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.253504 Божко Я.Д

Проверил: ассистент кафедры информатики

Гриценко Н.Ю

Минск 2024

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 3](#_Toc20629)

[1 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ 4](#_Toc22457)

[1.1 Подсистема ввода-вывода и файловая система 4](#_Toc10276)

[1.1.1 Подсистема ввода-вывода (I/O) 4](#_Toc31949)

[1.1.2 Основные функции подсистемы Ввод/Вывод: 4](#_Toc1730)

[1.1.3 Разновидности ввода-вывода 4](#_Toc1027)

[1.1.4 Логические устройства 5](#_Toc25106)

[1.2 Файл и файловая система 5](#_Toc4324)

[1.2.1 Файл и каталог как объект: 6](#_Toc2589)

[1.3 API для работы с файлами 6](#_Toc11093)

[1.4 Структура OVERLAPPED и функции ожидания 7](#_Toc1843)

[2 ИНСТРУМЕНТАЛЬНАЯ ЯЗЫКОВАЯ СРЕДА 8](#_Toc17019)

[3 РАБОТА ПРИЛОЖЕНИЯ 9](#_Toc15277)

[1.1 Структура приложения 9](#_Toc18410)

[1.2 Функции приложения 9](#_Toc4326)

[1.3 Анализ данных 9](#_Toc11587)

[1.3.1 Поиск "узкого места" 9](#_Toc28910)

[1.3.2 Сравнение с традиционным подходом: 10](#_Toc8092)

[1.3.3 Сравнение с многопоточной реализацией: 10](#_Toc6836)

[1.3.4 Вывод 10](#_Toc30580)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 11](#_Toc3273)

[ПРИЛОЖЕНИЕ А 12](#_Toc18690)

**ВВЕДЕНИЕ**

Цель работы: приложение, демонстрирующее работу с файлами, в первую очередь с использованием асинхронных методов и отображения файла в память.

Оценка эффективности по сравнению с простейшим (синхронным) вводом- выводом (и, возможно, с многопоточной реализацией приложения при выполнении аналогичных операций над данными).Реализация обработки содержимого файла данных, используя асинхронный ввод-вывод (чтение/запись очередных порций данных параллельно с выполнением обработки данных в памяти).

Варьирование количества параллельно инициированных операций ввода- вывода, влияние их на общею производительность. Поиск «узкого места» в программной реализации.

Оценка эффективности (производительности) по сравнению с традиционным подходом (чтение  обработка  выгрузка).

Опционально – то же по сравнению с многопоточной реализацией.

Базовые операции чтения и записи в файлы, предоставляемые стандартными API (как, например, ReadFile и WriteFile в Windows), составляют основу взаимодействия с файловой системой. Однако для оптимизации работы с файлами, особенно в ресурсоёмких приложениях или при работе с большим количеством файлов, требуется использовать более продвинутые методы и механизмы.

**1 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

**1.1 Подсистема ввода-вывода и файловая система**

**1.1.1** Подсистема ввода-вывода (I/O)

Подсистема ввода-вывода (I/O) – это часть операционной системы, отвечающая за взаимодействие с внешними устройствами, такими как клавиатура, мышь, монитор, жесткий диск, принтер и т.д. Она предоставляет интерфейс для программ, позволяя им получать данные от устройств и передавать данные на устройства.

**1.1.2** Основные функции подсистемы Ввод/Вывод:

Управление внешними устройствами: Она обеспечивает взаимодействие с различными внешними устройствами, такими как клавиатура, мышь, монитор, принтер, сканер, сетевая карта и т.д.

Преобразование данных: она преобразует данные в формат, понятный компьютеру (цифровой), и обратно, из цифрового формата в формат, понятный внешнему устройству.

Определение приоритетов и управления ресурсами: Она управляет потоками данных, распределяет ресурсы между устройствами и определяет приоритет для обработки данных.

Управление прерываниями: она обрабатывает прерывания от внешних устройств, сигнализируя процессору о необходимости обработки данных от устройства.

**1.1.3** Разновидности ввода-вывода

Выделяют следующие группы:

1. Физический ввод-вывод: непосредственное взаимодействие с физическими устройствами.
2. Логический ввод-вывод: абстрактный уровень, который использует драйверы устройств для взаимодействия с физическими устройствами.
3. Программный ввод-вывод: использование программных интерфейсов (API) для управления устройствами и потоками данных.

По способу взаимодействия:

1. Прямой доступ к памяти (DMA): в этом случае устройство напрямую получает доступ к памяти, минуя процессор. Это позволяет увеличить скорость передачи данных.
2. Прерывания: устройство посылает сигнал процессору, чтобы тот начал обработку данных.
3. Опрос: процессор периодически проверяет состояние устройства, чтобы узнать, доступны ли данные для обработки.

По типу данных:

1. Аналоговый ввод/вывод: преобразование аналоговых сигналов (например, из датчиков) в цифровой формат и наоборот.
2. Цифровой ввод/вывод: обмен цифровыми данными (например, клавиатура, монитор).

По направлению передачи данных:

1. Ввод: передача данных от внешнего устройства к компьютеру.
2. Вывод: передача данных от компьютера к внешнему устройству.

По скорости передачи данных:

1. Высокоскоростной: передача данных на большой скорости (например, видеокарта, жесткий диск).
2. Низкоскоростной: передача данных на медленной скорости (например, клавиатура, мышь).

**1.1.4** Логические устройства

Логические устройства – это абстракции физических устройств, которые используются операционной системой для управления вводом-выводом. Например, вместо того, чтобы работать с конкретным жестким диском, операционная система работает с логическим устройством "диск C".

Примеры логических устройств:

1. клавиатура: Устройство для ввода текстовой информации;
2. мышь: Устройство для управления курсором на экране;
3. монитор: Устройство для вывода графической информации;
4. принтер: Устройство для вывода информации на бумагу;
5. жесткий диск: Устройство для хранения данных.

Преимущества логических устройств:

1. Абстракция: они скрывают от пользователя детали реализации физического устройства.
2. Портативность: они позволяют использовать одно и то же приложение с различными физическими устройствами.
3. Упрощение программирования: они предоставляют единый интерфейс для взаимодействия с различными устройствами.

Взаимодействие логических и физических устройств:

1. Драйвер: программное обеспечение, которое связывает логическое устройство с физическим устройством.
2. Операционная система: обеспечивает управление логическими устройствами и их взаимодействие с физическими устройствами.

**1.2 Файл и файловая система**

Файл – это набор данных, хранящийся на вторичном носителе (например, жесткий диск) под определенным именем.

Файловая система – это иерархическая структура, которая позволяет организовать файлы и каталоги на вторичном носителе.

Иерархия файловой системы:

1. корневой каталог: вершина иерархической структуры;
2. каталоги (папки): контейнеры для файлов и других каталогов;
3. файлы: единицы данных, хранящиеся в каталогах.

Виды файлов:

1. Текстовые файлы: хранят текстовые данные.
2. Бинарные файлы: хранят данные в двоичном формате (например, изображения, видео, исполняемые файлы).
3. Системные файлы: файлы, используемые операционной системой для своей работы.

**1.2.1** Файл и каталог как объект:

Дескриптор - это уникальный идентификатор, который используется для доступа к файлу или каталогу.

Файловый дескриптор (Handle): дескриптор файла, используемый в операционной системе для управления доступом к файлам и каталогам[1].

**1.3 API для работы с файлами**

Основной набор API для работы с файлами:

1. CreateFile(): создание нового файла или открытие существующего файла;
2. CloseHandle(): закрытие файла;
3. ReadFile(): чтение данных из файла;
4. WriteFile(): запись данных в файл.

Синхронный ввод-вывод: эти операции ввода-вывода выполняются последовательно. Процесс ждет завершения операции ввода-вывода перед продолжением выполнения.

Неблокирующий синхронный ввод-вывод: операции ввода-вывода могут выполняться немедленно, но могут быть завершены позже. Процесс не ждет завершения операции ввода-вывода и может продолжить выполнение других операций. В Windows для неблокирующего ввода-вывода используются флаги FILE\_FLAG\_OVERLAPPED в CreateFile() и структура OVERLAPPED для хранения информации о неблокирующей операции.

Асинхронный ввод-вывод: эти операции ввода-вывода выполняются в фоновом режиме, не блокируя выполнение процесса.

Дополнительные API:

1. GetFileSize(): получает размер файла;
2. SetFilePointer(): изменяет текущую позицию в файле;
3. FlushFileBuffers(): очищает буферы записи в файл;
4. GetFileInformationByHandle(): получает информацию о файле.

Синхронный ввод-вывод: подходит для большинства простых операций с файлами.

Неблокирующий ввод-вывод: подходит для случаев, когда требуется повышение производительности и/или необходимость обработки других задач во время операции с файлом.

**1.4 Структура OVERLAPPED и функции ожидания**

Используется для асинхронного ввода-вывода. Она содержит информацию о состоянии операции ввода-вывода, а также о событиях, которые должны быть сгенерированы при завершении операции.

Функции ожидания (WaitFor\*): ожидают завершения операции ввода-вывода.

Варианты реализации контроля хода транзакций:

1. События: генерируются при завершении операций ввода-вывода.
2. Функции завершения: вызываются при завершении операций ввода-вывода.

Мультиплексированный ввод-вывод позволяет отслеживать и обрабатывать события ввода-вывода из нескольких устройств одновременно.

Функции select() используется для мультиплексированного ввода-вывода. Она позволяет указать список устройств и ждать события ввода-вывода от любого из них.

Отображение файлов в память позволяет отобразить файл в память как обычный массив байтов.

Функции CreateFileMapping(), MapViewOfFile(): используются для отображения файла в память.

Типовые применения файловых отображений:

1. повышение производительности: доступа к файлам:
2. общий доступ к файлам: между несколькими процессами;
3. редактирование больших файлов: без чтения и записи всего файла целиком[3].

Даже если дескриптор файла является синхронным (то есть созданным без флага FILE\_FLAG\_OVERLAPPED), структура OVERLAPPED может послужить в качестве альтернативы функции SetFilePointer для указания позиции в файле. В этом случае возврат после вызова функции ReadFile или иного вызова не происходит до тех пор, операция ввода/вывода пока не завершится. Этой возможностью мы уже воспользовались в главе 3. Также обратите внимание на то, что незавершенные операции ввода/вывода однозначно идентифицируются комбинацией дескриптора файла и соответствующей структуры OVERLAPPED.

**2 ИНСТРУМЕНТАЛЬНАЯ ЯЗЫКОВАЯ СРЕДА**

Для выполнения лабораторной работы был выбран язык разработки С++.

Язык программирования С++ представляет высокоуровневый компилируемый язык программирования общего назначения со статической типизацией, который подходит для создания самых различных приложений. На сегодняшний день С++ является одним из самых популярных и распространенных языков.

В качестве среды разраотки Visual Studio.

Win32 API (также называемый Windows API) – это исходная платформа для собственных Windows-приложений на языке C/C++, которым требуется прямой доступ к Windows и оборудованию. Он предоставляет интерфейс разработки первого класса без зависимости от управляемой среды выполнения, такой как .NET и WinRT (для приложений UWP для Windows 10). Благодаря этому API Win32 стает оптимальной платформой для приложений, которым требуется самый высокий уровень производительности и прямой доступ к системному оборудованию.

Файлы заголовков для API Windows позволяют создавать 32-разрядные и 64-разрядные приложения. Они включают объявления для версий API Юникода и ANSI.

Microsoft Visual C++ включает копии файлов заголовков Windows, которые были текущими в момент выпуска Visual C++. Таким образом, если вы устанавливаете обновленные файлы заголовков из пакета SDK, на компьютере может возникнуть несколько версий файлов заголовков Windows. Если вы не уверены, что используете последнюю версию файлов заголовков пакета SDK, при компиляции кода, использующего функции, представленные после выпуска Visual C++, будет получен следующий код ошибки: ошибка C2065: необъявленный идентификатор.

**3 РАБОТА ПРИЛОЖЕНИЯ**

* 1. **Структура приложения**

Приложение состоит из нескольких блоков, таких как:

1. функция асинхронного чтения;
2. функция асинхронной записи;
3. функция асинхронной сортировки;
4. main, где происходит их вызов для корректной работы приложения.

Также код содержит несколько обработчиков событий для ошибок.

* 1. **Функции приложения**

Само приложение производит асинхронное чтение, сортировку и запись.

Функции read\_file\_async, write\_file\_async и sort\_data\_async:

1. read\_file\_async: чтение данных.
2. write\_file\_async: запись данных в файл.
3. sort\_data\_async: сортировка полученных данных.

Блок main:

1. запуск и остановка таймера;
2. вызов функций чтения, записи и сортировки;
3. Подсчитывает затраченное время и выводит результат.
   1. **Анализ данных**

**1.3.1** Поиск "узкого места"

Время выполнения при разных размерах буфера:

1. 1024: 32 мс;
2. 4096: 3 мс;
3. 8192: 4 мс.

Время выполнения при разных количествах потоков:

1. 1 поток: 15 мс;
2. 2 потока: 20 мс;
3. 4 потока: 13 мс;
4. 8 потоков: 15 мс.

Вывод о влиянии на скорость работы:

1. Изменение размера буфера: Увеличение размера буфера может улучшить производительность, но при этом может привести к увеличению потребления памяти.
2. Изменение количества параллельно инициированных операций ввода-вывода: Увеличение количества параллельно инициированных операций ввода-вывода может улучшить производительность, но при этом может привести к перегрузке системы[2].

**1.3.2** Сравнение с традиционным подходом:

Время выполнения: 16 мс.

Асинхронный подход может значительно улучшить производительность, особенно для больших файлов, поскольку он позволяет выполнять чтение, обработку и запись данных параллельно.

Традиционный подход может быть более простым для реализации, но может быть медленнее для больших файлов.

**1.3.3** Сравнение с многопоточной реализацией:

Время выполнения: 15 мс.

Многопоточная реализация может также улучшить производительность, но может быть более сложной для реализации, особенно в Windows, где управление потоками может быть более ресурсоемким.

В некоторых случаях многопоточная реализация может быть более эффективной, чем асинхронный подход, особенно если обработка данных очень ресурсоемка.

**1.3.4** Вывод

Асинхронный подход может быть эффективным способом обработки больших файлов, особенно если обработка данных относительно проста.

Выбор между асинхронным подходом, многопоточным подходом и традиционным подходом зависит от конкретного приложения и его требований к производительности.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Подсистема ввода-вывода. Файловые системы – Электронный ресурс.– Режим доступа: https://intuit.ru/studies/courses/631/487/lecture/11059.

[2] Что такое узкое место производительности и как его определить – Электронный ресурс. – Режим доступа: https://www.loadview-testing.com/ru.

[3] WinAPI: Работа с файлами (основные функции). – Электронный ресурс. – Режим доступа: http://zetblog.ru/winapi-rabota-s-faylami-osnovnye-funktsii.html.

**ПРИЛОЖЕНИЕ** **А**

**(обязательное)**

**Реализация программы на языке С++**

#include <windows.h>

#include <iostream>

#include <vector>

#include <sstream>

#include <algorithm>

#include <string>

#include <chrono>

using namespace std;

vector<int> read\_file\_async(const wstring& filename) {

HANDLE fileHandle = CreateFile(

filename.c\_str(),

GENERIC\_READ,

FILE\_SHARE\_READ,

nullptr,

OPEN\_EXISTING,

FILE\_ATTRIBUTE\_NORMAL,

nullptr

);

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

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

return {};

}

OVERLAPPED overlapped = {};

overlapped.hEvent = CreateEvent(nullptr, TRUE, FALSE, nullptr);

DWORD bytesRead = 0;

char buffer[4096];

vector<int> data;

while (true) {

BOOL result = ReadFile(

fileHandle,

buffer,

sizeof(buffer),

&bytesRead,

&overlapped

);

if (result == FALSE && GetLastError() != ERROR\_IO\_PENDING) {

cerr << "Reading error : " << GetLastError() << endl;

CloseHandle(fileHandle);

CloseHandle(overlapped.hEvent);

return {};

}

WaitForSingleObject(overlapped.hEvent, INFINITE);

GetOverlappedResult(fileHandle, &overlapped, &bytesRead, TRUE);

string strData(buffer, bytesRead);

stringstream ss(strData);

string numberStr;

while (getline(ss, numberStr, ' ')) {

try {

int value = stoi(numberStr);

data.push\_back(value);

}

catch (const std::exception& e) {

cerr << "Number parsing error : " << e.what() << endl;

}

}

if (bytesRead < sizeof(buffer)) {

break;

}

}

CloseHandle(fileHandle);

CloseHandle(overlapped.hEvent);

return data;

}

void write\_file\_async(const wstring& filename, const vector<int>& data) {

HANDLE fileHandle = CreateFile(

filename.c\_str(),

GENERIC\_WRITE,

FILE\_SHARE\_WRITE,

nullptr,

CREATE\_ALWAYS,

FILE\_ATTRIBUTE\_NORMAL,

nullptr

);

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

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

return;

}

OVERLAPPED overlapped = {};

overlapped.hEvent = CreateEvent(nullptr, TRUE, FALSE, nullptr);

DWORD bytesWritten = 0;

string strData;

for (int i = 0; i < data.size(); i++) {

ostringstream oss;

oss << data[i] << " ";

strData += oss.str();

}

BOOL result = WriteFile(

fileHandle,

strData.c\_str(),

strData.size(),

&bytesWritten,

&overlapped

);

if (result == FALSE && GetLastError() != ERROR\_IO\_PENDING) {

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

CloseHandle(fileHandle);

CloseHandle(overlapped.hEvent);

return;

}

WaitForSingleObject(overlapped.hEvent, INFINITE);

GetOverlappedResult(fileHandle, &overlapped, &bytesWritten, TRUE);

CloseHandle(fileHandle);

CloseHandle(overlapped.hEvent);

}

void sort\_data\_async(vector<int>& data) {

sort(data.begin(), data.end());

}

int main() {

wstring filename = L"data.txt";

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

vector<int> data = read\_file\_async(filename);

sort\_data\_async(data);

write\_file\_async(filename, data);

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

auto duration = chrono::duration\_cast<chrono::milliseconds>(endTime - startTime);

cout << "Data sorted in file" << endl;

cout << "Time taken: " << duration.count() << " milliseconds" << endl;

return 0;

}