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 Сапроненко В.В.

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.

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

– выбрать файла для чтения;

– реализовать несколько методов чтения, сортировки и записи данных;

– подсчитать время, которое понадобилось на их обработку.

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]:

– инициализация файлов для записи и чтения данных;

– подсчёт времени операций считывания, сортировки и записи в файлы несколькими способами;

– вывод результатов.

**3.1 Запуск программы с заранее заданными параметрами**

Для начала обработки задаются необходимые значения в коде программы (рисунок 3.1).

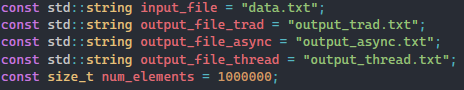


Рисунок 3.1– Инициализация данных

**3.2 Обработка данных и анализ временных затрат**

Программа использует асинхронные, многопоточные и традиционные операции чтения и записи для обработки данных из файла. Она заполняет файл случайными данными, после чего поочередно запускается процесс считывания, сортировки и записи данных в новые файлы. Программа замеряет время выполнения операций чтения, обработки и записи разными способами и выводит показатели. (рисунок 3.2).

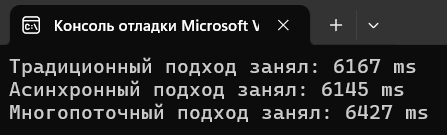


Рисунок 3.2 – Результат работы программы

ЗАКЛЮЧЕНИЕ

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

Основные аспекты, которые мы изучили, стали операции с файловой системой и подходы к организации ввода-вывода. Также были освоены 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 и выполнили обработку несколькими способами после чтения, а затем записали обработанные данные в соответствующие файлы. Затем замерили время выполнения операций.

Эта лабораторная работа не только позволила углубить 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 не только расширить 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 [Электронный ресурс]. – Режим доступа: https://www.youtube.com/watch?v=wTArIolxch0

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

ПРИЛОЖЕНИЕ А

(обязательное)

**Исходный код программы**

#include <iostream>

#include <fstream>

#include <vector>

#include <future>

#include <thread>

#include <algorithm>

#include <chrono>

void generate\_data\_file(const std::string& filename, size\_t num\_elements) {

std::ofstream out(filename);

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

out << rand() % 100000 << "\n";

}

}

void traditional\_processing(const std::string& input\_file, const std::string& output\_file) {

std::ifstream in(input\_file);

std::ofstream out(output\_file);

std::vector<int> data;

int value;

while (in >> value) {

data.push\_back(value);

}

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

for (const auto& v : data) {

out << v << "\n";

}

}

std::vector<int> async\_read\_data(const std::string& input\_file) {

std::ifstream in(input\_file);

std::vector<int> data;

int value;

while (in >> value) {

data.push\_back(value);

}

return data;

}

void async\_processing(const std::string& input\_file, const std::string& output\_file) {

auto read\_future = std::async(std::launch::async, async\_read\_data, input\_file);

auto data = read\_future.get();

auto sort\_future = std::async(std::launch::async, [&data]() {

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

});

sort\_future.wait();

std::ofstream out(output\_file);

for (const auto& v : data) {

out << v << "\n";

}

}

void threaded\_processing(const std::string& input\_file, const std::string& output\_file) {

std::vector<int> data;

std::thread read\_thread([&]() {

std::ifstream in(input\_file);

int value;

while (in >> value) {

data.push\_back(value);

}

});

read\_thread.join();

std::thread sort\_thread([&]() {

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

});

sort\_thread.join();

std::ofstream out(output\_file);

for (const auto& v : data) {

out << v << "\n";

}

}

void measure\_performance() {

setlocale(LC\_ALL, "ru");

const std::string input\_file = "data.txt";

const std::string output\_file\_trad = "output\_trad.txt";

const std::string output\_file\_async = "output\_async.txt";

const std::string output\_file\_thread = "output\_thread.txt";

const size\_t num\_elements = 1000000;

generate\_data\_file(input\_file, num\_elements);

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

traditional\_processing(input\_file, output\_file\_trad);

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

std::cout << "Традиционный подход занял: "

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

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

async\_processing(input\_file, output\_file\_async);

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

std::cout << "Асинхронный подход занял: "

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

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

threaded\_processing(input\_file, output\_file\_thread);

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

std::cout << "Многопоточный подход занял: "

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

}

int main() {

measure\_performance();

return 0;

}