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 №3

на тему

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

Выполнил:

студент группы 153503

Скворцов А. В.

Проверил:

Гриценко Н. Ю.

Минск 2023

СОДЕРЖАНИЕ

[1 Цель работы 3](#_Toc146728177)

[2 Краткие теоретические сведения 4](#_Toc146728178)

[3 Результаты выполнения лабораторной работы 5](#_Toc146728179)

[Выводы 7](#_Toc146728180)

[Список использованных источников 8](#_Toc146728181)

[Приложение А (обязательное) Листинг кода 9](#_Toc146728182)

## 1 ЦЕЛЬ РАБОТЫ

## Целью лабораторной работы является приобретение практических навыков по управлению памятью и вводом-выводом в операционной системе Windows, а также понимание принципов, концепций и возможностей, связанных с расширенными функциями ввода-вывода и API подсистемы памяти Win32. Отображение файлов в память позволяет создавать виртуальное представление файла в адресном пространстве процесса, что позволяет обращаться к содержимому файла так, будто он находится в памяти. Это особенно полезно при работе с большими файлами или при необходимости быстрого доступа к данным в файле. Для работы с отображением файлов в память в Windows использовались функции API, такие как CreateFileMapping, MapViewOfFile, UnmapViewOfFile и другие, которые предоставляют доступ к функциональности отображения файлов в память.2 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Управление памятью и вводом-выводом являются важными аспектами разработки программного обеспечения. В данной лабораторной работе рассмотрены расширенные возможности ввода-вывода в операционной системе Windows, а также функции API подсистемы памяти Win32. Windows предоставляет набор функций API, которые позволяют управлять памятью и вводом-выводом. Некоторые из них включают функции для работы с файлами, создания, открытия, чтения и записи файлов, а также функции для работы с памятью, такие как выделение памяти, копирование данных и управление виртуальной памятью.

Функции API подсистемы памяти Win32 позволяют программистам работать с памятью, включая выделение и освобождение памяти, копирование данных, изменение прав доступа к памяти и т.д. Эти функции обеспечивают гибкость и контроль над памятью в приложении. Отображение файлов в памяти, также известное как файловый маппинг, является механизмом операционной системы, позволяющим работать с содержимым файла как с областью памяти. Этот механизм предоставляет удобный и эффективный способ чтения и записи данных в файлы. Процесс отображения файла в память начинается с создания файлового объекта в операционной системе. Затем этот файловый объект отображается в виртуальную память процесса, что позволяет программам обращаться к содержимому файла, как будто оно находится в оперативной памяти. Одним из основных преимуществ отображения файлов в память является то, что доступ к данным в файле осуществляется через указатели и операции чтения/записи в память, что может быть более эффективным, чем использование стандартных операций чтения и записи файлов. Для работы с отображением файлов в память в Windows можно использовать функции API, такие как CreateFileMapping, MapViewOfFile и UnmapViewOfFile. Эти функции позволяют создавать файловые объекты, отображать их в память и освобождать память, связанную с отображением.

В целом, отображение файлов в память представляет собой мощный механизм для эффективной работы с файлами и обмена данными между процессами в операционной системе Windows.

## 3 РЕЗУЛЬТАТЫ ВЫПОЛНЕНИЯ ЛАБОРАТОРНОЙ РАБОТЫ

В ходе выполнения лабораторной работы был разработан текстовый редактор, напоминающий классическое приложение «Блокнот». Приложение позволяет пользователю создать текстовый файл и печатать текст с клавиатуры (рисунок 1).

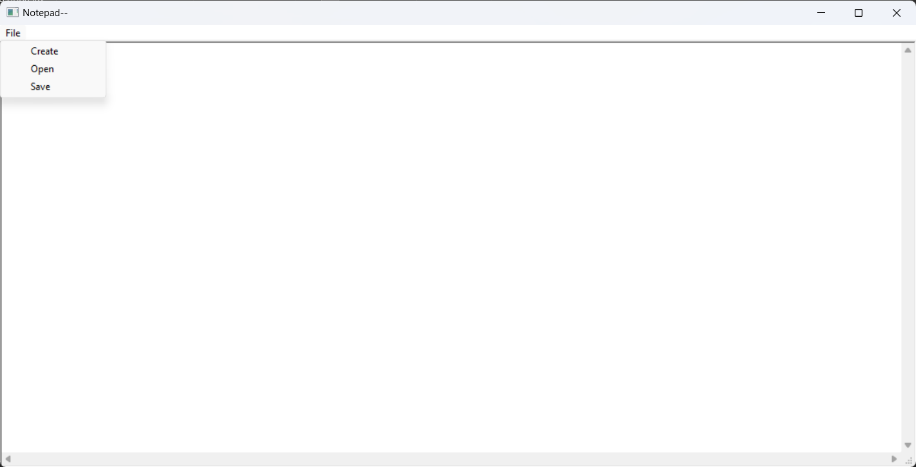


Рисунок 1 - Главный экран

Пользователь может открыть файл, чтобы просмотреть его содержимое и при необходимости отредактировать. Открытие файла происходит посредством отображения файла в памяти и, соответственно, чтение информации из него происходит благодаря данному механизму (рисунок 2).

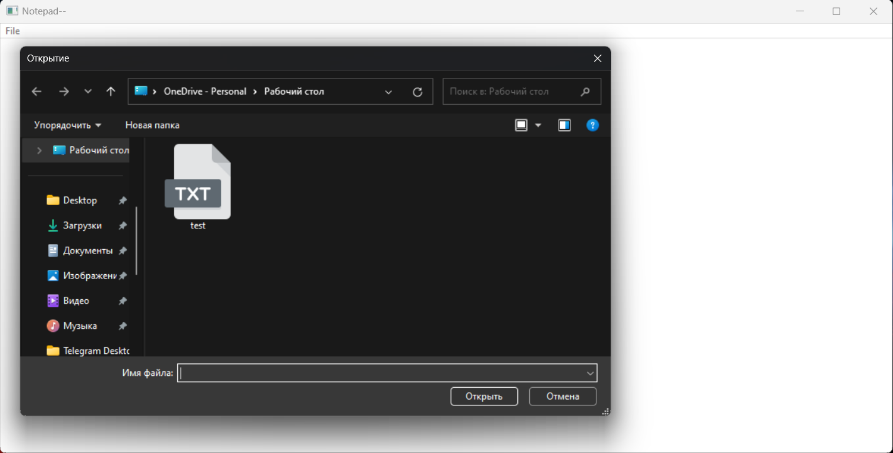


Рисунок 2 ─ Открытие файла

Пользователю предоставляется возможность сохранить свой файл. Сохранение файла так же, как и его чтение, осуществляется при помощи маппинга (рисунок 3).

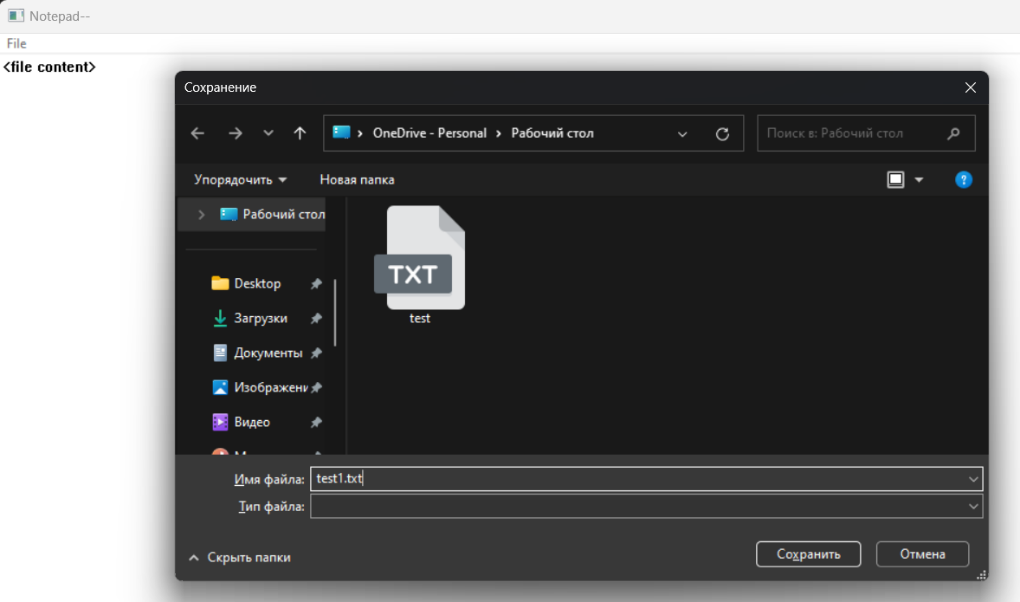


Рисунок 3 – Сохранение файла

## ВЫВОДЫ

В ходе выполнения данной лабораторной работы был разработан текстовый редактор с использованием механизма отображения файла в память. Этот механизм позволил эффективно работать с содержимым файла, как будто оно находилось в оперативной памяти. Отображение файла в память позволило использовать указатели и операции чтения/записи в память для доступа к данным в файле. Это оказалось более эффективным и удобным способом чтения и записи данных, поскольку не требовалось использовать стандартные операции чтения и записи файлов. В итоге, разработка текстового редактора с использованием отображения файла в память оказалась успешной. Получилось эффективное и удобное средство для работы с файлами, обеспечивающее высокую производительность.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Щупак Ю. Win32 API. Разработка приложений для Windows. ─ СПб: Питер, 2008. ─ 592 с.: ил.
2. Создание классических приложений для Windows с использованием API Win32 [Электронный ресурс]. ─ Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/api ─ Дата доступа 19.09.2023

## ПРИЛОЖЕНИЕ А

## (обязательное)

## Листинг кода

Листинг 1 – Файл main.cpp

#ifndef UNICODE

#define UNICODE

#endif

#define ID\_FILE\_CREATE 9001

#define ID\_FILE\_OPEN 9002

#define ID\_FILE\_SAVE 9003

#define VK\_A 0x41

#define ID\_FONT\_CHOICE 9004

#define ID\_BG\_CHOICE 9005

// Макросы для распознавания конкретной команды из MenuBar'а

#include <windows.h>

#include <shobjidl.h>

#include <iostream>

#include <optional>

#include <utility>

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

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

std::optional<std::tuple<HANDLE, HANDLE, LPVOID>> InitializeMapping(PWSTR pszFilePath, DWORD openFlag, DWORD dwFileSize = NULL);

void UnitializeMapping(HANDLE hFile, HANDLE hMapFile, LPVOID lpData);

void OpenFile(HWND hwnd);

void SaveFile(HWND hwnd);

void FontChoice(HWND hwnd);

void BackgroundColor(HWND hwnd);

HWND hWndEdit = NULL;

COLORREF hEditFontColor = RGB(0, 0, 0);

HFONT hEditFont = NULL;

COLORREF hEditBackgroundColor = RGB(255, 255, 255);

int WINAPI wWinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, PWSTR pCmdLine, int nCmdShow) {

const wchar\_t CLASS\_NAME[] = L"Sample Window Class";

WNDCLASS wc = { };

HHOOK hHook = SetWindowsHookEx(WH\_KEYBOARD\_LL, HotkeyProc, NULL, 0);

wc.lpfnWndProc = WindowProc;

wc.hInstance = hInstance;

wc.lpszClassName = CLASS\_NAME;

RegisterClass(&wc);

HWND hwnd = CreateWindowEx(

0,

CLASS\_NAME,

L"Notepad--",

WS\_OVERLAPPEDWINDOW,

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

NULL,

NULL,

hInstance,

NULL

);

if (hwnd == NULL) {

return 0;

}

ShowWindow(hwnd, nCmdShow);

MSG msg = { };

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

TranslateMessage(&msg);

DispatchMessage(&msg);

}

UnhookWindowsHookEx(hHook);

return 0;

}

UINT\_PTR CALLBACK Lpcchookproc(HWND hWnd, UINT msg, WPARAM wParam, LPARAM lParam) {

if (msg == WM\_INITDIALOG) {

SetWindowPos(hWnd, HWND\_TOPMOST, 400, 400, 0, 0, SWP\_NOSIZE);

}

return 0;

}

LRESULT CALLBACK HotkeyProc(int nCode, WPARAM wParam, LPARAM lParam) {

if (nCode < 0)

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

if (wParam == WM\_KEYDOWN) {

if (nCode >= 0) {

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

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

MessageBox(NULL, L"Вы нажали Ctrl + Shift + S!", L"Горячая клавиша", MB\_OK);

}

}

}

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

}

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

switch (uMsg) {

case WM\_DESTROY:

{

PostQuitMessage(0);

return 0;

}

case WM\_CREATE:

{

HRESULT hr = CoInitializeEx(NULL, COINIT\_APARTMENTTHREADED | COINIT\_DISABLE\_OLE1DDE);

if (!SUCCEEDED(hr)) {

DestroyWindow(hwnd);

return 0;

}

HMENU hMenu, hFileSubMenu, hToolsSubMenu;

// Создание MenuBar

hMenu = CreateMenu();

hFileSubMenu = CreatePopupMenu();

hToolsSubMenu = CreatePopupMenu();

AppendMenu(hFileSubMenu, MF\_STRING, ID\_FILE\_CREATE, L"&Create");

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

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

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

AppendMenu(hToolsSubMenu, MF\_STRING, ID\_FONT\_CHOICE, L"&Font");

AppendMenu(hToolsSubMenu, MF\_STRING, ID\_BG\_CHOICE, L"&Background");

AppendMenu(hMenu, MF\_POPUP, (UINT)hToolsSubMenu, L"&Tools");

SetMenu(hwnd, hMenu);

hWndEdit = CreateWindowEx(WS\_EX\_CLIENTEDGE, TEXT("Edit"), NULL,

WS\_CHILD | WS\_VSCROLL | WS\_HSCROLL | ES\_AUTOHSCROLL | ES\_AUTOVSCROLL | ES\_LEFT | ES\_MULTILINE | ES\_WANTRETURN | WS\_VISIBLE,

0, 0, 780, 560, hwnd, NULL,

(HINSTANCE)GetWindowLong(hwnd, GWL\_HINSTANCE),

NULL);

hEditFont = (HFONT)SendMessage(hWndEdit, WM\_GETFONT, 0, 0);

RegisterHotKey(hwnd, 1, MOD\_CONTROL, VK\_A);

return 0;

}

case WM\_COMMAND:

{

switch (LOWORD(wParam)) {

// Обработка сообщений из нажатых кнопок в меню

case ID\_FILE\_CREATE:

SetWindowTextA(hWndEdit, "");

break;

case ID\_FILE\_OPEN:

OpenFile(hwnd);

break;

case ID\_FILE\_SAVE:

SaveFile(hwnd);

break;

case ID\_FONT\_CHOICE:

FontChoice(hwnd);

break;

case ID\_BG\_CHOICE:

BackgroundColor(hwnd);

break;

default:

break;

}

return 0;

}

case WM\_PAINT:

{

PAINTSTRUCT ps;

HDC hdc = BeginPaint(hwnd, &ps);

FillRect(hdc, &ps.rcPaint, (HBRUSH)(COLOR\_WINDOW + 1));

EndPaint(hwnd, &ps);

return 0;

}

case WM\_SIZE:

{

// Растягивание окна текстового редактора совместно с растягиванием основного окна

MoveWindow(hWndEdit, 0, 0, LOWORD(lParam), HIWORD(lParam), false);

return 0;

}

case WM\_CTLCOLOREDIT:

{

HDC hdc = (HDC)wParam;

SetTextColor(hdc, hEditFontColor);

SetDCBrushColor(hdc, hEditBackgroundColor);

SetBkColor(hdc, hEditBackgroundColor);

return (LRESULT)GetStockObject(DC\_BRUSH);

}

case WM\_HOTKEY:

{

if (wParam == 1) {

SendMessage(hWndEdit, EM\_SETSEL, 0, GetWindowTextLength(hWndEdit));

}

break;

}

case WM\_CLOSE:

{

CoUninitialize();

DestroyWindow(hwnd);

return 0;

}

return 0;

}

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

}

void OpenFile(HWND hwnd) {

IFileOpenDialog\* pFileOpen;

HRESULT hr = CoCreateInstance(CLSID\_FileOpenDialog, NULL, CLSCTX\_ALL,

IID\_IFileOpenDialog, reinterpret\_cast<void\*\*>(&pFileOpen));

if (!SUCCEEDED(hr)) {

return;

}

hr = pFileOpen->Show(NULL);

if (!SUCCEEDED(hr)) {

return;

}

IShellItem\* pItem;

hr = pFileOpen->GetResult(&pItem);

if (!SUCCEEDED(hr)) {

pItem->Release();

pFileOpen->Release();

return;

}

PWSTR pszFilePath;

hr = pItem->GetDisplayName(SIGDN\_FILESYSPATH, &pszFilePath);

if (!SUCCEEDED(hr)) {

pItem->Release();

pFileOpen->Release();

return;

}

auto resultInitialize = InitializeMapping(pszFilePath, OPEN\_EXISTING);

if (!resultInitialize) {

return;

}

auto [hFile, hMapFile, lpData] = \*resultInitialize;

DWORD fileSize = GetFileSize(hFile, NULL);

LPSTR buffer = (LPSTR)GlobalAlloc(GPTR, fileSize + 1);

if (memcpy((CHAR\*)buffer, lpData, fileSize)) {

SetWindowTextA(hWndEdit, buffer);

} else {

MessageBoxA(hwnd, "Cannot read file", "Error", MB\_OK);

}

GlobalFree((HGLOBAL)buffer);

ShowWindow(hWndEdit, SW\_SHOW);

SetFocus(hWndEdit);

UnitializeMapping(hFile, hMapFile, lpData);

}

void SaveFile(HWND hwnd) {

IFileSaveDialog\* pFileSave;

HRESULT hr = CoCreateInstance(CLSID\_FileSaveDialog, NULL, CLSCTX\_ALL,

IID\_IFileSaveDialog, reinterpret\_cast<void\*\*>(&pFileSave));

if (!SUCCEEDED(hr)) {

MessageBoxA(hwnd, "File dialog open failed", "Failed", MB\_ICONERROR);

return;

}

hr = pFileSave->Show(NULL);

if (!SUCCEEDED(hr)) {

MessageBoxA(hwnd, "File dialog open show failed", "Failed", MB\_ICONERROR);

return;

}

IShellItem\* pItem;

hr = pFileSave->GetResult(&pItem);

if (!SUCCEEDED(hr)) {

MessageBoxA(hwnd, "Failed to get result from file dialog", "Failed", MB\_ICONERROR);

return;

}

PWSTR pszFilePath;

hr = pItem->GetDisplayName(SIGDN\_FILESYSPATH, &pszFilePath);

if (!SUCCEEDED(hr)) {

MessageBoxA(hwnd, "Failed to get file path", "Failed", MB\_ICONERROR);

return;

}

DWORD dwFileSize = GetWindowTextLength(hWndEdit) + 1;

auto resultInitialize = InitializeMapping(pszFilePath, CREATE\_ALWAYS, dwFileSize);

if (!resultInitialize) {

return;

}

auto [hFile, hMapFile, lpData] = \*resultInitialize;

LPSTR buffer = (LPSTR)GlobalAlloc(GPTR, dwFileSize);;

GetWindowTextA(hWndEdit, buffer, dwFileSize);

if (memcpy((CHAR\*)lpData, buffer, dwFileSize)) {

MessageBoxA(hwnd, "File successfully saved", "Saved", MB\_OK);

} else {

MessageBoxA(hwnd, "File save failed", "Failed", MB\_ICONERROR);

}

UnitializeMapping(hFile, hMapFile, lpData);

GlobalFree((HGLOBAL)buffer);

}

std::optional<std::tuple<HANDLE, HANDLE, LPVOID>> InitializeMapping(PWSTR pszFilePath, DWORD openFlag, DWORD dwFileSize) {

HANDLE hFile = CreateFile(pszFilePath, GENERIC\_READ | GENERIC\_WRITE, 0, NULL, openFlag, FILE\_ATTRIBUTE\_NORMAL, NULL);

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

MessageBoxA(NULL, "Error creating file", "Error", MB\_ICONERROR);

return {};

}

if (dwFileSize == NULL) {

dwFileSize = GetFileSize(hFile, NULL);

}

HANDLE hMapFile = CreateFileMapping(hFile, NULL, PAGE\_READWRITE, 0, dwFileSize, NULL);

if (hMapFile == NULL) {

MessageBoxA(NULL, "Error creating mapping", "Error", MB\_ICONERROR);

UnitializeMapping(hFile, INVALID\_HANDLE\_VALUE, NULL);

return {};

}

LPVOID lpData = MapViewOfFile(hMapFile, FILE\_MAP\_ALL\_ACCESS, 0, 0, dwFileSize);

if (lpData == NULL) {

MessageBoxA(NULL, "Error creating mapping view", "Error", MB\_ICONERROR);

UnitializeMapping(hFile, hMapFile, NULL);

return {};

}

return { { hFile, hMapFile, lpData} };

}

void UnitializeMapping(HANDLE hFile, HANDLE hMapFile, LPVOID lpData) {

if (lpData != NULL) {

UnmapViewOfFile(lpData);

}

if (hMapFile != INVALID\_HANDLE\_VALUE) {

CloseHandle(hMapFile);

}

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

CloseHandle(hFile);

}

}

void FontChoice(HWND hwnd) {

CHOOSEFONT cf = { sizeof(CHOOSEFONT) };

LOGFONT lf;

GetObject(hEditFont, sizeof(LOGFONT), &lf);

cf.Flags = CF\_EFFECTS | CF\_SCREENFONTS | CF\_INITTOLOGFONTSTRUCT;

cf.hwndOwner = hwnd;

cf.lpLogFont = &lf;

if (!ChooseFont(&cf))

return;

HFONT hf = CreateFontIndirect(&lf);

hEditFontColor = cf.rgbColors;

if (hf) {

SendMessage(hWndEdit, WM\_SETFONT, (WPARAM)hf, TRUE);

SendMessage(hWndEdit, WM\_CTLCOLOREDIT, (WPARAM)hf, TRUE);

}

}

void BackgroundColor(HWND hwnd) {

CHOOSECOLOR chColor;

COLORREF acrCustClr[16];

chColor.lStructSize = sizeof(chColor);

chColor.lpCustColors = (LPDWORD)acrCustClr;

chColor.rgbResult = hEditBackgroundColor;

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

chColor.hwndOwner = NULL;

chColor.lpfnHook = Lpcchookproc;

if (ChooseColor(&chColor)) {

hEditBackgroundColor = chColor.rgbResult;

HBRUSH hBrush = CreateSolidBrush(hEditBackgroundColor);

SetClassLongPtr(hWndEdit, GCLP\_HBRBACKGROUND, reinterpret\_cast<LONG\_PTR>(hBrush));

InvalidateRect(hWndEdit, nullptr, TRUE);

};

}