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. Функции API подсистемы памяти Win 32. Организация и контроль асинхронных операций ввода-вывода. Отображение файлов в память»

Выполнил:

студент гр. 153504

Мамченко К.А.

Проверил:

Гриценко Н.Ю.

Минск 2023

**СОДЕРЖАНИЕ**

[1 Цели работы 3](#_gjdgxs)

[2 Краткие теоретические сведения 4](#_30j0zll)

[3 Полученные результаты](#_1fob9te) 6

[Вывод](#_4q3p755gzys6) 8

[Список использованных источников](#_4q3p755gzys6) 9

[Приложение А (обязательное) Листинг кода](#_4q3p755gzys6) 10

**1 ЦЕЛИ РАБОТЫ**

1 Изучить расширенные возможности ввода-вывода Windows.

2 Изучить функции API подсистемы памяти Win32 и их использование.

3 Изучить организацию и контроль асинхронных операций ввода-вывода.

4 Изучить отображение файлов в память.

5 Создать текстовый редактор с поддержкой записи и чтения файлов в двоичном формате с использованием маппинга файлов в память.

**2 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

Маппинг файлов в память представляет собой механизм, который позволяет отобразить содержимое файла непосредственно в адресное пространство процесса. Такое отображение позволяет работать с файлом, как если бы он находился в оперативной памяти, и обеспечивает более эффективный доступ к данным.

Для работы с маппингом файлов в память в Win32 API используются следующие функции:

1 *CreateFileMapping* ­ функция используется для создания объекта отображения файла в память.

2 *MapViewOfFile* ­ функция используется для создания отображения файла в адресное пространство процесса.

3 *UnmapViewOfFile* ­ функция используется для разрыва связи между отображением файла и адресным пространством процесса.

Процесс работы с маппингом файлов в память обычно выглядит следующим образом:

1 Открытие файла с помощью функции *CreateFile*.

2 Создание объекта отображения файла с помощью функции *CreateFileMapping*.

3 Создание отображения файла в адресное пространство процесса с помощью функции *MapViewOfFile*.

4 Работа с данными, отображенными в памяти, как с обычиз массивом памяти.

5 Отмена отображения файла с помощью функции *UnmapViewOfFile*.

6 Закрытие дескриптора объекта отображения файла и файла.

На рисунке 2.1 представлено отношение между файлом на диске, файлом маппинга и его отображением.

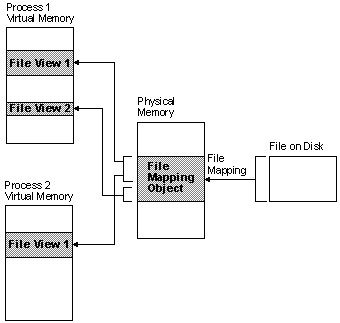


Рисунок 2.1 – Схема отображения в память

**3 ПОЛУЧЕННЫЕ РЕЗУЛЬТАТЫ**

Был доработан текстовый редактор из предыдущих двух лабораторных и добавлена поддержка записи и чтения файлов в двоичном формате с использованием маппинга файлов в память.

Изображение выглядит как текст, программное обеспечение, Мультимедийное программное обеспечение, Значок на компьютере

Автоматически созданное описание

Рисунок 3.1 – Чтение двоичного файла

Изображение выглядит как текст, программное обеспечение, Мультимедийное программное обеспечение, Значок на компьютере

Автоматически созданное описание

Рисунок 3.2 ­ Запись двоичного файла

На рисунке 3.1 продемонстрирована работа чтения двоичного файла маппингом в память с помощью чтения случайного двоичного файла на компьютере. На рисунке 3.2 продемонстрирована запись двоичного файла посредством создания нового файла и сохранения его на компьютере в двоичном формате.

**ВЫВОД**

В результате выполнения работы был разработан текстовый редактор с поддержкой записи и чтения файлов в двоичном формате с использованием маппинга файлов в память.

Для этой цели был изучен теоретический материал и средства Win32 Api, используемые для этих целей.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

[1] File Mapping, документация Microsoft [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/en-us/windows/win32/memory/file-mapping>.

[2] Записки программиста [Электронный ресурс] ­ Режим доступа: <https://eax.me/winapi-file-mapping/>.

**ПРИЛОЖЕНИЕ А**

**(обязательное)**

**Листинг кода**

#include <windows.h>

// Идентификаторы элементов управления

#define ID\_EDIT 1001

#define ID\_FILE\_NEW 2001

#define ID\_FILE\_OPEN 2002

#define ID\_FILE\_SAVE 2003

#define ID\_NEW\_WINDOW 2004

#define ID\_STYLE\_BOLD 3001

#define ID\_STYLE\_ITALIC 3002

#define ID\_STYLE\_UNDERLINE 3003

#define ID\_BACKGROUND\_COLOR 4000

#define ID\_TEXT\_COLOR 4001

#define ID\_FONT\_FACE\_TNR 5001

#define ID\_FONT\_FACE\_CALIBRI 5002

#define ID\_FONT\_FACE\_ARIAL 5003

#define ID\_FONT\_SIZE\_10 6010

#define ID\_FONT\_SIZE\_12 6012

#define ID\_FONT\_SIZE\_14 6014

#define ID\_FONT\_SIZE\_16 6016

#define ID\_FONT\_SIZE\_18 6018

#define ID\_FONT\_SIZE\_20 6020

#define ID\_BIN\_SAVE 7001

#define ID\_BIN\_UPLOAD 7002

bool g\_bBold = false;

bool g\_bItalic = false;

bool g\_bUnderline = false;

// Глобальные переменные

HWND g\_hMainWindow; // Основное окно приложения

HWND g\_hEdit; // Редактор текста

int g\_nDocCount = 1; // Счетчик документов

TCHAR g\_szCurrentFile[MAX\_PATH] = TEXT(""); // Текущий открытый файл

HBRUSH hEditBgBrush = NULL;

HBRUSH hEditTextColorBrush = NULL;

static COLORREF bgColor = RGB(255, 255, 255); // Цвет фона по умолчанию

static COLORREF textColor = RGB(0, 0, 0); // Цвет фона по умолчанию

const wchar\_t\* g\_FontFace = L"Arial";

int g\_fontSize = 16;

HANDLE g\_hFileMapping;

LPVOID g\_lpFileBase;

DWORD fileSize = 0;

// Прототипы функций

LRESULT CALLBACK WindowProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam);

void CreateNewDocument();

void OpenDocument();

void SaveDocument();

void UpdateWindowTitle();

void CreateNeWindow();

void UpdateTextStyle(HWND hWnd);

void ChangeBgc(HWND hwnd);

void ChangeTextColor(HWND hwnd);

void WriteToFile();

void ReadFromFile();

// Прототип функции-перехватчика

LRESULT CALLBACK KeyboardProc(int nCode, WPARAM wParam, LPARAM lParam);

// Глобальная переменная для хранения хука

HHOOK g\_hHook = NULL;

// Точка входа в программу

int WINAPI wWinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPWSTR lpCmdLine, int nCmdShow)

{

g\_hHook = SetWindowsHookEx(WH\_KEYBOARD\_LL, KeyboardProc, NULL, 0);

if (g\_hHook == NULL)

{

// Обработка ошибки

return 1;

}

// Регистрация класса окна

const wchar\_t CLASS\_NAME[] = L"TextEditorClass";

WNDCLASS wc = {};

wc.lpfnWndProc = WindowProc;

wc.hInstance = hInstance;

wc.lpszClassName = CLASS\_NAME;

RegisterClass(&wc);

// Создание главного окна

g\_hMainWindow = CreateWindowEx(

0,

CLASS\_NAME,

L"Text Editor",

WS\_OVERLAPPEDWINDOW,

CW\_USEDEFAULT, CW\_USEDEFAULT, CW\_USEDEFAULT, CW\_USEDEFAULT,

NULL,

NULL,

hInstance,

NULL

);

if (g\_hMainWindow == NULL)

{

return 0;

}

// Создание меню

HMENU hMenu = CreateMenu();

HMENU hFileMenu = CreateMenu();

HMENU hStyleMenu = CreateMenu();

HMENU hFontMenu = CreateMenu();

HMENU hFontSize = CreateMenu();

AppendMenu(hFileMenu, MF\_STRING, ID\_FILE\_NEW, L"&New");

AppendMenu(hFileMenu, MF\_STRING, ID\_FILE\_OPEN, L"&Open");

AppendMenu(hFileMenu, MF\_STRING, ID\_FILE\_SAVE, L"&Save");

AppendMenu(hFileMenu, MF\_STRING, ID\_NEW\_WINDOW, L"&New WIndow");

AppendMenu(hFileMenu, MF\_STRING, ID\_BIN\_SAVE, L"&Bin Save");

AppendMenu(hFileMenu, MF\_STRING, ID\_BIN\_UPLOAD, L"&Bin Open");

AppendMenu(hStyleMenu, MF\_STRING, ID\_STYLE\_BOLD, L"&Bold");

AppendMenu(hStyleMenu, MF\_STRING, ID\_STYLE\_ITALIC, L"&Italic");

AppendMenu(hStyleMenu, MF\_STRING, ID\_STYLE\_UNDERLINE, L"&Underline");

AppendMenu(hFontMenu, MF\_STRING, ID\_FONT\_FACE\_TNR, L"&Times New Roman");

AppendMenu(hFontMenu, MF\_STRING, ID\_FONT\_FACE\_ARIAL, L"&Arial");

AppendMenu(hFontMenu, MF\_STRING, ID\_FONT\_FACE\_CALIBRI, L"&Calibri");

AppendMenu(hFontSize, MF\_STRING, ID\_FONT\_SIZE\_10, L"&10");

AppendMenu(hFontSize, MF\_STRING, ID\_FONT\_SIZE\_12, L"&12");

AppendMenu(hFontSize, MF\_STRING, ID\_FONT\_SIZE\_14, L"&14");

AppendMenu(hFontSize, MF\_STRING, ID\_FONT\_SIZE\_16, L"&16");

AppendMenu(hFontSize, MF\_STRING, ID\_FONT\_SIZE\_18, L"&18");

AppendMenu(hFontSize, MF\_STRING, ID\_FONT\_SIZE\_20, L"&20");

AppendMenu(hMenu, MF\_POPUP, (UINT\_PTR)hFileMenu, L"&File");

AppendMenu(hMenu, MF\_POPUP, (UINT\_PTR)hStyleMenu, L"&Style");

AppendMenu(hMenu, MF\_POPUP, (UINT\_PTR)hFontMenu, L"&Font Face");

AppendMenu(hMenu, MF\_POPUP, (UINT\_PTR)hFontSize, L"&Font Size");

AppendMenu(hMenu, MF\_STRING, ID\_BACKGROUND\_COLOR, L"&Background color");

AppendMenu(hMenu, MF\_STRING, ID\_TEXT\_COLOR, L"&Text color");

SetMenu(g\_hMainWindow, hMenu);

ShowWindow(g\_hMainWindow, nCmdShow);

// Цикл обработки сообщений

MSG msg;

while (GetMessage(&msg, NULL, 0, 0))

{

TranslateMessage(&msg);

DispatchMessage(&msg);

}

// Удаление хука

UnhookWindowsHookEx(g\_hHook);

return 0;

}

// Обработчик сообщений окна

LRESULT CALLBACK WindowProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam)

{

static HFONT hDefaultFont = nullptr; // Хранит дескриптор шрифта по умолчанию

switch (uMsg)

{

case WM\_CREATE:

{

// Создаем шрифт с дефолтными параметрами

LOGFONT lf;

ZeroMemory(&lf, sizeof(LOGFONT));

lf.lfHeight = g\_fontSize;

lf.lfWeight = FW\_NORMAL;

lstrcpy(lf.lfFaceName, g\_FontFace);

hDefaultFont = CreateFontIndirect(&lf);

// Создание редактора текста

g\_hEdit = CreateWindowEx(

WS\_EX\_CLIENTEDGE,

L"EDIT",

NULL,

WS\_CHILD | WS\_VISIBLE | WS\_VSCROLL | ES\_MULTILINE | ES\_AUTOVSCROLL,

0, 0, 0, 0,

hwnd,

(HMENU)ID\_EDIT,

(HINSTANCE)GetWindowLongPtr(hwnd, GWLP\_HINSTANCE),

NULL

);

SendMessage(g\_hEdit, WM\_SETFONT, reinterpret\_cast<WPARAM>(hDefaultFont), MAKELPARAM(TRUE, 0));

// Создание файла и отображение его в память

g\_hFileMapping = CreateFileMapping(INVALID\_HANDLE\_VALUE, NULL, PAGE\_READWRITE, 0, 1024, L"MyMappedFile");

if (!g\_hFileMapping)

{

MessageBox(g\_hMainWindow, L"Failed to create file mapping", L"Error", MB\_OK | MB\_ICONERROR);

return -1;

}

g\_lpFileBase = MapViewOfFile(g\_hFileMapping, FILE\_MAP\_ALL\_ACCESS, 0, 0, 0);

if (!g\_lpFileBase)

{

MessageBox(g\_hMainWindow, L"Failed to map view of file", L"Error", MB\_OK | MB\_ICONERROR);

CloseHandle(g\_hFileMapping);

return -1;

}

break;

}

case WM\_SIZE:

{

// Перерасчет размеров редактора текста при изменении размеров окна

int nWidth = LOWORD(lParam);

int nHeight = HIWORD(lParam);

MoveWindow(g\_hEdit, 0, 0, nWidth, nHeight, TRUE);

break;

}

case WM\_COMMAND:

{

if (LOWORD(wParam) == ID\_BACKGROUND\_COLOR)

{

ChangeBgc(hwnd);

}

if (LOWORD(wParam) == ID\_TEXT\_COLOR)

{

ChangeTextColor(hwnd);

}

// Обработка командных сообщений

int wmId = LOWORD(wParam);

switch (wmId)

{

case ID\_FILE\_NEW:

CreateNewDocument();

break;

case ID\_FILE\_OPEN:

OpenDocument();

break;

case ID\_FILE\_SAVE:

SaveDocument();

break;

case ID\_NEW\_WINDOW:

CreateNeWindow();

break;

case ID\_STYLE\_BOLD:

g\_bBold = !g\_bBold;

UpdateTextStyle(g\_hEdit);

break;

case ID\_STYLE\_ITALIC:

g\_bItalic = !g\_bItalic;

UpdateTextStyle(g\_hEdit);

break;

case ID\_STYLE\_UNDERLINE:

g\_bUnderline = !g\_bUnderline;

UpdateTextStyle(g\_hEdit);

break;

case ID\_FONT\_FACE\_ARIAL:

g\_FontFace = L"Arial";

UpdateTextStyle(g\_hEdit);

break;

case ID\_FONT\_FACE\_CALIBRI:

g\_FontFace = L"Calibri";

UpdateTextStyle(g\_hEdit);

break;

case ID\_FONT\_FACE\_TNR:

g\_FontFace = L"Times New Roman";

UpdateTextStyle(g\_hEdit);

break;

case ID\_FONT\_SIZE\_10:

g\_fontSize = 10;

UpdateTextStyle(g\_hEdit);

break;

case ID\_FONT\_SIZE\_12:

g\_fontSize = 12;

UpdateTextStyle(g\_hEdit);

break;

case ID\_FONT\_SIZE\_14:

g\_fontSize = 14;

UpdateTextStyle(g\_hEdit);

break;

case ID\_FONT\_SIZE\_16:

g\_fontSize = 16;

UpdateTextStyle(g\_hEdit);

break;

case ID\_FONT\_SIZE\_18:

g\_fontSize = 18;

UpdateTextStyle(g\_hEdit);

break;

case ID\_FONT\_SIZE\_20:

g\_fontSize = 20;

UpdateTextStyle(g\_hEdit);

break;

case ID\_BIN\_SAVE:

WriteToFile();

break;

case ID\_BIN\_UPLOAD:

ReadFromFile();

break;

}

break;

}

case WM\_CTLCOLOREDIT:

{

HDC hdcEdit = (HDC)wParam;

SetTextColor(hdcEdit, textColor);

SetBkColor(hdcEdit, bgColor);

return reinterpret\_cast<LRESULT>(hEditBgBrush);

}

case WM\_CLOSE:

{

// Если пользователь соглашается сохранить изменения, закройте текущее окно

if (MessageBox(hwnd, L"Do you want to save the changes?", L"Save Changes", MB\_YESNO | MB\_ICONQUESTION) == IDYES)

{

SaveDocument();

}

DestroyWindow(hwnd);

return 0;

}

case WM\_DESTROY:

{

if (g\_lpFileBase != NULL)

{

UnmapViewOfFile(g\_lpFileBase);

g\_lpFileBase = NULL;

}

if (g\_hFileMapping != NULL)

{

CloseHandle(g\_hFileMapping);

g\_hFileMapping = NULL;

}

// Обработка сообщения разрушения окна

PostQuitMessage(0);

break;

}

}

return DefWindowProc(hwnd, uMsg, wParam, lParam);

}

void CreateNewDocument()

{

g\_nDocCount++;

g\_szCurrentFile[0] = TEXT('\0');

SetWindowText(g\_hMainWindow, L"Text Editor - New Document");

SetWindowText(g\_hEdit, L"");

}

// Открытие документа

void OpenDocument()

{

OPENFILENAME ofn = {};

TCHAR szFileName[MAX\_PATH] = TEXT("");

ofn.lStructSize = sizeof(OPENFILENAME);

ofn.hwndOwner = g\_hMainWindow;

ofn.lpstrFilter = TEXT("Text Files (\*.txt)\0\*.txt\0All Files (\*.\*)\0\*.\*\0");

ofn.lpstrFile = szFileName;

ofn.nMaxFile = MAX\_PATH;

ofn.Flags = OFN\_FILEMUSTEXIST | OFN\_PATHMUSTEXIST;

if (GetOpenFileName(&ofn))

{

// Открытие файла в текущем окне

g\_nDocCount++;

lstrcpy(g\_szCurrentFile, szFileName);

SetWindowText(g\_hMainWindow, g\_szCurrentFile);

HANDLE hFile = CreateFile(g\_szCurrentFile, GENERIC\_READ, 0, NULL, OPEN\_EXISTING, FILE\_ATTRIBUTE\_NORMAL, NULL);

if (hFile != INVALID\_HANDLE\_VALUE)

{

DWORD dwFileSize = GetFileSize(hFile, NULL);

if (dwFileSize != INVALID\_FILE\_SIZE)

{

LPSTR lpFileData = (LPSTR)GlobalAlloc(GPTR, dwFileSize + 1);

if (lpFileData != NULL)

{

DWORD dwBytesRead;

if (ReadFile(hFile, lpFileData, dwFileSize, &dwBytesRead, NULL))

{

lpFileData[dwBytesRead] = '\0';

SetWindowTextA(g\_hEdit, lpFileData);

}

GlobalFree(lpFileData);

}

}

CloseHandle(hFile);

}

}

}

// Сохранение документа

void SaveDocument()

{

if (g\_szCurrentFile[0] == TEXT('\0'))

{

// Если текущий файл пуст, вызываем диалог сохранения файла

OPENFILENAME ofn = {};

TCHAR szFileName[MAX\_PATH] = TEXT("");

ofn.lStructSize = sizeof(OPENFILENAME);

ofn.hwndOwner = g\_hMainWindow;

ofn.lpstrFilter = TEXT("Text Files (\*.txt)\0\*.txt\0All Files (\*.\*)\0\*.\*\0");

ofn.lpstrFile = szFileName;

ofn.nMaxFile = MAX\_PATH;

ofn.Flags = OFN\_OVERWRITEPROMPT;

if (GetSaveFileName(&ofn))

{

lstrcpy(g\_szCurrentFile, szFileName);

SetWindowText(g\_hMainWindow, g\_szCurrentFile);

}

else

{

return;

}

}

HANDLE hFile = CreateFile(g\_szCurrentFile, GENERIC\_WRITE, 0, NULL, CREATE\_ALWAYS, FILE\_ATTRIBUTE\_NORMAL, NULL);

if (hFile != INVALID\_HANDLE\_VALUE)

{

int nLength = GetWindowTextLength(g\_hEdit);

LPSTR lpFileData = (LPSTR)GlobalAlloc(GPTR, nLength + 1);

if (lpFileData != NULL)

{

GetWindowTextA(g\_hEdit, lpFileData, nLength + 1);

DWORD dwBytesWritten;

WriteFile(hFile, lpFileData, nLength, &dwBytesWritten, NULL);

GlobalFree(lpFileData);

}

CloseHandle(hFile);

}

}

// Обновление заголовка окна

void UpdateWindowTitle()

{

TCHAR szTitle[MAX\_PATH];

if (g\_szCurrentFile[0] == TEXT('\0'))

{

wsprintf(szTitle, TEXT("Text Editor - New Document %d"), g\_nDocCount);

}

else

{

wsprintf(szTitle, TEXT("Text Editor - %s"), g\_szCurrentFile);

}

SetWindowText(g\_hMainWindow, szTitle);

}

void CreateNeWindow() {

STARTUPINFO si = {};

PROCESS\_INFORMATION pi = {};

// Создание нового процесса с текущим исполняемым файлом

if (CreateProcess(NULL, GetCommandLine(), NULL, NULL, FALSE, 0, NULL, NULL, &si, &pi))

{

CloseHandle(pi.hThread);

CloseHandle(pi.hProcess);

}

}

void UpdateTextStyle(HWND hWnd)

{

LOGFONT lf;

ZeroMemory(&lf, sizeof(LOGFONT));

lf.lfHeight = g\_fontSize; // высота шрифта

if (g\_bBold){lf.lfWeight = FW\_BOLD;}

if (g\_bItalic) {

lf.lfItalic = TRUE; // курсив

}

if (g\_bUnderline) {

lf.lfUnderline = TRUE; // подчеркивание

}

lstrcpy(lf.lfFaceName, g\_FontFace); // имя шрифта

HFONT hFont = CreateFontIndirect(&lf);

SendMessage(hWnd, WM\_SETFONT, reinterpret\_cast<WPARAM>(hFont), MAKELPARAM(TRUE, 0));

// Обновляем окно

UpdateWindow(hWnd);

}

void ChangeBgc(HWND hwnd) {

// Открываем диалоговое окно цвета

CHOOSECOLOR cc;

ZeroMemory(&cc, sizeof(CHOOSECOLOR));

static COLORREF customColors[16] = { 0 };

cc.lStructSize = sizeof(CHOOSECOLOR);

cc.hwndOwner = hwnd;

cc.rgbResult = bgColor;

cc.lpCustColors = customColors;

cc.Flags = CC\_FULLOPEN | CC\_RGBINIT;

if (ChooseColor(&cc))

{

bgColor = cc.rgbResult;

hEditBgBrush = CreateSolidBrush(cc.rgbResult);

InvalidateRect(hwnd, NULL, TRUE);

}

}

void ChangeTextColor(HWND hwnd) {

// Открываем диалоговое окно цвета

CHOOSECOLOR cc;

ZeroMemory(&cc, sizeof(CHOOSECOLOR));

static COLORREF customColors[16] = { 0 };

cc.lStructSize = sizeof(CHOOSECOLOR);

cc.hwndOwner = hwnd;

cc.rgbResult = textColor;

cc.lpCustColors = customColors;

cc.Flags = CC\_FULLOPEN | CC\_RGBINIT;

if (ChooseColor(&cc))

{

textColor = cc.rgbResult;

hEditTextColorBrush = CreateSolidBrush(cc.rgbResult);

InvalidateRect(hwnd, NULL, TRUE);

}

}

// Функция-перехватчик клавиатуры

LRESULT CALLBACK KeyboardProc(int nCode, WPARAM wParam, LPARAM lParam)

{

if (nCode >= 0)

{

// Извлечение информации о событии клавиатуры

KBDLLHOOKSTRUCT\* pKeyboardStruct = (KBDLLHOOKSTRUCT\*)lParam;

// Проверка типа события

if (wParam == WM\_KEYDOWN)

{

// Проверка нажатия комбинации клавиш Ctrl+N

if ((pKeyboardStruct->vkCode == 'N') && (GetAsyncKeyState(VK\_CONTROL) & 0x8000))

{

// Выполнение действия по созданию нового файла

CreateNewDocument();

}

else if ((pKeyboardStruct->vkCode == 'O') && (GetAsyncKeyState(VK\_CONTROL) & 0x8000)) {

OpenDocument();

}

else if ((pKeyboardStruct->vkCode == 'S') && (GetAsyncKeyState(VK\_CONTROL) & 0x8000)) {

SaveDocument();

}

}

}

// Передача события дальше в цепочку хуков

return CallNextHookEx(NULL, nCode, wParam, lParam);

}

// Функция для записи данных в файл

void WriteToFile()

{

if (g\_szCurrentFile[0] == TEXT('\0'))

{

// Если текущий файл пуст, вызываем диалог сохранения файла

OPENFILENAME ofn = {};

TCHAR szFileName[MAX\_PATH] = TEXT("");

ofn.lStructSize = sizeof(OPENFILENAME);

ofn.hwndOwner = g\_hMainWindow;

ofn.lpstrFilter = TEXT("Text Files (\*.bin)\0\*.bin\0All Files (\*.\*)\0\*.\*\0");

ofn.lpstrFile = szFileName;

ofn.nMaxFile = MAX\_PATH;

ofn.Flags = OFN\_OVERWRITEPROMPT;

if (GetSaveFileName(&ofn))

{

lstrcpy(g\_szCurrentFile, szFileName);

SetWindowText(g\_hMainWindow, g\_szCurrentFile);

}

else

{

return;

}

}

HANDLE hFile = CreateFile(g\_szCurrentFile, GENERIC\_WRITE, 0, NULL, CREATE\_ALWAYS, FILE\_ATTRIBUTE\_NORMAL, NULL);

if (hFile != INVALID\_HANDLE\_VALUE)

{

int nLength = GetWindowTextLength(g\_hEdit);

LPSTR lpFileData = (LPSTR)GlobalAlloc(GPTR, nLength + 1);

if (lpFileData != NULL)

{

GetWindowTextA(g\_hEdit, lpFileData, nLength + 1);

DWORD dwBytesWritten;

WriteFile(hFile, lpFileData, nLength, &dwBytesWritten, NULL);

GlobalFree(lpFileData);

}

CloseHandle(hFile);

}

}

// Функция для чтения данных из файла

void ReadFromFile()

{

OPENFILENAME ofn;

TCHAR szFileName[MAX\_PATH] = TEXT("");

ZeroMemory(&ofn, sizeof(ofn));

ofn.lStructSize = sizeof(ofn);

ofn.hwndOwner = g\_hMainWindow;

ofn.lpstrFilter = L"Binary Files (\*.bin)\0\*.bin\0All Files (\*.\*)\0\*.\*\0";

ofn.lpstrFile = szFileName;

ofn.nMaxFile = MAX\_PATH;

ofn.Flags = OFN\_FILEMUSTEXIST | OFN\_PATHMUSTEXIST;

if (GetOpenFileName(&ofn))

{

g\_nDocCount++;

lstrcpy(g\_szCurrentFile, szFileName);

SetWindowText(g\_hMainWindow, g\_szCurrentFile);

HANDLE hFile = CreateFile(ofn.lpstrFile, GENERIC\_READ, FILE\_SHARE\_READ, NULL, OPEN\_EXISTING, FILE\_ATTRIBUTE\_NORMAL, NULL);

if (hFile != INVALID\_HANDLE\_VALUE)

{

DWORD fileSize = GetFileSize(hFile, NULL);

// Обновление размера представления файла в памяти

UnmapViewOfFile(g\_lpFileBase);

CloseHandle(g\_hFileMapping);

g\_hFileMapping = CreateFileMapping(hFile, NULL, PAGE\_READONLY, 0, fileSize, L"SharedFileMapping");

if (g\_hFileMapping)

{

LPSTR pFileDataA = (LPSTR)MapViewOfFile(g\_hFileMapping, FILE\_MAP\_READ, 0, 0, fileSize);

//g\_lpFileBase = (LPSTR)MapViewOfFile(g\_hFileMapping, FILE\_MAP\_READ, 0, 0, fileSize);

if (g\_lpFileBase)

{

int wideLength = MultiByteToWideChar(CP\_UTF8, 0, pFileDataA, -1, NULL, 0);

//UnmapViewOfFile(g\_lpFileBase);

g\_lpFileBase = new WCHAR[wideLength];

MultiByteToWideChar(CP\_UTF8, 0, pFileDataA, -1, (LPWSTR)g\_lpFileBase, wideLength);

WCHAR\* pszText = (WCHAR\*)g\_lpFileBase;

SetWindowText(g\_hEdit, pszText);

// Перерисовка окна

InvalidateRect(g\_hMainWindow, NULL, TRUE);

}

else

{

MessageBox(g\_hMainWindow, L"Failed to map view of file", L"Error", MB\_OK | MB\_ICONERROR);

UnmapViewOfFile(g\_lpFileBase);

CloseHandle(g\_hFileMapping);

}

}

else

{

MessageBox(g\_hMainWindow, L"Failed to create file mapping", L"Error", MB\_OK | MB\_ICONERROR);

UnmapViewOfFile(g\_lpFileBase);

CloseHandle(g\_hFileMapping);

}

CloseHandle(hFile);

}

else

{

MessageBox(g\_hMainWindow, L"Failed to open file", L"Error", MB\_OK | MB\_ICONERROR);

}

}

}