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 Косяков М.М.

Проверил: ассистент кафедры информатики Гриценко Н.Ю.

Минск 2024

**СОДЕРЖАНИЕ**

[1 Формулировка задачи 3](#_Toc178093096)

[2 Краткие теоритические сведения 4](#_Toc178093097)

[3 Описание функций программы 5](#_Toc178093098)

[3.1 Выбор файла для чтения 5](#_Toc178093099)

[3.2 Обработка данных файла и анализ временных затрат 5](#_Toc178093100)

[Заключение 6](#_Toc178093101)

[Список использованных источников 7](#_Toc178093102)

[Приложение А (обязательное) Исходный код программы 8](#_Toc178093103)

1 ФОРМУЛИРОВКА ЗАДАЧИ

В рамках этой лабораторной работы ставлю перед собой цель углубить и закрепить навыки работы с файловой системой и механизмами организации ввода-вывода в операционной системе Windows. В ходе выполнения работы необходимо изучить расширенные возможности работы с файлами, включая неблокирующие и асинхронные операции, мультиплексирование ввода-вывода, отображение файлов в память и другие ключевые аспекты. Особое внимание будет уделено подсистеме ввода-вывода, различным видам ввода-вывода и работе с логическими устройствами. Важно освоить концепции файловой системы: иерархия, виды файлов, работа с файлами и директориями (папками). Также предстоит изучить работу с открытыми файлами (устройствами) как объектами, доступ к ним через файловые дескрипторы (Handle), и использование основного набора API-функций для работы с файлами.

Для выполнения лабораторной работы по обработке содержимого файла с использованием асинхронных операций необходимо реализовать приложение, которое будет работать с асинхронным вводом-выводом (чтение и запись данных) параллельно с их обработкой в памяти. Программа будет написана на языке C с использованием WinAPI.

В качестве задачи необходимо выполнить многопоточную обработку файла, а для этого:

– выбрать файла для чтения;

– реализовать асинхронное чтение данных с возможностью варьирования количества параллельно инициированных операций ввода-вывода;

– выполнять обработку данных в памяти по мере их получения;

– подсчитать время, которое понадобилось на его обработку;

В результате выполнения этой лабораторной работы будут не только получены теоретические знания работы с асинхронной обработкой файла, но и практический опыт в разработке приложений, которые могут эффективно использовать ресурсы компьютера. Понимание концепций файловой системы и механизмов организации ввода-вывода, станет необходимым для создания высокопроизводительных программных решений.

2 КРАТКИЕ ТЕОРИТИЧЕСКИЕ СВЕДЕНИЯ

Подсистема ввода-вывода обеспечивает связь вычислительной системы с внешними устройствами [1]. В современных системах многие из таких устройств бывают представлены на логическом уровне в виде файлов, поэтому понятия «управление вводом-выводом» и «управление файлами» в значительной мере синонимичны.

Файл — набор данных (обычно подразумеваются данные на внешнем носителе), пригодный для использования прикладными программами в вычислительной системе. Удобно представлять файл как совокупность данных (используются прикладными программами) и метаданных (используются системными программами). Файловая система — способ организации данных на внешних носителях [2]. Иерархическая файловая система имеет корневую директорию, содержащую файлы и поддиректории. В Microsoft каждый логический диск имел свое дерево директорий, начиная с буквы диска. Win NT ввел общий корневой узел, объединяющий логические диски как директории, но не все программы его отображают явно.

Идентификатор файла — его имя в файловой системе. Полное имя файла состоит из имени файла и пути к нему (включая вышестоящие директории). Путь может быть абсолютным (начиная с корневого каталога) или относительным (начиная с текущего каталога). Имя должно быть уникальным в текущем каталоге, а полное имя — во всей файловой системе. Глубина иерархии обычно не ограничена, но общая длина полных имен имеет ограничения.

Модели ввода-вывода можно классифицировать по различным критериям, каждая из которых имеет свои особенности и преимущества.

Блокирующий ввод-вывод предполагает, что поток останавливается и ждет завершения транзакции. Неблокирующий ввод-вывод позволяет выполнять предварительную проверку состояния устройства.

Мультиплексированный ввод-вывод подразумевает, что программа анализирует состояния нескольких устройств одновременно и инициирует транзакции только с теми устройствами, которые готовы к обмену данными. Это обеспечивает параллельный обмен и улучшает общую производительность системы. В многопоточной реализации блокирующие операции могут выполняться в отдельных потоках. Это позволяет избежать блокировок основного потока и повышает отзывчивость программы.

Асинхронный ввод-вывод использует системные механизмы для организации параллельных транзакций и потоков. Это обеспечивает эффективное управление ресурсами. Наконец, отображение файлов в память позволяет проецировать файлы в память, обеспечивая прямой доступ к их содержимому.

3 ОПИСАНИЕ ФУНКЦИЙ ПРОГРАММЫ

Согласно формулировке задачи, были спроектированы следующие функции программы [3]:

– выбор файла для запуска (с обработкой ошибок при выборе некорректного файла);

– подсчёт времени для считывания файла;

– завершение работы программы;

**3.1 Выбор файла для чтения**

Для начала обработки следует выбрать файл (рисунок 3.1).

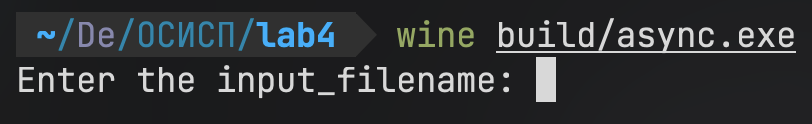


Рисунок 3.1– Выбор файла для обработки

При попытке открыть некорректный файл появится всплывающее окно с сообщением об ошибке (рисунок 3.2).

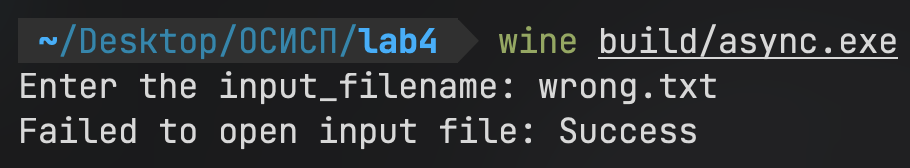


Рисунок 3.2 – Сообщение об ошибке при открытии некорректного файла

**3.2 Обработка данных файла и анализ временных затрат**

Программа использует асинхронные операции чтения и записи для обработки данных из файла в параллельных потоках. Она разделяет файл на сегменты и выполняет обработку для каждого сегмента после чтения, а затем записывает обработанные данные в новый файл. Программа замеряет время выполнения операций чтения, обработки и записи, обеспечивая эффективную обработку данных. (рисунок 3.3).

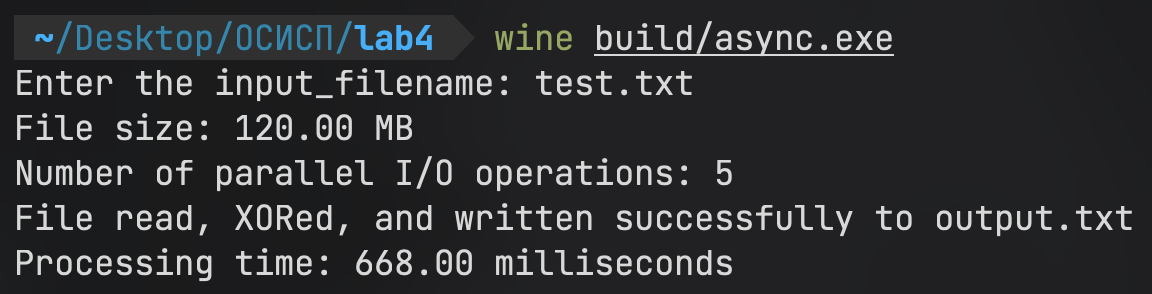


Рисунок 3.3 – Сообщение об обработке файла

ЗАКЛЮЧЕНИЕ

В ходе выполнения данной лабораторной работы были изучены и закреплены навыки программирования приложений для операционной системы Windows с использованием асинхронных операций. Мы погрузились в концепции работы с файлами, и научились их реализации в среде Windows с помощью WinAPI.

Основными аспектами, которыми мы ознакомились, стали операции с файловой системой, подходы к организации ввода-вывода. Мы также освоили принципы многопоточной, асинхронной и других моделей ввода-вывода, что сыграет важную роль в разработке производительных приложений.

В ходе работы над асинхронной обработкой файла мы выбрали файл для чтения, разделили файл на сегменты и выполнили обработку для каждого сегмента после чтения, а затем записали обработанные данные в новый файл. Затем замерили время выполнения операций.

Эта лабораторная работа не только позволила углубить наши теоретические знания в области асинхронных операций, но и предоставила ценный практический опыт в разработке эффективных приложений, способных эффективно использовать ресурсы компьютера. Понимание концепций ввода-вывода и файловых систем, а также навыки их управления, стали ключевыми в создании высокопроизводительных программных решений.

Выполнение этой лабораторной работы позволило нам не только расширить нашу базу знаний, но и приобрести практические навыки, которые будут полезны в дальнейшей разработке программного обеспечения для Windows.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Build desktop Windows apps using the Win32 API [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/en-us/windows/win32/

[2] Основные сообщения ОС Windows (Win32 API). Программирование в ОС Windows. Лекция 1. – Режим доступа: https://www.youtube.com/watch?v=wTArIolxch0

[3] Разработка приложений с помощью WinAPI. – Режим доступа: https://shorturl.at/BDJW8

ПРИЛОЖЕНИЕ А

(обязательное)

**Исходный код программы**

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <windows.h>

#define MAX\_THREADS 5 // The number of parallel operations

// Structure to hold data for asynchronous I/O operations

struct AsyncIOData

{

OVERLAPPED overlapped; // Structure for asynchronous I/O

HANDLE file\_handle; // Handle for the file being read/written

char \*buffer; // Buffer to store data

size\_t size; // Size of the data to read/write

size\_t offset; // Offset for reading/writing

};

// Thread function for reading from a file

DWORD WINAPI ReadFileThread(LPVOID param)

{

AsyncIOData \*io\_data = (AsyncIOData \*)param; // Cast parameter to AsyncIOData

DWORD bytes\_read;

// Set the offset for the read operation

io\_data->overlapped.Offset = (DWORD)io\_data->offset;

io\_data->overlapped.OffsetHigh = (DWORD)(io\_data->offset >> 32);

// Start the read operation

if (!ReadFile(io\_data->file\_handle, io\_data->buffer, io\_data->size, &bytes\_read, &io\_data->overlapped))

{

// Check if the read operation is pending

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

{

perror("Failed to initiate read");

return 1;

}

}

// Wait for the read operation to complete

GetOverlappedResult(io\_data->file\_handle, &io\_data->overlapped, &bytes\_read, TRUE);

return bytes\_read; // Return number of bytes read

}

// Thread function for writing to a file

DWORD WINAPI WriteFileThread(LPVOID param)

{

AsyncIOData \*io\_data = (AsyncIOData \*)param; // Cast parameter to AsyncIOData

DWORD bytes\_written;

// Set the offset for the write operation

io\_data->overlapped.Offset = (DWORD)io\_data->offset;

io\_data->overlapped.OffsetHigh = (DWORD)(io\_data->offset >> 32);

// Start the write operation

if (!WriteFile(io\_data->file\_handle, io\_data->buffer, io\_data->size, &bytes\_written, &io\_data->overlapped))

{

// Check if the write operation is pending

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

{

perror("Failed to initiate write");

return 1;

}

}

// Wait for the write operation to complete

GetOverlappedResult(io\_data->file\_handle, &io\_data->overlapped, &bytes\_written, TRUE);

return bytes\_written; // Return number of bytes written

}

// Function to XOR the contents of a buffer

void XorBuffer(char \*buffer, size\_t size)

{

for (size\_t i = 0; i < size; ++i)

{

buffer[i] ^= 0xFF;

buffer[i] ^= 0xFF;

}

}

int main()

{

char input\_filename[] = "test.txt"; // Input file name

char output\_filename[] = "output.txt"; // Output file name

printf("Enter the input\_filename: ");

scanf("%255s", input\_filename);

// Open input file for reading

HANDLE file\_handle = CreateFile(input\_filename, GENERIC\_READ, FILE\_SHARE\_READ, NULL, OPEN\_EXISTING, FILE\_FLAG\_OVERLAPPED, NULL);

if (file\_handle == INVALID\_HANDLE\_VALUE)

{

perror("Failed to open input file");

return EXIT\_FAILURE;

}

LARGE\_INTEGER file\_size;

// Get the size of the input file

if (!GetFileSizeEx(file\_handle, &file\_size))

{

CloseHandle(file\_handle);

perror("Failed to get file size");

return EXIT\_FAILURE;

}

printf("File size: %.2f MB\n", (double)file\_size.QuadPart / (1024 \* 1024));

printf("Number of parallel I/O operations: %d\n", MAX\_THREADS);

size\_t chunk\_size = file\_size.QuadPart / MAX\_THREADS; // Calculate chunk size for each thread

HANDLE read\_threads[MAX\_THREADS]; // Array to hold read thread handles

HANDLE write\_threads[MAX\_THREADS]; // Array to hold write thread handles

AsyncIOData read\_data[MAX\_THREADS]; // Array to hold read data structures

AsyncIOData write\_data[MAX\_THREADS]; // Array to hold write data structures

char \*buffers[MAX\_THREADS]; // Array of buffers for each thread

// Start time for processing

clock\_t start\_time = clock();

// Read operations in parallel

for (int i = 0; i < MAX\_THREADS; ++i)

{

buffers[i] = (char \*)malloc(chunk\_size); // Allocate memory for buffer

if (!buffers[i])

{

perror("Failed to allocate memory for buffer");

return EXIT\_FAILURE;

}

// Initialize read data for each thread

read\_data[i].overlapped.hEvent = CreateEvent(NULL, TRUE, FALSE, NULL);

read\_data[i].file\_handle = file\_handle;

read\_data[i].buffer = buffers[i];

read\_data[i].size = chunk\_size;

read\_data[i].offset = i \* chunk\_size;

// Create a thread for reading

read\_threads[i] = CreateThread(NULL, 0, ReadFileThread, &read\_data[i], 0, NULL);

if (!read\_threads[i])

{

perror("Failed to create read thread");

return EXIT\_FAILURE;

}

}

// Wait for all read operations to complete

WaitForMultipleObjects(MAX\_THREADS, read\_threads, TRUE, INFINITE);

for (int i = 0; i < MAX\_THREADS; ++i)

{

CloseHandle(read\_threads[i]); // Close thread handles

}

// Perform XOR operation on read buffers

for (int i = 0; i < MAX\_THREADS; ++i)

{

XorBuffer(buffers[i], chunk\_size); // Modify the buffer

}

// Create output file handle for writing

HANDLE output\_handle = CreateFile(output\_filename, GENERIC\_WRITE, 0, NULL, CREATE\_ALWAYS, FILE\_FLAG\_OVERLAPPED, NULL);

if (output\_handle == INVALID\_HANDLE\_VALUE)

{

CloseHandle(file\_handle);

perror("Failed to open output file");

return EXIT\_FAILURE;

}

// Write operations in parallel

for (int i = 0; i < MAX\_THREADS; ++i)

{

// Initialize write data for each thread

write\_data[i].overlapped.hEvent = CreateEvent(NULL, TRUE, FALSE, NULL);

write\_data[i].file\_handle = output\_handle;

write\_data[i].buffer = buffers[i];

write\_data[i].size = chunk\_size;

write\_data[i].offset = i \* chunk\_size;

// Create a thread for writing

write\_threads[i] = CreateThread(NULL, 0, WriteFileThread, &write\_data[i], 0, NULL);

if (!write\_threads[i])

{

perror("Failed to create write thread");

return EXIT\_FAILURE;

}

}

// Wait for all write operations to complete

WaitForMultipleObjects(MAX\_THREADS, write\_threads, TRUE, INFINITE);

for (int i = 0; i < MAX\_THREADS; ++i)

{

CloseHandle(write\_threads[i]); // Close thread handles

free(buffers[i]); // Free allocated buffers

}

printf("File read, XORed, and written successfully to %s\n", output\_filename);

// End time for processing

clock\_t end\_time = clock();

double elapsed\_time = (double)(end\_time - start\_time) / CLOCKS\_PER\_SEC \* 1000;

printf("Processing time: %.2f milliseconds\n", elapsed\_time);

// Cleanup

CloseHandle(file\_handle);

CloseHandle(output\_handle);

return EXIT\_SUCCESS;

}