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

ОТЧЕТ

к лабораторной работе №3

на тему

**УПРАВЛЕНИЕ ПАМЯТЬЮ И ВВОДОМ-ВЫВОДОМ, РАСШИРЕННЫЕ ВОЗМОЖНОСТИ ВВОДА-ВЫВОДА WINDOWS.**

Студент Е. А. Сироткин

Преподаватель Н. Ю. Гриценко

Минск 2023

СОДЕРЖАНИЕ

[1 Цель работы 3](#_Toc146752068)

[2 Теоретические сведения 4](#_Toc146752069)

[3 Результат выполнения 5](#_Toc146752070)

[Заключение 6](#_Toc146752071)

[Список использованных источников 7](#_Toc146752072)

[Приложение А (обязательное) Листинг кода 8](#_Toc146752073)

1. **ЦЕЛЬ РАБОТЫ**

Целью данной лабораторной работы является исследование и практическое применение методов управления памятью и вводом-выводом, а также использование продвинутых функциональных возможностей ввода-вывода в операционной системе *Windows* через *API* подсистемы памяти *Win32*. Основные задачи лабораторной работы включают в себя освоение навыков работы с асинхронными операциями ввода-вывода, реализацию отображения файлов в память и приобретение практического опыта в разработке эффективных приложений, использующих указанные возможности операционной системы *Windows*.

1. **ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

Сопоставление файлов (*Memory-Mapped Files*) представляет собой механизм, который устанавливает связь между содержимым файла и адресным пространством процесса. В процессе создается объект сопоставления файлов, который позволяет процессу взаимодействовать с содержимым файла через представление файла. Это представление представляет собой часть виртуального адресного пространства, доступное процессу для работы с данными файла. Сопоставление файлов дает возможность для выполнения как случайных, так и последовательных операций ввода-вывода.

Эффективность работы с сопоставленными файлами заключается в том, что файл находится на диске, а его представление находится в памяти, что позволяет быстро и эффективно обрабатывать данные.

Асинхронность подразумевает выполнение задачи или операции, которая не препятствует выполнению основного потока программы. Вместо того, чтобы ожидать завершения длительной операции, приложение может продолжать выполнять другие задачи или операции. Использование асинхронности способствует повышению производительности и отзывчивости приложений, так как не блокирует пользовательский интерфейс или другие потоки выполнения во время выполнения задач.

*Win32 API* предоставляет инструменты для контроля над файловыми указателями с помощью функций, таких как *SetFilePointer* и *SetEndOfFile*. С их помощью можно устанавливать положение файлового указателя в файле, что позволяет выполнять чтение данных с определенного места или запись данных в конкретное положение файла.

Потоки представляют собой легковесные подпроцессы внутри основного процесса, которые могут выполняться параллельно и многозадачно. Они используются для выполнения асинхронных задач, таких как обработка данных, выполнение вычислений, ввод-вывод и другие операции, которые могут выполняться параллельно.

Взаимодействие с устройствами в рамках *Win32* *API* осуществляется с помощью функций, таких как *CreateFile*, при которой указывается имя устройства. Этот подход позволяет выполнять операции ввода-вывода, такие как чтение и запись данных, непосредственно с устройствами, включая последовательные порты и сетевые устройства.

1. **РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ**

Основой является приложение, которое способно производить запись, изменять текст и отображать количество введенных символов(рисунок 1)*.*

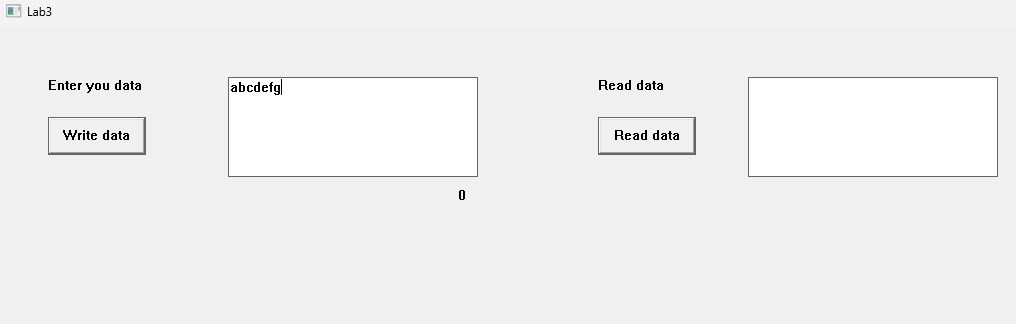


Рисунок 1 – Окно приложения

Был реализован функционал в виде сохранения текста в файл с использованием маппинга файла в память и дальнейшее его считывание из этого файла (рисунок 2).

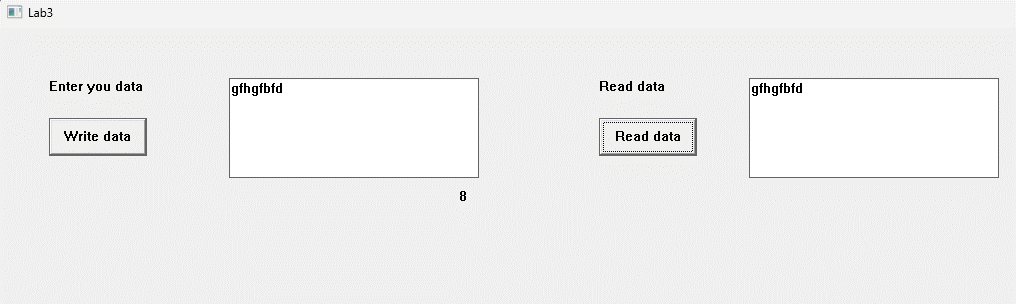


Рисунок 2– Окно приложения после считывания

# ЗАКЛЮЧЕНИЕ

В процессе выполнения лабораторной работы были освоены основы взаимодействия с операциями ввода и вывода, включая аспекты управления памятью, асинхронных операций и отображения файлов в память. Для приложения была добавлена возможность считывания текста в файл с последующим его выводом используя маппинг.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Безруков В. Win32 API. Программирование /учебное пособие. ─ СПб: СПбГУ ИТМО, 2009. ─ 90 с.
2. Создание классических приложений для Windows с использованием API Win32 [Электронный ресурс]. ─ Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/api.

# ПРИЛОЖЕНИЕ А (обязательное) Листинг кода

Листинг 1 – Файл *Lab3.cpp*

#include <windows.h>

#include <stdlib.h>

#include <string>

using namespace std;

#define WRITE\_DATA 1

#define READ\_DATA 2

LRESULT CALLBACK WindowProcedure(HWND, UINT, WPARAM, LPARAM);

void AddControls(HWND);

void WriteData(HWND);

void ReadData(HWND);

HWND hDataToWrite, hReadData, hCharCount;

wchar\_t textToWrite[200];

int WINAPI WinMain(HINSTANCE hInst, HINSTANCE hPrevInst, LPSTR args, int ncmdshow) {

WNDCLASS wc = { 0 };

wc.hbrBackground = (HBRUSH)COLOR\_WINDOW;

wc.hCursor = LoadCursor(NULL, IDC\_ARROW);

wc.hInstance = hInst;

wc.lpszClassName = L"Lab3";

wc.lpfnWndProc = WindowProcedure;

if (!RegisterClassW(&wc)) {

return -1;

}

CreateWindowW(L"Lab3", L"Lab3", WS\_OVERLAPPEDWINDOW | WS\_VISIBLE, 100, 100, 1400, 700, NULL, NULL, NULL, NULL);

MSG msg = { 0 };

while (GetMessage(&msg, NULL, NULL, NULL)) {

TranslateMessage(&msg);

DispatchMessage(&msg);

}

return 0;

}

LRESULT CALLBACK WindowProcedure(HWND hWnd, UINT msg, WPARAM wp, LPARAM lp) {

switch (msg) {

case WM\_COMMAND:

switch (wp) {

case WRITE\_DATA:

WriteData(hWnd);

break;

case READ\_DATA:

ReadData(hWnd);

break;

}

break;

case WM\_CREATE:

AddControls(hWnd);

break;

case WM\_DESTROY:

PostQuitMessage(0);

break;

default:

return DefWindowProcW(hWnd, msg, wp, lp);

}

}

void AddControls(HWND hWnd) {

CreateWindow(L"static", L"Enter you data", WS\_VISIBLE | WS\_CHILD, 50, 50, 210, 38, hWnd, NULL, NULL, NULL);

hDataToWrite = CreateWindow(L"edit", L"", WS\_VISIBLE | WS\_CHILD | WS\_BORDER | ES\_MULTILINE, 230, 50, 250, 100, hWnd, NULL, NULL, NULL);

CreateWindow(L"button", L"Write data", WS\_VISIBLE | WS\_CHILD | WS\_BORDER, 50, 90, 98, 38, hWnd, (HMENU)WRITE\_DATA, NULL, NULL);

CreateWindow(L"static", L"Read data", WS\_VISIBLE | WS\_CHILD, 600, 50, 210, 38, hWnd, NULL, NULL, NULL);

hReadData = CreateWindow(L"edit", L"", WS\_VISIBLE | WS\_CHILD | WS\_BORDER | ES\_MULTILINE, 750, 50, 250, 100, hWnd, NULL, NULL, NULL);

CreateWindow(L"button", L"Read data", WS\_VISIBLE | WS\_CHILD | WS\_BORDER, 600, 90, 98, 38, hWnd, (HMENU)READ\_DATA, NULL, NULL);

hCharCount = CreateWindow(L"static", L"0", WS\_VISIBLE | WS\_CHILD, 460, 160, 40, 20, hWnd, NULL, NULL, NULL);

}

void WriteData(HWND hWnd) {

// Создаем или открываем файл

HANDLE hFile = CreateFileW(

L"mytextfile.txt",

GENERIC\_WRITE | GENERIC\_READ,

FILE\_SHARE\_READ | FILE\_SHARE\_WRITE,

NULL,

CREATE\_ALWAYS,

FILE\_ATTRIBUTE\_NORMAL | FILE\_FLAG\_OVERLAPPED,

NULL

);

GetWindowTextW(hDataToWrite, textToWrite, 200);

int numChars = wcslen(textToWrite) + 1;

SetWindowTextW(hCharCount, std::to\_wstring(numChars-1).c\_str());

int numBytes = numChars \* sizeof(wchar\_t);

DWORD fileSize = numBytes;

SetFilePointer(hFile, fileSize, NULL, FILE\_BEGIN);

SetEndOfFile(hFile);

// Создаем файловый объект отображения

HANDLE hMapFile = CreateFileMapping(

hFile,

NULL,

PAGE\_READWRITE,

0,

fileSize,

NULL

);

// Отображаем файл в память

LPVOID pData = MapViewOfFile(

hMapFile,

FILE\_MAP\_WRITE,

0,

0,

0

);

memcpy(pData, textToWrite, numBytes);

// Освобождаем ресурсы

UnmapViewOfFile(pData);

CloseHandle(hMapFile);

CloseHandle(hFile);

}

void ReadData(HWND hWnd) {

HANDLE hFile = CreateFile(

L"mytextfile.txt",

GENERIC\_READ,

0,

NULL,

OPEN\_EXISTING,

FILE\_ATTRIBUTE\_NORMAL,

NULL

);

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

// Обработка ошибки открытия файла

return;

}

// Определяем размер файла

DWORD fileSize = GetFileSize(hFile, NULL);

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

// Обработка ошибки определения размера файла

CloseHandle(hFile);

return;

}

// Создаем файловый объект отображения

HANDLE hMapFile = CreateFileMapping(

hFile,

NULL,

PAGE\_READONLY,

0,

fileSize,

NULL

);

if (hMapFile == NULL) {

// Обработка ошибки создания файлового отображения

CloseHandle(hFile);

return;

}

// Отображаем файл в память

LPVOID pData = MapViewOfFile(

hMapFile,

FILE\_MAP\_READ,

0,

0,

0

);

if (pData == NULL) {

// Обработка ошибки отображения файла в память

CloseHandle(hMapFile);

CloseHandle(hFile);

return;

}

// Копируем данные из отображенного файла в ваш буфер

wchar\_t textToRead[200];

memcpy(textToRead, pData, fileSize);

SetWindowText(hReadData, textToRead);

// Освобождаем ресурсы

UnmapViewOfFile(pData);

CloseHandle(hMapFile);

CloseHandle(hFile);

}