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 Н. Д. Князев

Преподаватель Н. Ю. Гриценко

Минск 2023

**СОДЕРЖАНИЕ**

[1 Постановка задачи 3](#_6127s8b6t8wy)

[2 Краткие теоретические сведения 4](#_r4y5108cx8yu)

[3 Результат выполнения лабораторной работы](#_jnfz1qbvv5j4) 5

[Выводы](#_ldsbh3e5c0xh) 6

[Список использованных источников 7](#_qto16fco0nf2)

[Приложение А (обязательное) Листинг кода](#_54xlhz7mfhe) 8

# 1 ПОСТАНОВКА ЗАДАЧИ

Целью выполнения данной лабораторной работы является разработка графического приложения с использованием расширенных возможностей ввода-вывода и функций API подсистемы памяти Win32, а также контроля асинхронных операций ввода-вывода и отображения файлов в память. Приложение должно использовать расширенный функционал ввода-вывода для работы с чтением и записью файлов, иметь древовидную структуру, отображающую выбранную директорию, а также уметь открывать файлы, содержащиеся в этой структуре.

# 2 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ ТРЕБОВАНИЯ

Объект файла предоставляет представление ресурса (физического устройства или ресурса, расположенного на физическом устройстве), которым может управлять система ввода-вывода. Как и другие объекты, они обеспечивают общий доступ к ресурсу, имеют имена, защищаются безопасностью на основе объектов и поддерживают синхронизацию. Система ввода-вывода также позволяет выполнять операции чтения с ресурса или записи в нее.

Сопоставленный в памяти файл (или сопоставление файлов) является результатом связывания содержимого файла с частью виртуального адресного пространства процесса. Его можно использовать для совместного использования файла или памяти между двумя или более процессами.

Управление памятью в WinAPI включает в себя эффективное распределение и освобождение ресурсов оперативной памяти в процессе работы приложения. Это важный аспект для эффективного использования памяти, предотвращения утечек и конфликтов, а также обеспечения стабильной работы программы.

Для выполнения данной лабораторной работы, были использованы следующие теоретические сведения и концепции:

* Для асинхронного чтения или записи файлов в Windows используются функции ReadFileEx и WriteFileEx. Они позволяют выполнять операции чтения и записи файлов асинхронно. При асинхронном чтении или записи используются объекты OVERLAPPED для асинхронного управления операциями.
* Для создания древовидной структуры отражающей содержимое директории используется элемент управления TreeView. Элемент TreeView в WinAPI представляет собой графический компонент для отображения иерархических данных в виде древовидной структуры. Для добавления элементов в дерево используется функция TreeView\_InsertItem, которая позволяет пошагово построить структуру дерева, где каждый элемент может иметь связи с другими элементами, обеспечивая навигацию и управление иерархией данных в приложении.
* Для маппинга файла в память в Windows используются функции CreateFileMapping, которая создает объект отображения файла, и MapViewOfFile, которая отображает файл в виртуальное адресное пространство процесса. Сначала вызывается CreateFileMapping, где указывается дескриптор файла и параметры отображения. Затем вызывается MapViewOfFile, где передается дескриптор отображения файла и опции доступа, чтобы отобразить файл в память. После работы с данными в памяти, необходимо вызвать UnmapViewOfFile для освобождения отображения, а CloseHandle для освобождения ресурсов отображения и файла.

# 3 РЕЗУЛЬТАТЫ ВЫПОЛНЕНИЯ ЛАБОРАТОРНОЙ РАБОТЫ

В ходе выполнения лабораторной работы было реализовано графическое приложение для архивации выбранной в диалоговом окне папки, а также извлечения файлов их архива. Две клавиши с соответствующими названиями отвечают за функционал приложения(Рисунок 1).

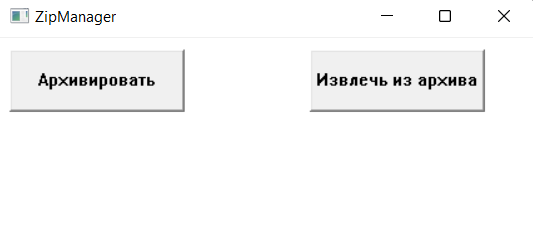


Рисунок 1 – Результат работы программы

# ВЫВОДЫ

В результате выполнения данной лабораторной работы было разработано графическое приложение, которое использует расширенные возможности ввода-вывода и функций API подсистемы памяти Win32. Приложение обладает функционалом для эффективной работы с чтением и записью файлов, а также умеет контролировать асинхронные операции ввода-вывода и отображения файлов в память. Также, оно имеет древовидную структуру, позволяющую пользователю просматривать файлы в выбранной директории, и предоставляет возможность открывать файлы, содержащиеся в этой структуре.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Щупак Ю. Win32 API. Разработка приложений для Windows. – СПб: Питер, 2008. – 592 с.: ип.
2. File management [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/fileio/file-management

# ПРИЛОЖЕНИЕ А

# (обязательное)

# Листинг кода

Листинг 1 – Файл main.cpp

#include "main.h"

int WINAPI wWinMain(HINSTANCE hInstance, HINSTANCE, PWSTR szCmdLine, int nCmdShow)

{

WNDCLASSEX mainWindowClass = CreateMainWindowClass(hInstance);

if (not RegisterClassEx(&mainWindowClass)) {

MessageBoxA(NULL, "Main window hasn't registered", "Error", MB\_OK);

return EXIT\_FAILURE;

}

HWND mainWindow = InstantiateMainWindow(hInstance);

HWND archiveButton = CreateWindow(

L"BUTTON", L"Архивировать",

WS\_TABSTOP | WS\_VISIBLE | WS\_CHILD,

10, 10, BUTTON\_WIDTH, BUTTON\_HEIGHT,

mainWindow,

(HMENU)IDC\_ARCHIVE\_BUTTON,

hInstance,

NULL

);

HWND extractButton = CreateWindow(

L"BUTTON", L"Извлечь из архива",

WS\_TABSTOP | WS\_VISIBLE | WS\_CHILD,

250, 10, BUTTON\_WIDTH, BUTTON\_HEIGHT,

mainWindow,

(HMENU)IDC\_EXTRACT\_BUTTON,

hInstance,

NULL

);

ShowWindow(mainWindow, nCmdShow);

UpdateWindow(mainWindow);

MSG message;

while (GetMessage(&message, NULL, 0, 0) > 0)

{

TranslateMessage(&message);

DispatchMessage(&message);

}

return message.wParam;

}

LRESULT CALLBACK WindowProc(HWND hWindow, UINT message, WPARAM wParam, LPARAM lParam)

{

switch (message) {

case WM\_COMMAND: {

switch (LOWORD(wParam)) {

case IDC\_ARCHIVE\_BUTTON: {

bool isPicked = PickFolder();

if (isPicked) {

std::wstring folderPath(selectedFolder);

std::wstring zipFilePath = std::wstring(selectedFolder).append(L".zip");

AddFolderToZip(folderPath.c\_str(), zipFilePath.c\_str());

MessageBoxA(hWindow, "Zip created successfully!", "Success", MB\_OK);

}

break;

}

case IDC\_EXTRACT\_BUTTON: {

bool isPicked = PickZipToExtract();

if (isPicked) {

std::string outputFolderPath(selectedZip.begin(), selectedZip.end());

size\_t dotPos = outputFolderPath.find\_last\_of(".");

std::string filePath(selectedZip.begin(), selectedZip.end());

if (dotPos != std::string::npos) {

outputFolderPath = outputFolderPath.substr(0, dotPos);

}

ExtractZip(filePath.c\_str(), outputFolderPath.c\_str());

MessageBoxA(hWindow, "Files extracted successfully!", "Success", MB\_OK);

}

break;

}

}

return 0;

}

case WM\_CLOSE:

DestroyWindow(hWindow);

return 0;

case WM\_DESTROY:

PostQuitMessage(0);

return 0;

default:

return DefWindowProc(hWindow, message, wParam, lParam);

}

}

WNDCLASSEX CreateMainWindowClass(HINSTANCE hInstance)

{

WNDCLASSEX wc{};

wc.cbSize = sizeof(WNDCLASSEX);

wc.lpfnWndProc = WindowProc;

wc.hInstance = hInstance;

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

wc.hbrBackground = (HBRUSH)(COLOR\_WINDOW + 1);

wc.hIconSm = LoadIcon(NULL, IDI\_APPLICATION);

wc.lpszClassName = MAIN\_WINDOW\_CLASS\_NAME;

return wc;

}

HWND InstantiateMainWindow(HINSTANCE hInstance)

{

return CreateWindowEx(

WS\_EX\_CLIENTEDGE,

MAIN\_WINDOW\_CLASS\_NAME, APPLICATION\_NAME, //APPLICATION\_NAME,

WS\_OVERLAPPEDWINDOW,

CW\_USEDEFAULT, CW\_USEDEFAULT, 450, 200,

NULL, NULL, hInstance, NULL

);

}

bool PickFolder()

{

HRESULT hr = CoInitialize(NULL);

if (FAILED(hr)) {

return false;

}

IFileOpenDialog\* pfd;

hr = CoCreateInstance(CLSID\_FileOpenDialog, NULL, CLSCTX\_INPROC\_SERVER, IID\_PPV\_ARGS(&pfd));

if (FAILED(hr)) {

CoUninitialize();

return false;

}

SetFileDialogOptions(pfd);

BrowseFolder(pfd);

pfd->Release();

CoUninitialize();

return true;

}

void SetFileDialogOptions(IFileOpenDialog\* pfd)

{

DWORD dwOptions;

pfd->GetOptions(&dwOptions);

pfd->SetOptions(dwOptions | FOS\_PICKFOLDERS);

}

void BrowseFolder(IFileOpenDialog\* pfd)

{

HRESULT hr = pfd->Show(NULL);

if (FAILED(hr))

return;

IShellItem\* result;

hr = pfd->GetResult(&result);

if (FAILED(hr))

return;

PWSTR selectedFolderPath;

hr = result->GetDisplayName(SIGDN\_DESKTOPABSOLUTEPARSING, &selectedFolderPath);

if (FAILED(hr))

return;

OutputDebugString(selectedFolderPath);

selectedFolder = selectedFolderPath;

}

void AddFileToZip(const wchar\_t\* zipFileName, const wchar\_t\* sourceFileName)

{

struct zip\* archive;

struct zip\_source\* source;

archive = zip\_open(WcharToString(zipFileName).c\_str(), ZIP\_CREATE, NULL);

if (archive == NULL) {

perror("Failed to open zip archive");

return;

}

source = zip\_source\_file(archive, WcharToString(sourceFileName).c\_str(), 0, 0);

if (source == NULL) {

perror("Failed to create zip source from file");

zip\_close(archive);

return;

}

std::wstring filePath = sourceFileName;

size\_t found = filePath.find(selectedFolder + L"\\");

if (found != std::wstring::npos) {

filePath.erase(found, selectedFolder.length() + 1);

}

if (zip\_file\_add(archive, WcharToString(filePath.c\_str()).c\_str(), source, 0) < 0) {

perror("Failed to add file to zip archive");

zip\_source\_free(source);

zip\_close(archive);

return;

}

zip\_close(archive);

}

void AddFolderToZip(const wchar\_t\* folderPath, const wchar\_t\* zipFileName) {

WIN32\_FIND\_DATA findFileData;

wchar\_t searchPattern[MAX\_PATH];

swprintf(searchPattern, sizeof(searchPattern) / sizeof(searchPattern[0]), L"%s\\\*", folderPath);

HANDLE hFind = FindFirstFile(searchPattern, &findFileData);

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

do {

if (!(findFileData.dwFileAttributes & FILE\_ATTRIBUTE\_DIRECTORY)) {

wchar\_t filePath[MAX\_PATH];

swprintf(filePath, sizeof(filePath) / sizeof(filePath[0]), L"%s\\%s", folderPath, findFileData.cFileName);

AddFileToZip(zipFileName, filePath);

}

else if (wcscmp(findFileData.cFileName, L".") != 0 && wcscmp(findFileData.cFileName, L"..") != 0) {

wchar\_t subFolderPath[MAX\_PATH];

swprintf(subFolderPath, sizeof(subFolderPath) / sizeof(subFolderPath[0]), L"%s\\%s", folderPath, findFileData.cFileName);

AddFolderToZip(subFolderPath, zipFileName);

}

} while (FindNextFile(hFind, &findFileData) != 0);

FindClose(hFind);

}

}

void ExtractZip(const char\* zipFileName, const char\* outputFolder) {

struct zip\* archive;

archive = zip\_open(zipFileName, ZIP\_RDONLY, NULL);

if (!archive) {

return;

}

int numEntries = zip\_get\_num\_entries(archive, 0);

if (numEntries < 0) {

zip\_close(archive);

return;

}

for (int i = 0; i < numEntries; i++) {

struct zip\_stat entryStat;

if (zip\_stat\_index(archive, i, 0, &entryStat) == 0) {

std::string entryName(entryStat.name);

std::string outputPath;

if (entryName.find("\\") == 0) {

outputPath = std::string(outputFolder) + entryStat.name;

}

else {

outputPath = std::string(outputFolder) + '\\' + entryStat.name;

}

// Ensure the directory exists

size\_t lastSlashPos = outputPath.find\_last\_of("\\");

if (lastSlashPos != std::string::npos) {

std::string dirPath = outputPath.substr(0, lastSlashPos);

SHCreateDirectoryExA(NULL, dirPath.c\_str(), NULL);

}

zip\_file\* zipFile = zip\_fopen\_index(archive, i, 0);

if (zipFile) {

FILE\* outputFile;

if (fopen\_s(&outputFile, outputPath.c\_str(), "wb") == 0) {

char buffer[1024];

int bytesRead;

while ((bytesRead = zip\_fread(zipFile, buffer, sizeof(buffer))) > 0) {

fwrite(buffer, 1, bytesRead, outputFile);

}

fclose(outputFile);

}

zip\_fclose(zipFile);

}

else if (entryStat.name[strlen(entryStat.name) - 1] == '\\') {

// This entry is a directory

CreateDirectoryA(outputPath.c\_str(), NULL);

}

}

}

zip\_close(archive);

}

bool PickZipToExtract()

{

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

if (FAILED(hr)) {

MessageBox(NULL, L"COM Initialization Failed", L"Error", MB\_ICONERROR);

return false;

}

IFileDialog\* pFileDialog = nullptr;

hr = CoCreateInstance(CLSID\_FileOpenDialog, NULL, CLSCTX\_INPROC\_SERVER, IID\_PPV\_ARGS(&pFileDialog));

if (FAILED(hr)) {

MessageBox(NULL, L"Failed to create File Open Dialog", L"Error", MB\_ICONERROR);

CoUninitialize();

return false;

}

COMDLG\_FILTERSPEC fileTypes[] = { L"Zip Archives", L"\*.zip" };

pFileDialog->SetFileTypes(ARRAYSIZE(fileTypes), fileTypes);

hr = pFileDialog->Show(NULL);

if (SUCCEEDED(hr)) {

IShellItem\* pItem;

hr = pFileDialog->GetResult(&pItem);

if (SUCCEEDED(hr)) {

PWSTR filePath;

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

if (SUCCEEDED(hr)) {

std::wstring selectedFilePath(filePath);

selectedZip = selectedFilePath;

CoTaskMemFree(filePath);

}

pItem->Release();

}

}

pFileDialog->Release();

CoUninitialize();

if (FAILED(hr)) {

MessageBox(NULL, L"File Dialog Operation Failed", L"Error", MB\_ICONERROR);

return false;

}

return true;

}

std::string WcharToString(const wchar\_t\* data)

{

std::wstring wideString(data);

return std::string(wideString.begin(), wideString.end());

}