: ассистент кафедры информатики Гриценко Н.Ю.

Минск 2024

СОДЕРЖАНИЕ

[1 Цель работы 3](#_Toc181205452)

[2 Теоритические сведения 4](#_Toc181205453)

[2.1 Определение файлов 4](#_Toc181205454)

[2.2 Управление файлами 4](#_Toc181205455)

[2.3 Асинхронные операции ввода-вывода 5](#_Toc181205456)

[3 Инструментальная языковая среда 6](#_Toc181205457)

[4 Результат запуска программы 7](#_Toc181205458)

[4.1 Иллюстрация запуска и работы программы 7](#_Toc181205459)

[Заключение 9](#_Toc181205460)

[Список использованных источников 10](#_Toc181205461)

[Приложение А (обязательное) Исходный код программного продукта 11](#_Toc181205462)

# 1 ЦЕЛЬ РАБОТЫ

Приложение, разработанное в рамках данной лабораторной работы, представляет собой систему асинхронной обработки файлов для операционной системы Windows, написанную на языке программирования C++ с использованием Windows API. Основной функционал программы заключается в чтении данных из входного текстового файла, их обработке (сортировке) и записи результатов в выходной файл.

Для выполнения этих задач используются функции API Windows, такие как CreateFile для открытия файлов, ReadFile и WriteFile для чтения и записи данных, а также управление событиями через структуру OVERLAPPED. Программа создает дескрипторы для входного и выходного файлов, что позволяет эффективно управлять асинхронными операциями ввода-вывода.

Программа начинается с открытия входного файла в режиме чтения с флагом FILE\_FLAG\_OVERLAPPED, что позволяет использовать асинхронный ввод-вывод. Входные данные считываются порциями размером BUFFER\_SIZE (4096 байт), которые затем обрабатываются с помощью функции ProcessData, сортирующей данные в буфере. После обработки программа записывает отсортированные данные в выходной файл, также используя асинхронный подход.

Визуализация информации о процессе работы осуществляется через консольные выводы, информирующие пользователя о количестве обработанных байтов и времени выполнения операций. Программа также учитывает возможность ожидания завершения асинхронных операций, что обеспечивает корректное завершение всех процессов перед выходом.

Для разработки приложения использовалась среда Microsoft Visual Studio 2022, которая предоставила все необходимые инструменты для написания, компиляции и отладки программы. Использование Visual Studio упростило управление проектом, позволяя сосредоточиться на разработке функционала и тестировании приложения.

Отчет по лабораторной работе содержит полное описание реализации программы, её структуры и возможностей, а также особенности использования языка C++ и Windows API для работы с файлами и асинхронными операциями ввода-вывода.

# 2 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

## 2.1 Определение файлов

Файл — это структурированная единица данных, сохраняемая на носителе информации, такой как жесткий диск или SSD. Файлы могут иметь различные форматы и содержать различные типы информации: текст, изображения, видео и другие. Каждому файлу присваивается уникальное имя и расширение, что позволяет операционной системе идентифицировать его тип и определять, какие программы могут его открыть. Операционная система управляет файлами, предоставляя интерфейсы для их открытия, чтения, записи и удаления.

## 2.2 Управление файлами

В Windows управление файлами осуществляется через API, такие как CreateFile, ReadFile, и WriteFile. CreateFile — это наиболее распространенная функция для открытия файлов, томов дисков, анонимных каналов и других аналогичных устройств, операции ввода-вывода также можно выполнять с помощью дескриптора типа из других системных объектов, таких как сокет или прием функций. Когда файл был открыт для асинхронного ввода-вывода, указатель на структуру OVERLAPPED передается в вызов ReadFile и WriteFile.[1] Эти функции позволяют разработчикам открывать файлы, читать из них и записывать данные обратно. Для асинхронной работы с файлами используется флаг FILE\_FLAG\_OVERLAPPED, что позволяет выполнять операции ввода-вывода без блокировки интерфейса приложения. Это особенно важно для программ, работающих с большими объемами данных, поскольку оно позволяет пользователю продолжать взаимодействие с приложением во время выполнения операций.

Корректное управление файлами включает обработку ошибок, таких как проверка возвращаемых значений функций и корректное закрытие дескрипторов файлов. Файловые дескрипторы закрываются с помощью функции CloseHandle.[2]

## 2.3 Асинхронные операции ввода-вывода

Асинхронные операции ввода-вывода представляют собой мощный механизм, позволяющий выполнять операции чтения и записи данных без блокировки основного потока выполнения программы. Это значит, что во время выполнения таких операций код может продолжать исполняться, что значительно увеличивает общую эффективность и отзывчивость приложений.

Одним из ключевых аспектов асинхронных операций является использование специализированных структур, таких как OVERLAPPED. Эти структуры позволяют системе отслеживать состояние выполнения операций ввода-вывода, что делает возможным обработку данных в фоновом режиме. В результате, приложения могут продолжать выполнять другие задачи, например, обрабатывать пользовательский ввод или выполнять вычисления, не дожидаясь завершения операций с файлами.

Применение асинхронных методов, таких как ReadFile и WriteFile, особенно полезно при работе с большими объемами данных. Эти методы позволяют избежать значительных задержек, которые могут возникнуть при синхронных операциях, когда программа простаивает, ожидая завершения ввода-вывода. Эффективное использование асинхронных операций способствует улучшению пользовательского опыта, так как приложение может предоставлять результаты быстрее и реагировать на действия пользователя без заметных задержек.

Тем не менее, при использовании асинхронных операций важно учитывать необходимость синхронизации. Параллельный доступ к общим ресурсам может привести к конфликтам и непредсказуемым результатам. Для предотвращения таких проблем разработчики могут использовать различные механизмы синхронизации, такие как события, мьютексы или семафоры. Эти инструменты помогают координировать доступ к ресурсам и обеспечивают целостность данных.

# 3 ИНСТРУМЕНТАЛЬНАЯ ЯЗЫКОВАЯ СРЕДА

Для разработки программы, демонстрирующей управление файлами с использованием асинхронных операций и отображения данных в память, был выбран язык программирования C++. Это высокопроизводительный и мощный язык, который позволяет напрямую работать с системными ресурсами и Windows API, что особенно важно для эффективной обработки файлов. C++ широко применяется при разработке приложений, требующих высокой скорости и детального контроля над ресурсами операционной системы, таких как программное обеспечение для обработки данных, системные утилиты и драйверы.

В качестве интегрированной среды разработки использовалась Microsoft Visual Studio 2022. Эта IDE предоставляет удобные и мощные инструменты для написания, отладки и тестирования кода, поддерживает возможности Windows API и обеспечивает высокую производительность приложений. Разработка и тестирование проводились в среде Windows 10, что позволило задействовать нативные возможности асинхронного и неблокирующего ввода-вывода, включая функции CreateFile, ReadFile, WriteFile. Такой подход к разработке обеспечил не только максимальную производительность, но и удобство, особенно при разработке на мобильных устройствах, таких как ноутбук.

# 4 РЕЗУЛЬТАТ ЗАПУСКА ПРОГРАММЫ

## 4.1 Иллюстрация запуска и работы программы

На рисунке 4.1 мы наблюдаем консоль с результатом выполнения программы.

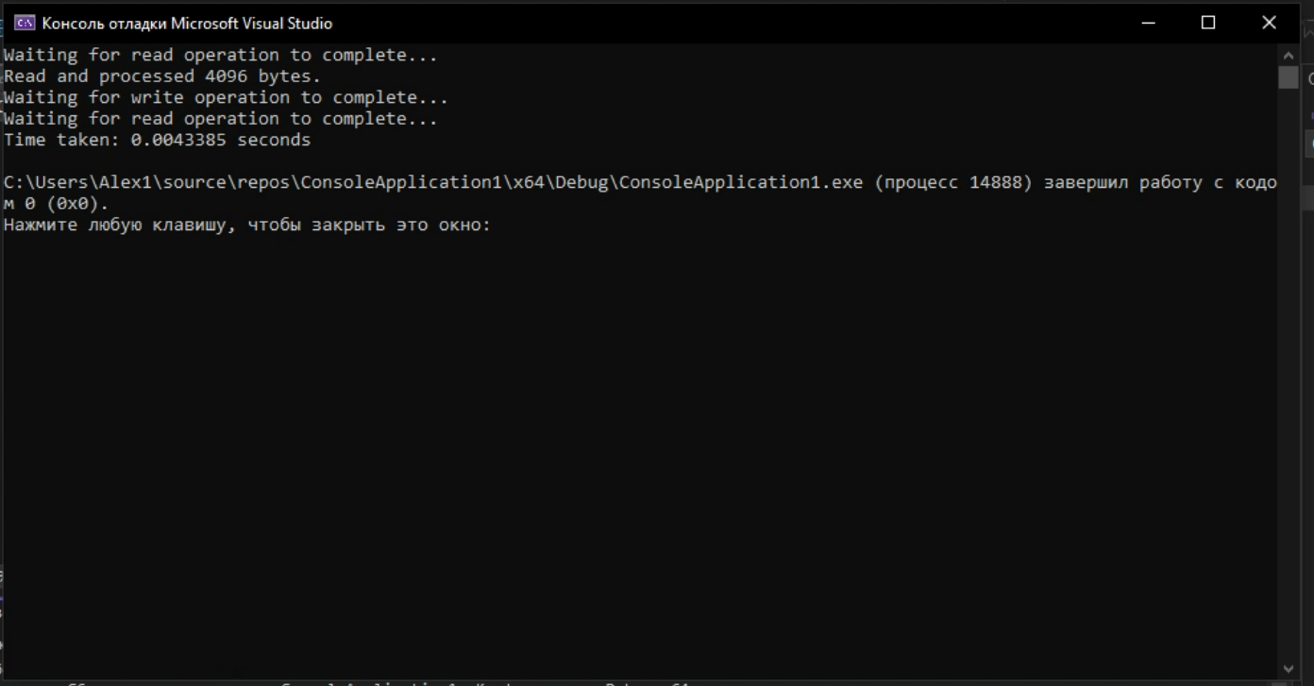


Рисунок 4.1 — Консоль с результатом программы

На рисунках 4.2 и 4.3 изображено содержание входного и выходного файлов соответственно.

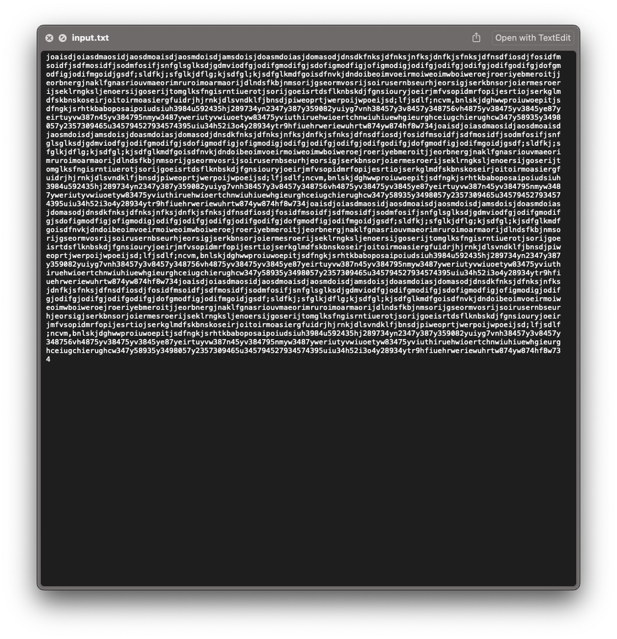


Рисунок 4.2 — Содержание входного файла

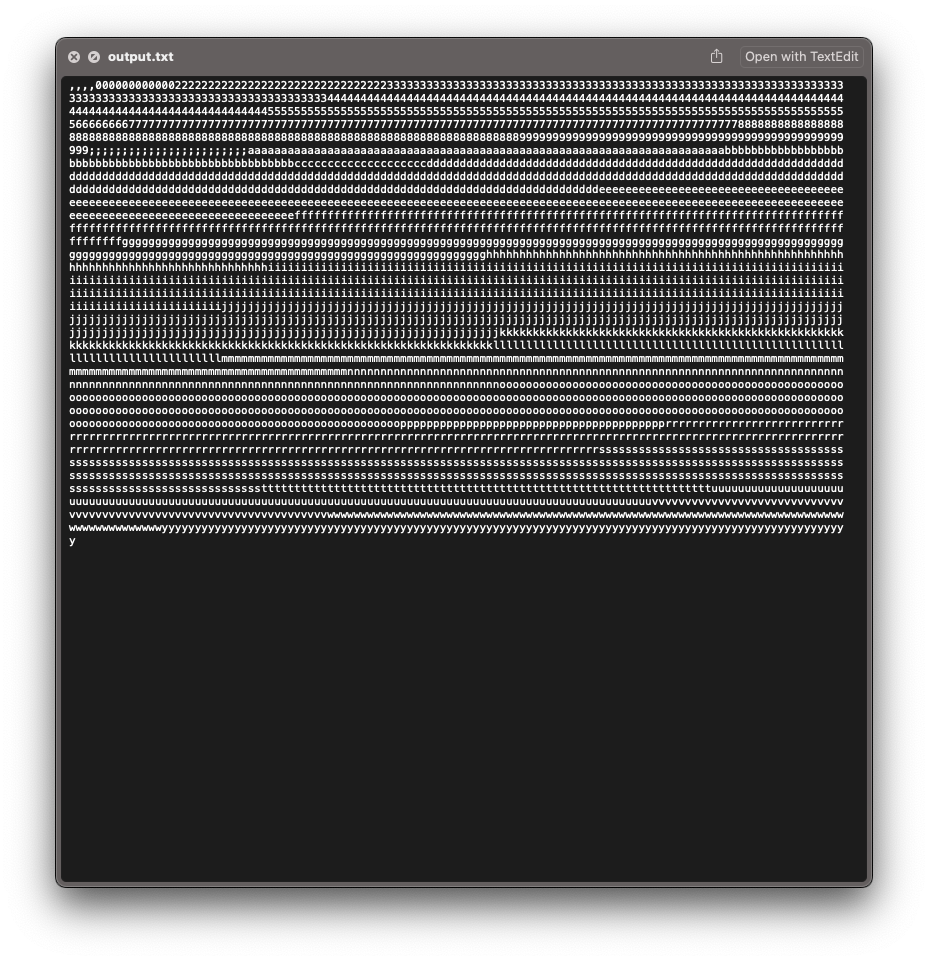


Рисунок 4.3 — Содержание выходного файла

Входной файл содержит 4096 символов, которые будут подвергнуты сортировке после выполнения программы. В процессе работы программа считывает данные из файла, обрабатывает их и сортирует в определенном порядке. После завершения сортировки, отсортированные символы будут записаны в отдельный выходной файл, который можно будет использовать для дальнейшей работы или анализа. Этот процесс позволяет упорядочить данные, что делает их более удобными для восприятия и последующей обработки.

# ЗАКЛЮЧЕНИЕ

В ходе выполнения данной лабораторной работы по курсу “Операционные системы и системное программирование” была создана программа, позволяющая продемонстрировать различные методы работы с файлами, включая асинхронный ввод-вывод и отображение файлов в память. Основная задача программы заключалась в реализации операций асинхронного чтения и записи данных, а также в сравнении их эффективности с синхронным методом.

В реализации программы использовались функции Windows API, такие как CreateFile для создания и открытия файлов, ReadFile и WriteFile для асинхронного ввода-вывода. Эти методы обеспечивают управление чтением и записью данных без блокировки основного потока выполнения программы, что значительно ускоряет обработку больших объёмов данных.

В процессе работы особое внимание уделялось корректному завершению операций и освобождению ресурсов, включая правильное закрытие файловых дескрипторов, завершение асинхронных операций и очистку отображённых в память областей. Программа показала, насколько эффективными могут быть асинхронные методы для работы с большими файлами, а также позволила углубить знания по асинхронному вводу-выводу и работе с системными ресурсами Windows.

Данная лабораторная работа дала практический опыт использования асинхронных функций для работы с файлами, продемонстрировала преимущества и особенности таких операций, а также позволила закрепить навыки разработки эффективного кода для Windows с использованием Windows API.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Синхронный и асинхронный ввод-вывод - Win32 apps - Microsoft Learn – Электронный ресурс. – Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/fileio/synchronous-and-asynchronous-i-o

[2] Учимся работать с файлами через Windows API | Записки программиста – Электронный ресурс. – Режим доступа: https://eax.me/winapi-files

# ПРИЛОЖЕНИЕ А (обязательное) Исходный код программного продукта

#include <windows.h>

#include <iostream>

#include <vector>

#include <chrono>

#include <algorithm>

#define BUFFER\_SIZE 4096

void ProcessData(char\* buffer, DWORD bytesRead) {

std::sort(buffer, buffer + bytesRead);

}

void AsyncFileProcessing(LPCWSTR inputFilePath, LPCWSTR outputFilePath) {

HANDLE hInputFile = CreateFile(inputFilePath, GENERIC\_READ, 0, NULL, OPEN\_EXISTING, FILE\_FLAG\_OVERLAPPED, NULL);

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

std::cerr << "Error opening input file: " << GetLastError() << std::endl;

return;

}

HANDLE hOutputFile = CreateFile(outputFilePath, GENERIC\_WRITE, 0, NULL, CREATE\_ALWAYS, FILE\_FLAG\_OVERLAPPED, NULL);

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

std::cerr << "Error creating output file: " << GetLastError() << std::endl;

CloseHandle(hInputFile);

return;

}

OVERLAPPED readOverlapped = { 0 };

OVERLAPPED writeOverlapped = { 0 };

readOverlapped.hEvent = CreateEvent(NULL, TRUE, FALSE, NULL);

writeOverlapped.hEvent = CreateEvent(NULL, TRUE, FALSE, NULL);

if (readOverlapped.hEvent == NULL || writeOverlapped.hEvent == NULL) {

std::cerr << "Error creating event: " << GetLastError() << std::endl;

CloseHandle(hInputFile);

CloseHandle(hOutputFile);

return;

}

DWORD bytesRead = 0, bytesWritten = 0;

char buffer[BUFFER\_SIZE];

DWORD totalBytesRead = 0;

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

while (true) {

readOverlapped.Offset = totalBytesRead;

readOverlapped.OffsetHigh = 0;

BOOL readResult = ReadFile(hInputFile, buffer, BUFFER\_SIZE, &bytesRead, &readOverlapped);

if (!readResult) {

if (GetLastError() == ERROR\_IO\_PENDING) {

std::cout << "Waiting for read operation to complete...\n";

WaitForSingleObject(readOverlapped.hEvent, INFINITE);

GetOverlappedResult(hInputFile, &readOverlapped, &bytesRead, FALSE);

}

else {

std::cerr << "Read error: " << GetLastError() << std::endl;

break;

}

}

if (bytesRead == 0) break;

totalBytesRead += bytesRead;

ProcessData(buffer, bytesRead);

std::cout << "Read and processed " << bytesRead << " bytes.\n";

writeOverlapped.Offset = totalBytesRead - bytesRead;

writeOverlapped.OffsetHigh = 0;

BOOL writeResult = WriteFile(hOutputFile, buffer, bytesRead, &bytesWritten, &writeOverlapped);

if (!writeResult) {

if (GetLastError() == ERROR\_IO\_PENDING) {

std::cout << "Waiting for write operation to complete...\n";

WaitForSingleObject(writeOverlapped.hEvent, INFINITE);

GetOverlappedResult(hOutputFile, &writeOverlapped, &bytesWritten, FALSE);

}

else {

std::cerr << "Write error: " << GetLastError() << std::endl;

break;

}

}

}

if (bytesWritten > 0) {

WaitForSingleObject(writeOverlapped.hEvent, INFINITE);

GetOverlappedResult(hOutputFile, &writeOverlapped, &bytesWritten, FALSE);

}

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

std::chrono::duration<double> elapsed = end - start;

std::cout << "Time taken: " << elapsed.count() << " seconds\n";

CloseHandle(hInputFile);

CloseHandle(hOutputFile);

}

int main() {

LPCWSTR inputFilePath = L"C:/Users/Alex1/source/repos/ConsoleApplication2/ConsoleApplication2/large\_text\_file.txt";

LPCWSTR outputFilePath = L"C:/Users/Alex1/source/repos/ConsoleApplication2/ConsoleApplication2/processed\_text\_file.txt";

AsyncFileProcessing(inputFilePath, outputFilePath);

return 0;

}