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

ФУНКЦИИ API ПОДСИСТЕМЫ ПАМЯТИ WIN 32.

ОРГАНИЗАЦИЯ И КОНТРОЛЬ АСИНХРОННЫХ

ОПЕРАЦИЙ ВВОДА-ВЫВОДА. ОТОБРАЖЕНИЕ ФАЙЛОВ В ПАМЯТЬ.

Выполнил студент гр.153502 Толстой Д.В.

Проверил ассистент кафедры информатики Гриценко Н.Ю.

Минск 2023

**СОДЕРЖАНИЕ**

[1 Формулировка задачи 3](#_Toc148021362)

[2 Теоретические сведения 4](#_Toc148021363)

[3 Описание функций программы 5](#_Toc148021364)

[3.1 Выбор требуемой операции 5](#_Toc148021365)

[3.2 Выбор требуемого файла или архива 5](#_Toc148021366)

[3.3 Архивация файлов 6](#_Toc148021367)

[3.4 Извлечение файла из архива с требуемым расширением 6](#_Toc148021368)

[Список использованных источников 8](#_Toc148021369)

[Приложение А 9](#_Toc148021370)

1 ФОРМУЛИРОВКА ЗАДАЧИ

Целью выполнения лабораторной работы является построение приложение для сжатия и распаковки файлов, использующее API для работы с архивами

В качестве задачи необходимо создать приложение для архивирование любого файла, а затем последующего разархивирования в файл выбранного расширение.

# **2** ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Архивация – это процесс упаковки одного или нескольких файлов в один архивный файл. За архивацию файлов может быть отвечен формат архива, такой как ZIP, GZIP, TAR и другие.

Сжатие – это процесс уменьшения размера данных. Библиотека zlib предоставляет методы сжатия и распаковки данных с использованием алгоритмов сжатия, таких как Deflate.

zlib – это библиотека для сжатия и распаковки данных. Она предоставляет функции и структуры данных, необходимые для работы с сжатием данных.

В WinAPI можно использовать библиотеку zlib, чтобы архивировать и извлекать файлы. Для этого нам потребуется включить заголовочный файл <zlib.h> и подключить библиотеку zlib к вашему проекту.

В WinAPI для работы с файлами, как правило, используются функции CreateFile, ReadFile, WriteFile, CloseHandle и другие.

При архивации файла вы должны открыть исходный файл для чтения и создать файл архива для записи. После чего вы можете читать данные из исходного файла, сжимать их с использованием zlib и записывать в архив.

При работе с файлами и сжатием данных важно обрабатывать ошибки. Например, мы должны проверять, успешно ли были открыты файлы, были ли успешными операции сжатия и записи данных.

Если мы архивируем файлы, то нам нужно определиться с форматом архива (например, GZIP). Формат архива определяет структуру и методы сжатия данных в архиве.

3 ОПИСАНИЕ ФУНКЦИЙ ПРОГРАММЫ

Согласно формулировке задачи, были спроектированы следующие функции программы:

– Выбор требуемой операции;

– Выбор требуемого файла или архива;

– Архивация файлов;

– Извлечение файла из архива с требуемым расширением;

3.1 Выбор требуемой операции

Для выбора требуемой операции нужно просто нажать на соответствующую кнопку (рисунок 1).

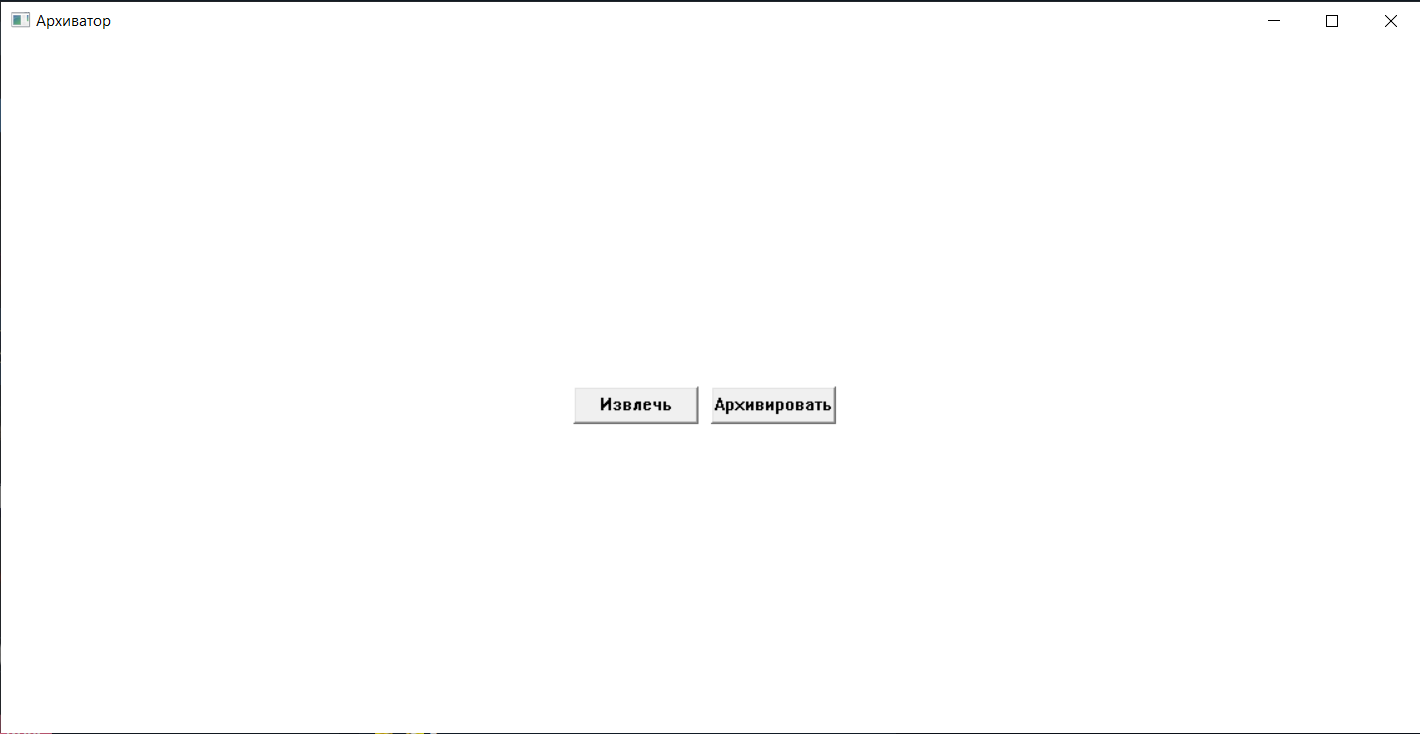


Рисунок 1 – Выбор Операции

3.2 Выбор требуемого файла или архива

Вы можете ввести полный путь до требуемого файла или архива, или же выбрать его в проводнике с помощью кнопки выбрать (рисунок 2).

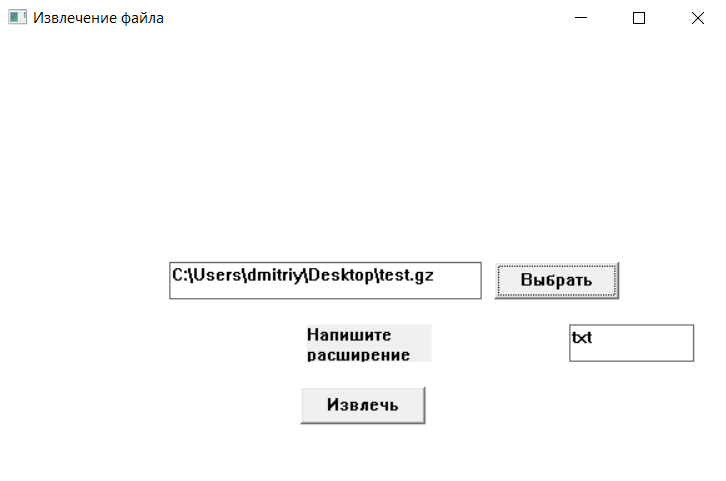


Рисунок 2 – Выбор файла или архива

3.3 Архивация файлов

Выберите архивация, укажите путь до требуемого файла, нажмите архивировать и начнется процесс архивирования этого файла (рисунок 3).

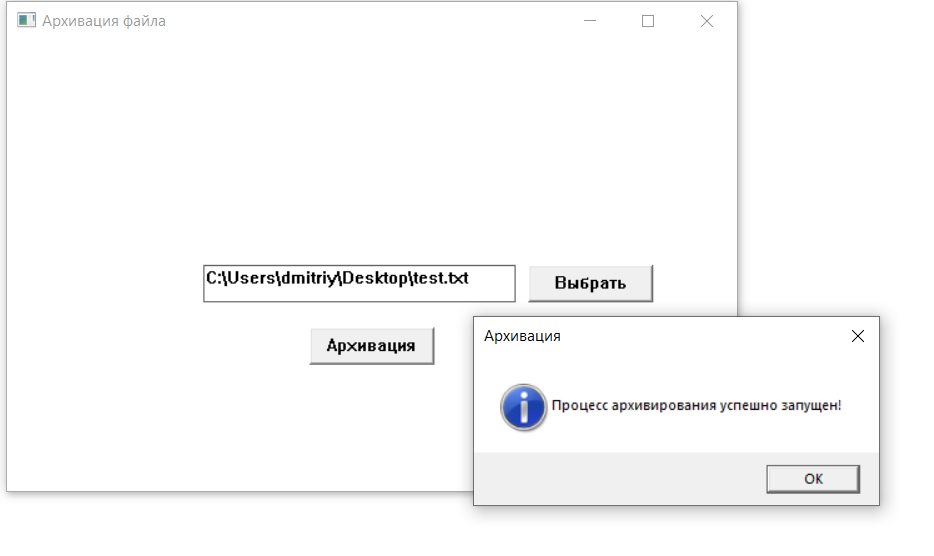


Рисунок 3 –Архивация файла

3.4 Извлечение файла из архива с требуемым расширением

Требуется выбрать путь до архива с расширением .gz, затем выбрать расширение требуемого файла (по умолчанию txt). Запуститься процесс извлечения (рисунок 4).

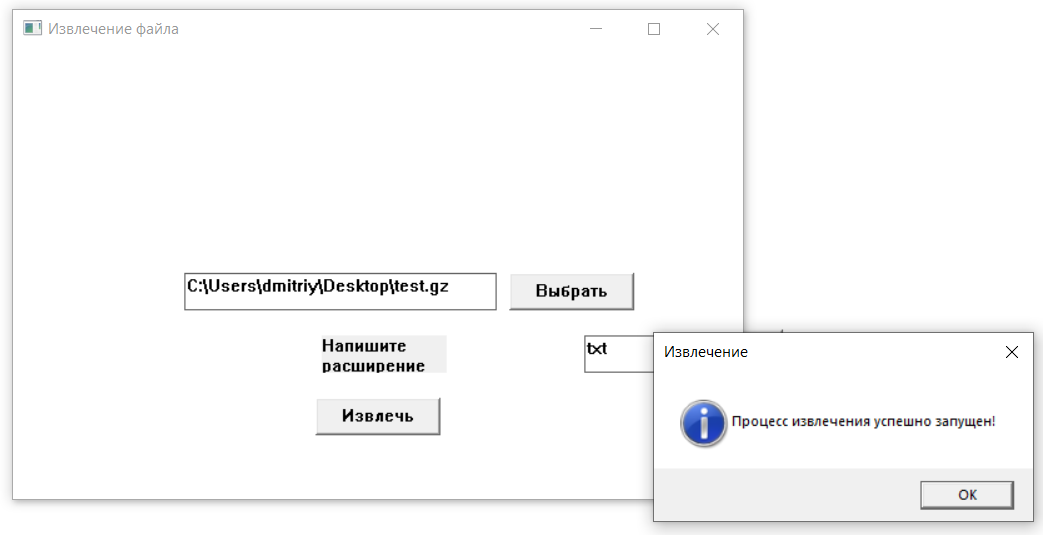


Рисунок 4 – Извлечение файла из архива

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Build desktop Windows apps using the Win32 API [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://learn.microsoft.com/en-us/windows/win32>

[2] C++ WINDOWS API – Электронные данные. – Режим доступа: <https://www.youtube.com/@egordeveloper3794>

ПРИЛОЖЕНИЕ А

Листинг кода

Файл Archiver.cpp

#ifndef UNICODE

#define UNICODE

#endif

#include <windows.h>

#include "Archiver.h"

#include "ExtractWindow.h"

#include "ArchivateWindow.h"

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

{

mainHinstance = hInstance;

// Register the window class.

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

WNDCLASS wc = { };

wc.lpfnWndProc = WindowProc;

wc.hInstance = hInstance;

wc.lpszClassName = CLASS\_NAME;

RegisterClass(&wc);

// Create the window.

HWND hwnd = CreateWindowEx(

0, // Optional window styles.

CLASS\_NAME, // Window class

L"Архиватор", // Window text

WS\_OVERLAPPEDWINDOW, // Window style

// Size and position

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

NULL, // Parent window

NULL, // Menu

hInstance, // Instance handle

NULL // Additional application data

);

mainWindowHwnd = hwnd;

if (hwnd == NULL)

{

return 0;

}

archivateWindow = ArchivateWindow(mainWindowHwnd, mainHinstance);

extractWindow = ExtractWindow(mainWindowHwnd, mainHinstance);

extractButton = CreateWindowW(L"BUTTON", L"Извлечь", WS\_CHILD | WS\_VISIBLE | BS\_PUSHBUTTON,

100, 100, 100, 100, hwnd, (HMENU)extract, hInstance, NULL);

if (extractButton == NULL) {

MessageBox(NULL, L"Создание первой кнопки не удалось!", L"Ошибка", MB\_ICONERROR);

return 1;

}

archiveButton = CreateWindow(L"BUTTON", L"Архивировать", WS\_CHILD | WS\_VISIBLE | BS\_PUSHBUTTON,

250, 100, 100, 100, hwnd, (HMENU)archive, hInstance, NULL);

if (archiveButton == NULL) {

MessageBox(NULL, L"Создание второй кнопки не удалось!", L"Ошибка", MB\_ICONERROR);

return 1;

}

ShowWindow(hwnd, nCmdShow);

// Run the message loop.

MSG msg = { };

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

{

TranslateMessage(&msg);

DispatchMessage(&msg);

}

return 0;

}

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

{

switch (uMsg)

{

case WM\_DESTROY:

{

PostQuitMessage(0);

return 0;

}

case WM\_PAINT:

{

PAINTSTRUCT ps;

HDC hdc = BeginPaint(hwnd, &ps);

// All painting occurs here, between BeginPaint and EndPaint.

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

EndPaint(hwnd, &ps);

break;

}

case WM\_SIZE:

{

int width = LOWORD(lParam);

int height = HIWORD(lParam);

int buttonWidth = 100;

int buttonHeight = 30;

int buttonSpacing = 10;

int x1 = (width - 2 \* buttonSpacing - 2 \* buttonWidth) / 2;

int x2 = x1 + buttonWidth + buttonSpacing;

SetWindowPos(extractButton, NULL, x1, height / 2, buttonWidth, buttonHeight, SWP\_NOZORDER);

SetWindowPos(archiveButton, NULL, x2, height / 2, buttonWidth, buttonHeight, SWP\_NOZORDER);

break;

}

case WM\_COMMAND:

switch (wParam)

{

case extract:

{

ShowWindow(extractWindow.Window(), SW\_SHOW);

break;

}

case archive:

{

ShowWindow(archivateWindow.Window(), SW\_SHOW);

break;

}

default:

break;

}

default:

{

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

}

}

}

Файл BaseWindow.cpp

#include <Windows.h>

template <class DERIVED\_TYPE>

class BaseWindow

{

public:

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

{

DERIVED\_TYPE\* pThis = NULL;

if (uMsg == WM\_NCCREATE)

{

CREATESTRUCT\* pCreate = (CREATESTRUCT\*)lParam;

pThis = (DERIVED\_TYPE\*)pCreate->lpCreateParams;

SetWindowLongPtr(hwnd, GWLP\_USERDATA, (LONG\_PTR)pThis);

pThis->m\_hwnd = hwnd;

}

else

{

pThis = (DERIVED\_TYPE\*)GetWindowLongPtr(hwnd, GWLP\_USERDATA);

}

if (pThis)

{

return pThis->HandleMessage(uMsg, wParam, lParam);

}

else

{

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

}

}

BaseWindow() : m\_hwnd(NULL) { }

BOOL Create(

PCWSTR lpWindowName,

DWORD dwStyle,

DWORD dwExStyle = 0,

int x = CW\_USEDEFAULT,

int y = CW\_USEDEFAULT,

int nWidth = CW\_USEDEFAULT,

int nHeight = CW\_USEDEFAULT,

HWND hWndParent = 0,

HMENU hMenu = 0,

HINSTANCE hInstance = 0

)

{

WNDCLASS wc = { 0 };

wc.lpfnWndProc = DERIVED\_TYPE::WindowProc;

wc.hInstance = hInstance;

wc.lpszClassName = ClassName();

RegisterClass(&wc);

m\_hwnd = CreateWindowEx(

dwExStyle, ClassName(), lpWindowName, dwStyle, x, y,

nWidth, nHeight, hWndParent, hMenu, GetModuleHandle(NULL), this

);

return (m\_hwnd ? TRUE : FALSE);

}

HWND Window() const { return m\_hwnd; }

protected:

virtual PCWSTR ClassName() const = 0;

virtual LRESULT HandleMessage(UINT uMsg, WPARAM wParam, LPARAM lParam) = 0;

HWND m\_hwnd;

};

Файл ArchivateWindow.cpp

#include "ArchivateWindow.h"

#include "Globals.h"

#include <tchar.h>

#include <windows.h>

#include <string>

#include "ArchivateWindow.h"

#include "Globals.h"

#include <tchar.h>

#include <windows.h>

#include <iostream>

#include <string>

ArchivateWindow::ArchivateWindow() { }

ArchivateWindow::ArchivateWindow(HWND parentWindow, HINSTANCE hInstance)

{

m\_parentWindow = parentWindow;

m\_hInstance = hInstance;

this->Create(L"Архивация файла", WS\_OVERLAPPEDWINDOW, NULL,

CW\_USEDEFAULT, CW\_USEDEFAULT, 600, 400, m\_parentWindow, NULL, m\_hInstance);

if (m\_hwnd == NULL)

{

MessageBox(m\_parentWindow, L"Создание нового окна не удалось!", L"Ошибка", MB\_ICONERROR);

}

else

{

m\_hEditPath = CreateWindowW(L"EDIT", L"", WS\_CHILD | WS\_VISIBLE | WS\_BORDER | ES\_AUTOHSCROLL,

200, 185, 200, 30, m\_hwnd, NULL, m\_hInstance, NULL);

if (m\_hEditPath == NULL)

{

MessageBox(m\_parentWindow, L"Создание текстового поля не удалось!", L"Ошибка", MB\_ICONERROR);

}

m\_hButtonChoosePath = CreateWindowW(L"BUTTON", L"Выбрать", WS\_CHILD | WS\_VISIBLE | BS\_PUSHBUTTON,

410, 185, 100, 30, m\_hwnd, (HMENU)choosePath, m\_hInstance, NULL);

if (m\_hButtonChoosePath == NULL)

{

MessageBox(m\_parentWindow, L"Создание кнопки не удалось!", L"Ошибка", MB\_ICONERROR);

}

m\_hButtonArchivate = CreateWindowW(L"BUTTON", L"Архивация", WS\_CHILD | WS\_VISIBLE | BS\_PUSHBUTTON,

250, 225, 100, 30, m\_hwnd, (HMENU)archivate, m\_hInstance, NULL);

if (m\_hButtonArchivate == NULL)

{

MessageBox(m\_parentWindow, L"Создание кнопки не удалось!", L"Ошибка", MB\_ICONERROR);

}

}

}

LRESULT ArchivateWindow::HandleMessage(UINT uMsg, WPARAM wParam, LPARAM lParam)

{

switch (uMsg)

{

case WM\_PAINT:

{

PAINTSTRUCT ps;

HDC hdc = BeginPaint(m\_hwnd, &ps);

// All painting occurs here, between BeginPaint and EndPaint.

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

EndPaint(m\_hwnd, &ps);

break;

}

case WM\_SIZE:

{

int width = LOWORD(lParam);

int height = HIWORD(lParam);

int buttonWidth = 100;

int buttonHeight = 30;

int editWidth = 250;

int editHeight = 30;

int spacing = 10;

int x1 = (width - 2 \* spacing - 2 \* editWidth) / 2 + editWidth / 2;

int x2 = x1 + editWidth + spacing;

int x3 = width / 2 - 50;

SetWindowPos(m\_hEditPath, NULL, x1, height / 2, editWidth, editHeight, SWP\_NOZORDER);

SetWindowPos(m\_hButtonChoosePath, NULL, x2, height / 2, buttonWidth, buttonHeight, SWP\_NOZORDER);

SetWindowPos(m\_hButtonArchivate, NULL, x3, height / 2 + 50, buttonWidth, buttonHeight, SWP\_NOZORDER);

break;

}

case WM\_COMMAND:

{

switch (wParam)

{

case choosePath:

{

lstrcpyn(m\_wfilePath, \_T(""), MAX\_PATH);

OPENFILENAME ofn;

ZeroMemory(&ofn, sizeof(ofn));

ofn.lStructSize = sizeof(ofn);

ofn.hwndOwner = m\_hwnd;

ofn.lpstrFile = m\_wfilePath;

ofn.nMaxFile = MAX\_PATH;

ofn.lpstrFilter = L"All Files\0\*.\*\0";

ofn.nFilterIndex = 1;

ofn.lpstrFileTitle = NULL;

ofn.nMaxFileTitle = 0;

ofn.lpstrInitialDir = NULL;

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

if (GetOpenFileName(&ofn)) {

SetWindowText(m\_hEditPath, m\_wfilePath);

}

break;

}

case archivate:

{

GetWindowText(m\_hEditPath, m\_wfilePath, MAX\_PATH);

char filePath[MAX\_PATH];

size\_t convertedChars = 0;

errno\_t err = wcstombs\_s(&convertedChars, filePath, MAX\_PATH, m\_wfilePath, MAX\_PATH - 1);

if (err != 0) {

MessageBox(m\_hwnd, L"Ошибка конвертации строки", L"Архивация", MB\_ICONERROR);

}

if (!fileExists(m\_wfilePath))

{

MessageBox(m\_hwnd, L"Неверный формат файла", L"Архивация", MB\_ICONERROR);

break;

}

std::string destPath(filePath);

size\_t found = destPath.find\_last\_of(".");

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

destPath = (destPath.substr(0, found)).c\_str();

}

else {

destPath = destPath.c\_str();

}

if (compressFile(m\_hwnd, m\_wfilePath, (destPath + ".gz").c\_str()))

{

MessageBox(m\_hwnd, L"Процесс архивирования успешно запущен!", L"Архивация", MB\_ICONINFORMATION);

}

break;

}

default:

{

break;

}

}

}

case WM\_CLOSE:

{

ShowWindow(m\_hwnd, SW\_HIDE);

break;

}

default:

{

return DefWindowProc(m\_hwnd, uMsg, wParam, lParam);

}

}

}

bool ArchivateWindow::fileExists(LPWSTR filePath)

{

WIN32\_FIND\_DATA findFileData;

HANDLE hFile = FindFirstFile(filePath, &findFileData);

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

FindClose(hFile);

return true;

}

return false;

}

Файл ExtractWindow.cpp

#include "ExtractWindow.h"

#include "Globals.h"

#include <tchar.h>

#include <windows.h>

#include <iostream>

#include <string>

ExtractWindow::ExtractWindow() { }

ExtractWindow::ExtractWindow(HWND parentWindow, HINSTANCE hInstance)

{

m\_parentWindow = parentWindow;

m\_hInstance = hInstance;

this->Create(L"Извлечение файла", WS\_OVERLAPPEDWINDOW, NULL,

CW\_USEDEFAULT, CW\_USEDEFAULT, 600, 400, m\_parentWindow, NULL, m\_hInstance);

if (m\_hwnd == NULL)

{

MessageBox(parentWindow, L"Создание нового окна не удалось!", L"Ошибка", MB\_ICONERROR);

CloseWindow(m\_parentWindow);

}

else

{

m\_hEditPath = CreateWindowW(L"EDIT", L"", WS\_CHILD | WS\_VISIBLE | WS\_BORDER | ES\_AUTOHSCROLL,

200, 185, 200, 30, m\_hwnd, NULL, m\_hInstance, NULL);

if (m\_hEditPath == NULL)

{

MessageBox(m\_parentWindow, L"Создание текстового поля не удалось!", L"Ошибка", MB\_ICONERROR);

CloseWindow(m\_parentWindow);

}

m\_hEditType = CreateWindowW(L"EDIT", L"", WS\_CHILD | WS\_VISIBLE | WS\_BORDER | ES\_AUTOHSCROLL,

250, 240, 100, 30, m\_hwnd, NULL, m\_hInstance, NULL);

if (m\_hEditType == NULL)

{

MessageBox(m\_parentWindow, L"Создание текстового поля не удалось!", L"Ошибка", MB\_ICONERROR);

CloseWindow(m\_parentWindow);

}

SetWindowText(m\_hEditType, L"txt");

m\_hStaticLabel = CreateWindowW(

L"Static", // Класс элемента управления "Static"

L"Напишите расширение файла, которого хотите извлечь", // Текст надписи

WS\_CHILD | WS\_VISIBLE, // Стиль окна

150, 30, // Положение (x, y)

200, 30, // Ширина и высота

m\_hwnd, // Родительское окно

(HMENU)NULL, // Идентификатор элемента управления (может быть NULL)

GetModuleHandle(NULL), // Дескриптор экземпляра приложения

NULL // Дополнительные данные (может быть NULL)

);

m\_hButtonChoosePath = CreateWindowW(L"BUTTON", L"Выбрать", WS\_CHILD | WS\_VISIBLE | BS\_PUSHBUTTON,

410, 185, 100, 30, m\_hwnd, (HMENU)choosePath, m\_hInstance, NULL);

if (m\_hButtonChoosePath == NULL)

{

MessageBox(m\_parentWindow, L"Создание кнопки не удалось!", L"Ошибка", MB\_ICONERROR);

CloseWindow(m\_parentWindow);

}

m\_hButtonExtract = CreateWindowW(L"BUTTON", L"Извлечь", WS\_CHILD | WS\_VISIBLE | BS\_PUSHBUTTON,

250, 275, 100, 30, m\_hwnd, (HMENU)extract, m\_hInstance, NULL);

if (m\_hButtonExtract == NULL)

{

MessageBox(m\_parentWindow, L"Создание кнопки не удалось!", L"Ошибка", MB\_ICONERROR);

CloseWindow(m\_parentWindow);

}

}

}

LRESULT ExtractWindow::HandleMessage(UINT uMsg, WPARAM wParam, LPARAM lParam)

{

switch (uMsg)

{

case WM\_PAINT:

{

PAINTSTRUCT ps;

HDC hdc = BeginPaint(m\_hwnd, &ps);

// All painting occurs here, between BeginPaint and EndPaint.

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

EndPaint(m\_hwnd, &ps);

break;

}

case WM\_SIZE:

{

int width = LOWORD(lParam);

int height = HIWORD(lParam);

int buttonWidth = 100;

int buttonHeight = 30;

int editWidth = 250;

int editTypeWidth = 150;

int editHeight = 30;

int spacing = 10;

int x1 = (width - 2 \* spacing - 2 \* editWidth) / 2 + editWidth / 2 - spacing \* 2;

int x2 = x1 + editWidth + spacing;

int x3 = width / 2 - 50;

int x4 = x2 + 50 + spacing;

int x5\_text = x4 - 200 - spacing;

SetWindowPos(m\_hEditPath, NULL, x1, height / 2, editWidth, editHeight, SWP\_NOZORDER);

SetWindowPos(m\_hButtonChoosePath, NULL, x2, height / 2, buttonWidth, buttonHeight, SWP\_NOZORDER);

SetWindowPos(m\_hEditType, NULL, x4, height / 2 + 50, buttonWidth, buttonHeight, SWP\_NOZORDER);

SetWindowPos(m\_hStaticLabel, NULL, x5\_text, height / 2 + 50, buttonWidth, buttonHeight, SWP\_NOZORDER);

SetWindowPos(m\_hButtonExtract, NULL, x3, height / 2 + 100, buttonWidth, buttonHeight, SWP\_NOZORDER);

break;

}

case WM\_COMMAND:

{

switch (wParam)

{

case choosePath:

{

OPENFILENAME ofn;

ZeroMemory(&ofn, sizeof(ofn));

ofn.lStructSize = sizeof(ofn);

ofn.hwndOwner = m\_hwnd;

ofn.lpstrFile = m\_wfilePath;

ofn.nMaxFile = MAX\_PATH;

ofn.lpstrFilter = L"Архивы Gzip (\*.gz)\0\*.gz\0Все файлы (\*.\*)\0\*.\*\0";

ofn.nFilterIndex = 1;

ofn.lpstrFileTitle = NULL;

ofn.nMaxFileTitle = 0;

ofn.lpstrInitialDir = NULL;

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

if (GetOpenFileName(&ofn)) {

SetWindowText(m\_hEditPath, m\_wfilePath);

char filePath[MAX\_PATH];

size\_t convertedChars = 0;

errno\_t err = wcstombs\_s(&convertedChars, filePath, MAX\_PATH, m\_wfilePath, MAX\_PATH - 1);

if (err != 0) {

MessageBox(m\_hwnd, L"Ошибка конвертации строки", L"Архивация", MB\_ICONERROR);

}

std::string str(filePath);

if (!str.find(".gz"))

{

MessageBox(m\_hwnd, L"Неверный формат файла", L"Архивация", MB\_ICONERROR);

break;

}

}

break;

}

case extract:

{

GetWindowText(m\_hEditPath, m\_wfilePath, MAX\_PATH);

char filePath[MAX\_PATH];

size\_t convertedChars = 0;

errno\_t err = wcstombs\_s(&convertedChars, filePath, MAX\_PATH, m\_wfilePath, MAX\_PATH - 1);

if (err != 0) {

MessageBox(m\_hwnd, L"Ошибка конвертации строки", L"Извлечение", MB\_ICONERROR);

break;

}

std::string str(filePath);

if (!str.find(".gz") || !fileExists(m\_wfilePath))

{

MessageBox(m\_hwnd, L"Неверный формат файла", L"Извлечение", MB\_ICONERROR);

break;

}

std::string destPath(filePath);

size\_t found = destPath.find\_last\_of(".");

wchar\_t type[100];

GetWindowTextW(m\_hEditType, type, 100);

std::wstring ws(type);

// your new String

std::string str\_type(ws.begin(), ws.end());

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

destPath = (destPath.substr(0, found) + "." + str\_type).c\_str();

}

else {

destPath = destPath.c\_str();

}

// newsize describes the length of the

// wchar\_t string called wcstring in terms of the number

// of wide characters, not the number of bytes.

size\_t newsize = destPath.length() + 1;

// The following creates a buffer large enough to contain

// the exact number of characters in the original string

// in the new format. If you want to add more characters

// to the end of the string, increase the value of newsize

// to increase the size of the buffer.

wchar\_t\* wcstring = new wchar\_t[newsize];

// Convert char\* string to a wchar\_t\* string.

size\_t convertedChars2 = 0;

mbstowcs\_s(&convertedChars2, wcstring, newsize, destPath.c\_str(), \_TRUNCATE);

// Display the result and indicate the type of string that it is.

if (extractFile(m\_hwnd, filePath, wcstring))

{

MessageBox(m\_hwnd, L"Процесс извлечения успешно запущен!", L"Извлечение", MB\_ICONINFORMATION);

}

delete[] wcstring;

break;

}

default:

break;

}

break;

}

case WM\_CLOSE:

{

ShowWindow(m\_hwnd, SW\_HIDE);

break;

}

default:

{

return DefWindowProc(m\_hwnd, uMsg, wParam, lParam);

}

}

}

bool ExtractWindow::fileExists(LPWSTR filePath)

{

WIN32\_FIND\_DATA findFileData;

HANDLE hFile = FindFirstFile(filePath, &findFileData);

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

FindClose(hFile);

return true;

}

return false;

}