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 Таргонский Д.А.

Проверил:

ассистент кафедры информатики Гриценко Н.Ю.

Минск 2024

**СОДЕРЖАНИЕ**

[1 Цель работы 3](#_Toc179320137)

[2 Краткие теоретические сведения 4](#_Toc179320138)

[3 Инструментальная языковая среда 5](#_Toc179320139)

[4 Описание программного продукта 6](#_Toc179320140)

[4.1 Описание используемых функций 6](#_Toc179320141)

[4.2 Алгоритм работы программы 7](#_Toc179320142)

[5 Результат выполнения программы 8](#_Toc179320143)

[Заключение 9](#_Toc179320144)

[Список использованных источников 10](#_Toc179320145)

[Приложения А (обязательное) исходный код продукта 11](#_Toc179320146)

# **1 ЦЕЛЬ РАБОТЫ**

Цель данной лабораторной работы – освоение и закрепление навыков работы с расширенными методами ввода-вывода и файловыми операциями в многопоточной среде. В ходе выполнения необходимо исследовать неблокирующие и асинхронные операции ввода-вывода, мультиплексирование, отображение файлов в память и их обработку через доступ к памяти.

Задача состоит в реализации программы, которая:

* отображает файл в память для дальнейшей обработки его содержимого;
* выполняет модификацию данных, вставляя символ в заданную позицию;
* сравнивает эффективность этого метода с традиционным способом (чтение файла, обработка и запись изменений);
* измеряет время выполнения программы при использовании как метода отображения файла в память, так и традиционного способа обработки.

2 КРАТКИЕ ТЕОРИТИЧЕСКИЕ СВЕДЕНИЯ

Подсистема ввода-вывода в операционных системах отвечает за связь программ с внешними устройствами и файлами. Существуют три основных типа ввода-вывода: синхронный, неблокирующий и асинхронный. Синхронный ввод-вывод приостанавливает выполнение программы до завершения операции, тогда как неблокирующий позволяет программе продолжать работу, не дожидаясь окончания ввода-вывода. Асинхронный ввод-вывод осуществляется в фоне, уведомляя систему о завершении задачи.

Логические устройства представляют собой абстракции физических устройств, таких как жесткие диски, сетевые адаптеры или принтеры, и позволяют программам взаимодействовать с ними через файловую систему или другие интерфейсы. [1]

Файлы — это именованные области данных, хранящиеся на устройствах. Файловая система управляет файлами и предоставляет к ним доступ. Файлы могут быть различных типов: обычные (содержащие данные), директории (папки со ссылками на другие файлы) и символические ссылки (указатели на файлы). Файловая система организует файлы в иерархическую структуру, начиная с корневой директории, которая содержит поддиректории и файлы, образуя древовидную систему.

Основные функции Windows API для работы с файлами включают CreateFile(), CloseHandle(), ReadFile() и WriteFile(). CreateFile() открывает файл или устройство, возвращая дескриптор для дальнейших операций. CloseHandle() освобождает ресурсы, связанные с этим дескриптором. ReadFile() и WriteFile() выполняют операции чтения и записи данных из/в файл, поддерживая как синхронный, так и асинхронный режим работы.

# **3 ИНСТРУМЕНТАЛЬНАЯ ЯЗЫКОВАЯ СРЕДА**

Для разработки программы был выбран язык программирования C++. Это компилируемый, высокопроизводительный язык программирования общего назначения, поддерживающий как низкоуровневые, так и высокоуровневые парадигмы программирования. C++ используется для разработки системного программного обеспечения, приложений, драйверов устройств, а также других программ, требующих высокой эффективности и прямого доступа к системным ресурсам.

В качестве интегрированной среды разработки (IDE) был выбран Microsoft Visual Studio, что является важным решением для разработчиков. Интегрированная среда разработки — это программа, которая объединяет все необходимые инструменты для написания кода, отладки и тестирования в одном интерфейсе, что значительно упрощает процесс разработки. Microsoft Visual Studio является одной из самых популярных IDE для разработки на C++. Она предлагает широкий набор функций, включая редактор кода с подсветкой синтаксиса и автозавершением, инструменты для отладки, встроенные средства для юнит-тестирования и возможности для профилирования и анализа производительности кода.

Разработка осуществляется на Microsoft Windows 10. Использование этой операционной системы позволяет эффективно работать с API и системными вызовами Windows, что важно для разработки программ, которые будут функционировать в этой среде. Windows 10 предоставляет разработчикам мощные инструменты для управления процессами и ресурсами, что критично для создания высокоэффективных приложений.

Вся работа ведется на ноутбуке, что обеспечивает мобильность и возможность работать в разных условиях — будь то офис, дом или в пути. Современные ноутбуки могут обеспечить достаточную производительность для разработки и тестирования программ, особенно если они оснащены хорошими процессорами и достаточным объемом оперативной памяти.

Таким образом, выбор Microsoft Visual Studio в качестве IDE, использование операционной системы Windows 10 и работа на ноутбуке создают оптимальные условия для разработки программ на C++. Это позволяет эффективно использовать все доступные инструменты и ресурсы, что в свою очередь способствует созданию качественного и производительного программного продукта.

4 ОПИСАНИЕ ПРОГРАММНОГО ПРОДУКТА

Программа демонстрирует два подхода работы с файлами на примере вставки символа в определённую позицию файла. Первый метод использует классический способ чтения и записи файлов, а второй — более продвинутый метод отображения файла в память (memory mapping). Разберём каждый из этих подходов с точки зрения используемых функций и работы программы.[2]

# **4.1 Описание используемых функций**

Далее будут представлены используемые функции для работы с файлами и вводом-выводов в них и из них:

1. CreateFileA(): Эта функция открывает файл для чтения или записи. Она возвращает дескриптор файла, который потом используется для операций ввода-вывода. В программе она применяется дважды: сначала для открытия исходного файла, затем — для создания нового файла, куда будут записаны результаты работы программы.
2. GetFileSize(): Позволяет получить размер файла, что необходимо для точного определения, сколько данных нужно обработать. Функция возвращает количество байт в файле.
3. CreateFileMappingA(): Создаёт объект отображения файла в память, позволяя работать с содержимым файла как с блоком памяти, минуя стандартные операции чтения и записи. Это улучшает производительность при работе с большими файлами.
4. MapViewOfFile(): Отображает файл в память и позволяет работать с его содержимым напрямую. Это важный шаг при использовании отображения файлов в память, так как дальнейшие операции проводятся именно в этом блоке памяти.
5. InsertCharacter(): Вставляет заданный символ в определённую позицию в массиве данных. Эта функция копирует содержимое файла до указанной позиции, вставляет символ и затем добавляет оставшуюся часть файла.
6. WriteFile(): Записывает данные в файл. Используется как при традиционном методе, так и при методе с отображением файла в память для сохранения обработанных данных в новый файл.
7. UnmapViewOfFile(): Закрывает отображение файла в память, освобождая ресурсы. Это важный шаг для завершения работы с памятью и файлом.
8. CloseHandle(): Закрывает дескрипторы файлов и освобождает ресурсы, связанные с объектами файлов и памяти.

Таким образом были представлены используемые в программном продукте функции работы с файлами

# **4.2 Алгоритм работы программы**

Алгоритм программного продукта можно представить следующим образом:

1. Открытие исходного файла для чтения с помощью функции CreateFileA().
2. Определение размера файла с использованием функции GetFileSize().
3. Традиционный метод (чтение-запись):
4. Метод отображения файла в память:
5. Закрытие отображения файла и освобождение памяти с помощью UnmapViewOfFile().
6. Закрытие всех открытых файловых дескрипторов через CloseHandle().

Программа начинает с того, что открывает исходный файл, размер которого измеряется с помощью функции GetFileSize(). Этот размер необходим для того, чтобы определить, где именно следует вставить новый символ.

После определения размера программа применяет два разных метода работы с файлами.

В первом методе файл читается по частям, каждая часть проверяется на то, содержит ли она позицию для вставки символа. Если текущий блок данных содержит нужную позицию, программа вставляет символ в этот блок, а затем записывает его в новый файл. Такой подход использует классический алгоритм чтения и записи данных через ReadFile() и WriteFile().

Во втором методе используется механизм отображения файла в память. Вместо того чтобы считывать файл частями, программа отображает весь файл в память с помощью CreateFileMappingA() и MapViewOfFile(). После этого можно напрямую изменять данные файла в памяти, что значительно ускоряет процесс. Вставка символа происходит аналогично первому методу, только теперь она выполняется в памяти, что минимизирует количество операций с файловой системой. [3]

После того как данные файла были изменены, программа записывает их в новый файл. Далее программа закрывает все открытые дескрипторы файлов и освобождает ресурсы памяти, завершив выполнение программы.

**5 РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ ПРОГРАММЫ**

Приложение, построенное на базе, будет выполнять копирование данных из исходного файла в новый, добавляя заданный символ в определённую позицию.

Оно будет поддерживать два ключевых режима работы: синхронную обработку данных с использованием стандартных операций чтения и записи, а также обработку через отображение файла в память для повышения эффективности.

На рисунке 5.1 представлен результат работы синхронной функции.

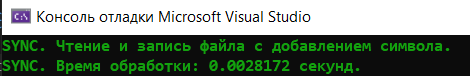


Рисунок 5.1 – Результат работы синхронной функции

На рисунке 5.2 представлен результат работы функции отображения файла в память.

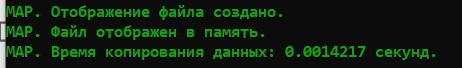


Рисунок 5.2 – Результат работы функции отображения файла

На рисунке 5.3 представлен изменённый файл через синхронную функцию.

Изображение выглядит как текст, Шрифт, снимок экрана, белый

Автоматически созданное описание

Рисунок 5.3 – Изменённый файл через синхронную функцию

На рисунке 5.4 представлен изменённый файл через отображение файла.

Изображение выглядит как текст, Шрифт, снимок экрана, белый

Автоматически созданное описание

Рисунок 5.4 – Изменённый файл через отображение файла

Таким образом, была наглядно показана разница между синхронной функцией и отображением файла в память.

# **ЗАКЛЮЧЕНИЕ**

В ходе выполнения лабораторной работы была разработана программа, которая копирует данные из одного файла в другой с добавлением указанного символа в заданную позицию. Программа реализует два подхода к обработке данных: синхронный метод, основанный на стандартных операциях чтения и записи файлов, и асинхронный метод, использующий отображение файла в память.

По результатам экспериментов можно заключить, что метод с отображением файла в память работает значительно быстрее по сравнению с традиционным подходом.

Во время работы над проектом были приобретены навыки по следующим направлениям: основные принципы работы с файлами и файловыми системами в операционной системе Windows, включая операции открытия, чтения, записи и закрытия файлов; использование Windows API для управления файловыми дескрипторами и отображения файлов в память, что позволяет более эффективно распоряжаться системными ресурсами и повышать производительность; изучение основ многопоточности и асинхронного ввода-вывода, что является важным аспектом разработки высокопроизводительных программ; практическое применение методов вставки данных в буферы и работы с памятью, что углубило понимание работы с динамическими структурами данных.

Таким образом, выполненная работа способствовала расширению знаний в области системного программирования и укреплению навыков работы с файловыми операциями и управлением памятью в операционной среде Windows.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

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

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

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

ПРИЛОЖЕНИЕ А

(обязательное)

**Исходный код программы**

#include <windows.h>

#include <iostream>

#include <vector>

#include <thread>

#include <chrono>

#include <fstream>

void InsertCharacterMap(void\* buffer, DWORD fileSize, char insertChar, DWORD position, void\*& newBuffer, DWORD& newBufferSize) {

if (position > fileSize) {

std::cerr << "Ошибка: позиция вставки выходит за пределы размера файла." << std::endl;

return;

}

newBufferSize = fileSize + 1;

newBuffer = malloc(newBufferSize);

if (!newBuffer) {

std::cerr << "Ошибка выделения памяти для нового буфера." << std::endl;

return;

}

memcpy(newBuffer, buffer, position);

static\_cast<char\*>(newBuffer)[position] = insertChar;

memcpy(static\_cast<char\*>(newBuffer) + position + 1, static\_cast<char\*>(buffer) + position, fileSize - position);

}

void MapFileToMemory(const char\* inputFilename) {

HANDLE hFile = CreateFileA(inputFilename, GENERIC\_READ, 0, NULL, OPEN\_EXISTING, FILE\_ATTRIBUTE\_NORMAL, NULL);

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

std::cerr << "Ошибка открытия файла для чтения." << std::endl;

return;

}

DWORD fileSize = GetFileSize(hFile, NULL);

if (fileSize == INVALID\_FILE\_SIZE) {

std::cerr << "Ошибка получения размера файла." << std::endl;

CloseHandle(hFile);

return;

}

HANDLE hMapping = CreateFileMappingA(hFile, NULL, PAGE\_READONLY, 0, 0, NULL);

if (hMapping == NULL) {

std::cerr << "Ошибка создания отображения файла." << std::endl;

CloseHandle(hFile);

return;

}

std::cout << "MAP.Отображение файла создано." << std::endl;

char insertChar = 'X';

DWORD insertPosition = 5;

void\* newBuffer = nullptr;

DWORD newBufferSize = 0;

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

void\* dataArray = MapViewOfFile(hMapping, FILE\_MAP\_READ, 0, 0, 0);

if (dataArray == NULL) {

std::cerr << "Ошибка отображения файла в память." << std::endl;

CloseHandle(hMapping);

CloseHandle(hFile);

return;

}

std::cout << "MAP.Файл отображен в память." << std::endl;

InsertCharacterMap(dataArray, fileSize, insertChar, insertPosition, newBuffer, newBufferSize);

if (!newBuffer) {

UnmapViewOfFile(dataArray);

CloseHandle(hMapping);

CloseHandle(hFile);

return;

}

HANDLE hOutputFile = CreateFileA("output.txt", GENERIC\_WRITE, 0, NULL, CREATE\_ALWAYS, FILE\_ATTRIBUTE\_NORMAL, NULL);

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

std::cerr << "Ошибка открытия выходного файла." << std::endl;

UnmapViewOfFile(dataArray);

CloseHandle(hMapping);

CloseHandle(hFile);

return;

}

DWORD bytesWritten;

if (!WriteFile(hOutputFile, newBuffer, newBufferSize, &bytesWritten, NULL)) {

std::cerr << "Ошибка записи в выходной файл." << std::endl;

}

else if (bytesWritten != newBufferSize) {

std::cerr << "Количество записанных байт не соответствует размеру буфера!" << std::endl;

}

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

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

std::cout << "MAP.Время копирования данных: " << elapsed.count() << " секунд." << std::endl;

UnmapViewOfFile(dataArray);

CloseHandle(hOutputFile);

CloseHandle(hMapping);

CloseHandle(hFile);

}

void InsertCharacter(char\* buffer, DWORD& bytesRead, char insertChar, DWORD position, DWORD bufferSize, char\*& newBuffer, DWORD& newBufferSize) {

if (position > bytesRead) {

std::cerr << "Ошибка: позиция вставки выходит за пределы прочитанных данных." << std::endl;

return;

}

newBufferSize = bytesRead + 1;

newBuffer = new char[newBufferSize];

memcpy(newBuffer, buffer, position);

newBuffer[position] = insertChar;

memcpy(newBuffer + position + 1, buffer + position, bytesRead - position);

bytesRead = newBufferSize;

}

void CopyFileData(const char\* sourceFilename, const char\* destFilename, char insertChar, DWORD insertPosition) {

HANDLE hSourceFile = CreateFileA(sourceFilename, GENERIC\_READ, 0, NULL, OPEN\_EXISTING, FILE\_ATTRIBUTE\_NORMAL, NULL);

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

std::cerr << "Ошибка открытия файла для чтения: " << GetLastError() << std::endl;

return;

}

HANDLE hDestFile = CreateFileA(destFilename, GENERIC\_WRITE, 0, NULL, CREATE\_ALWAYS, FILE\_ATTRIBUTE\_NORMAL, NULL);

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

std::cerr << "Ошибка открытия файла для записи: " << GetLastError() << std::endl;

CloseHandle(hSourceFile);

return;

}

const DWORD bufferSize = 4096;

char buffer[bufferSize];

DWORD bytesRead, bytesWritten;

DWORD totalBytesRead = 0;

char\* newBuffer = nullptr;

DWORD newBufferSize = 0;

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

std::cout << "SYNC. Чтение и запись файла с добавлением символа." << std::endl;

while (ReadFile(hSourceFile, buffer, bufferSize, &bytesRead, NULL) && bytesRead > 0) {

if (totalBytesRead <= insertPosition && insertPosition < totalBytesRead + bytesRead) {

DWORD localInsertPos = insertPosition - totalBytesRead;

InsertCharacter(buffer, bytesRead, insertChar, localInsertPos, bufferSize, newBuffer, newBufferSize);

}

else {

newBuffer = buffer;

newBufferSize = bytesRead;

}

totalBytesRead += bytesRead;

if (!WriteFile(hDestFile, newBuffer, newBufferSize, &bytesWritten, NULL) || bytesWritten != newBufferSize) {

std::cerr << "Ошибка записи в файл: " << GetLastError() << std::endl;

CloseHandle(hSourceFile);

CloseHandle(hDestFile);

return;

}

if (newBuffer != buffer) {

delete[] newBuffer;

}

}

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

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

std::cout << "SYNC. Время обработки: " << elapsed.count() << " секунд." << std::endl;

CloseHandle(hSourceFile);

CloseHandle(hDestFile);

}

int main() {

const char\* filename = "file.txt";

char insertChar = 'X';

DWORD insertPosition = 5;

CopyFileData(filename, "destFilename.txt", insertChar, insertPosition);

MapFileToMemory(filename);

return 0;

}