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.

Студент: гр.153502 Матвеев Н.С.

Руководитель: ассистент кафедры информатики Гриценко Н.Ю.

Минск 2023

**СОДЕРЖАНИЕ**

1 Формулировка задачи 3

2 Теоретические сведения 4

3 Описание функций программы 5

Список использованных источников 7

Приложение А (обязательное) Исходный код программы 8

**1 ФОРМУЛИРОВКА ЗАДАЧИ**

Целью выполнения лабораторной работы является разработка приложения для сжатия файлов с использованием асинхронных операций ввода и вывода.

В качестве задачи необходимо реализовать возможность сжатия и распаковки файлов.

**2 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

Существует два типа синхронизации ввода-вывода: синхронный ввод-вывод и асинхронный ввод-вывод. Асинхронный ввод-вывод также называется перекрывающимся вводом-выводом.

В синхронном файловом вводе-вывод поток запускает операцию ввода-вывода и сразу же переходит в состояние ожидания, пока запрос ввода-вывода не завершится. Поток, выполняющий асинхронный файловый ввод-вывод , отправляет запрос ввода-вывода в ядро путем вызова соответствующей функции. Если запрос принимается ядром, вызывающий поток продолжает обработку другого задания, пока ядро не сигнализирует потоку о завершении операции ввода-вывода. Затем он прерывает текущее задание и при необходимости обрабатывает данные операции ввода-вывода [1].

Для сжатия файлов использовался *API* сжатия. *API* сжатия предназначен для профессиональных разработчиков приложений *windows* на *C/C*++. *API* сжатия предоставляет алгоритмы сжатия без потерь, используемые Windows, через общедоступный интерфейс и *API* пользовательского режима. *API* сжатия — это рекомендуемый метод для разработчиков *Windows* для управления алгоритмами сжатия. Эта функция является 64-разрядной в 64-разрядной версии *Windows* и 32-разрядной в 32-разрядной версии *Windows* [2].

**3 ОПИСАНИЕ ФУНКЦИЙ ПРОГРАММЫ**

В программе предусмотрены следующие функции:

* возможность сжать файл;
* возможность извлечь файл;
* возможность сохранить результат операции.

Операции считывания и записи в файл выполняются асинхронно.

Графический интерфейс программы представлен на рисунке 1.

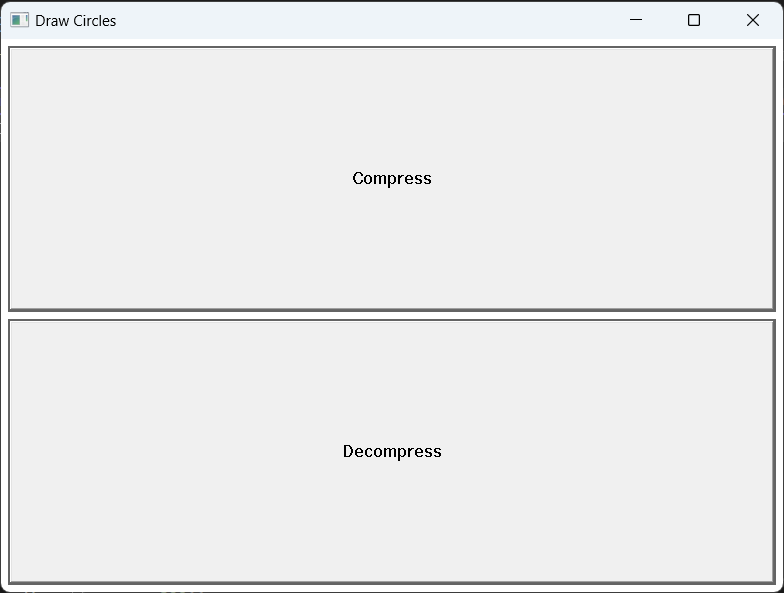


Рисунок 1 – Пример графического интерфейса программы

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

[1] Синхронные и асинхронные операции ввода-вывода [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/fileio/synchronous-and-asynchronous-i-o

[2] API Сжатия [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/cmpapi/-compression-portal

**ПРИЛОЖЕНИЕ А**

**(обязательное)**

**Исходный код программы**

**Файл BaseWindow.h**

*#pragma once*

*template <class DERIVED\_TYPE>*

*class BaseWindow*

*{*

*public:*

*const PCWSTR CLASS\_NAME;*

*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(PCWSTR CLASS\_NAME) : CLASS\_NAME(CLASS\_NAME), m\_hwnd(NULL) { }*

*BOOL Create(*

*PCWSTR lpWindowName,*

*DWORD dwStyle,*

*HWND hWndParent = 0,*

*DWORD dwExStyle = 0,*

*int x = CW\_USEDEFAULT,*

*int y = CW\_USEDEFAULT,*

*int nWidth = CW\_USEDEFAULT,*

*int nHeight = CW\_USEDEFAULT,*

*HMENU hMenu = 0*

*)*

*{*

*WNDCLASS wc = {};*

*wc.lpfnWndProc = DERIVED\_TYPE::WindowProc;*

*wc.hInstance = GetModuleHandle(NULL);*

*wc.lpszClassName = CLASS\_NAME;*

*RegisterClass(&wc);*

*m\_hwnd = CreateWindowEx(*

*dwExStyle, CLASS\_NAME, lpWindowName, dwStyle, x, y,*

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

*);*

*return (m\_hwnd ? TRUE : FALSE);*

*}*

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

*protected:*

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

*HWND m\_hwnd;*

*};*

**Файл DPIScale.h**

*#pragma once*

*#include <d2d1.h>*

*class DPIScale*

*{*

*static float scaleX;*

*static float scaleY;*

*public:*

*static void Initialize()*

*{*

*FLOAT dpi = GetDpiForSystem();*

*scaleX = dpi / 96.0f;*

*scaleY = dpi / 96.0f;*

*}*

*template <typename T>*

*static float PixelsToDipsX(T x)*

*{*

*return static\_cast<float>(x) / scaleX;*

*}*

*template <typename T>*

*static float PixelsToDipsY(T y)*

*{*

*return static\_cast<float>(y) / scaleY;*

*}*

*template <typename T>*

*static T DipXToPixels(float x)*

*{*

*return static\_cast<T>(x) \* scaleX;*

*}*

*template <typename T>*

*static T DipYToPixels(float y)*

*{*

*return static\_cast<T>(y) \* scaleY;*

*}*

*};*

**Файл main.cpp**

*#include <windows.h>*

*#include <Windowsx.h>*

*#include <d2d1.h>*

*#include <list>*

*#include <memory>*

*using namespace std;*

*#pragma comment(lib, "d2d1")*

*#include "BaseWindow.h"*

*#include "EllipseFigure.h"*

*#include "resource.h"*

*#include "GraphicsScene.h"*

*#include "SceneControl.h"*

*#include "MainWindow.h"*

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

*{*

*DPIScale::Initialize();*

*MainWindow win = MainWindow();*

*//Mode mode = Mode::SelectMode;*

*//Figure figure = Figure::Ellipse;*

*//D2D1\_COLOR\_F color = D2D1::ColorF(D2D1::ColorF::Black);*

*//SceneControl win = SceneControl(&mode, &figure, &color);*

*//GraphicsScene win = GraphicsScene(&mode, &figure, &color);*

*if (!win.Create(L"Draw Circles",*

*WS\_OVERLAPPEDWINDOW,*

*NULL,*

*NULL,*

*CW\_USEDEFAULT,*

*CW\_USEDEFAULT,*

*800,*

*600))*

*{*

*return 0;*

*}*

*ShowWindow(win.Window(), nCmdShow);*

*MSG msg;*

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

*{*

*TranslateMessage(&msg);*

*DispatchMessage(&msg);*

*}*

*return 0;*

*}*

**Файл MainWindow.h**

*#pragma once*

*#include <windows.h>*

*#include <Windowsx.h>*

*#include <d2d1.h>*

*#include <ShObjIdl\_core.h>*

*#include <compressapi.h>*

*#include "settings.h"*

*#include "BaseWindow.h"*

*#include "DPIScale.h"*

*#include "resource.h"*

*#include "GraphicsScene.h"*

*#include "SceneControl.h"*

*class MainWindow : public BaseWindow<MainWindow>*

*{*

*private:*

*static const PCWSTR DEFAULT\_CLASS\_NAME;*

*static const float MARGIN\_X;*

*static const float MARGIN\_Y;*

*static const LPWSTR DEFAULT\_EXTENSIONS;*

*static const COMDLG\_FILTERSPEC FILE\_TYPES[];*

*public:*

*MainWindow(PCWSTR CLASS\_NAME = DEFAULT\_CLASS\_NAME);*

*~MainWindow();*

*bool compress();*

*bool decompress();*

*HANDLE openFile(UINT fileTypesSize = 0, const COMDLG\_FILTERSPEC\* fileTypes = NULL, LPWSTR defaultExtension = NULL);*

*HANDLE saveFile(UINT fileTypesSize = 0, const COMDLG\_FILTERSPEC\* fileTypes = NULL, LPWSTR defaultExtension = NULL);*

*virtual LRESULT HandleMessage(UINT uMsg, WPARAM wParam, LPARAM lParam) override;*

*protected:*

*void CreateLayout();*

*void SetLayout();*

*HWND compressButton;*

*HWND decompressButton;*

*};*

**Файл MainWindow.cpp**

*#include "MainWindow.h"*

*#include "helper\_functions.h"*

*#pragma comment(lib,"cabinet.lib")*

*const PCWSTR MainWindow::DEFAULT\_CLASS\_NAME = L"Graphics";*

*const float MainWindow::MARGIN\_X = 6.0F;*

*const float MainWindow::MARGIN\_Y = 6.0F;*

*const LPWSTR MainWindow::DEFAULT\_EXTENSIONS = L"zip";*

*const COMDLG\_FILTERSPEC MainWindow::FILE\_TYPES[] = {*

*{ L"Archive (.zip)", L"\*.zip" },*

*};*

*MainWindow::MainWindow(PCWSTR CLASS\_NAME) :*

*BaseWindow<MainWindow>(CLASS\_NAME)*

*{*

*}*

*MainWindow::~MainWindow()*

*{*

*}*

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

*{*

*switch (uMsg)*

*{*

*case WM\_CREATE:*

*CreateLayout();*

*return 0;*

*case WM\_PAINT:*

*{*

*PAINTSTRUCT ps;*

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

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

*EndPaint(m\_hwnd, &ps);*

*return 0;*

*}*

*case WM\_SIZE:*

*SetLayout();*

*return 0;*

*case WM\_SIZING:*

*{*

*RECT\* dragRc = (RECT\*)lParam;*

*RECT rcWindow;*

*GetWindowRect(m\_hwnd, &rcWindow);*

*int WINDOW\_HEIGHT\_PIX = rcWindow.bottom - rcWindow.top;*

*int WINDOW\_WIDTH\_PIX = rcWindow.right - rcWindow.left;*

*switch (wParam)*

*{*

*case WMSZ\_BOTTOM:*

*dragRc->bottom = dragRc->top + WINDOW\_HEIGHT\_PIX;*

*break;*

*case WMSZ\_BOTTOMLEFT:*

*dragRc->bottom = dragRc->top + WINDOW\_HEIGHT\_PIX;*

*dragRc->left = dragRc->right - WINDOW\_WIDTH\_PIX;*

*break;*

*case WMSZ\_LEFT:*

*dragRc->left = dragRc->right - WINDOW\_WIDTH\_PIX;*

*break;*

*case WMSZ\_TOPLEFT:*

*dragRc->left = dragRc->right - WINDOW\_WIDTH\_PIX;*

*dragRc->top = dragRc->bottom - WINDOW\_HEIGHT\_PIX;*

*break;*

*case WMSZ\_TOP:*

*dragRc->top = dragRc->bottom - WINDOW\_HEIGHT\_PIX;*

*break;*

*case WMSZ\_TOPRIGHT:*

*dragRc->top = dragRc->bottom - WINDOW\_HEIGHT\_PIX;*

*dragRc->right = dragRc->left + WINDOW\_WIDTH\_PIX;*

*break;*

*case WMSZ\_RIGHT:*

*dragRc->right = dragRc->left + WINDOW\_WIDTH\_PIX;*

*break;*

*case WMSZ\_BOTTOMRIGHT:*

*dragRc->right = dragRc->left + WINDOW\_WIDTH\_PIX;*

*dragRc->bottom = dragRc->top + WINDOW\_HEIGHT\_PIX;*

*break;*

*}*

*return TRUE;*

*}*

*case WM\_COMMAND:*

*if (HIWORD(wParam) == BN\_CLICKED)*

*{*

*if ((HWND)lParam == compressButton)*

*{*

*compress();*

*}*

*else if ((HWND)lParam == decompressButton)*

*{*

*decompress();*

*}*

*return 0;*

*}*

*break;*

*}*

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

*}*

*void MainWindow::CreateLayout()*

*{*

*compressButton = CreateWindow(L"BUTTON",*

*L"Compress",*

*WS\_TABSTOP | WS\_VISIBLE | WS\_CHILD | BS\_DEFPUSHBUTTON | WS\_BORDER,*

*0,*

*0,*

*0,*

*0,*

*m\_hwnd,*

*NULL,*

*GetModuleHandle(NULL),*

*NULL);*

*decompressButton = CreateWindow(L"BUTTON",*

*L"Decompress",*

*WS\_TABSTOP | WS\_VISIBLE | WS\_CHILD | BS\_DEFPUSHBUTTON | WS\_BORDER,*

*0,*

*0,*

*0,*

*0,*

*m\_hwnd,*

*NULL,*

*GetModuleHandle(NULL),*

*NULL);*

*}*

*void MainWindow::SetLayout()*

*{*

*RECT rcClient;*

*GetClientRect(m\_hwnd, &rcClient);*

*int MARGIN\_XPix = DPIScale::DipXToPixels<int>(MARGIN\_X);*

*int MARGIN\_YPix = DPIScale::DipYToPixels<int>(MARGIN\_Y);*

*int WINDOW\_HEIGHT\_PIX = rcClient.bottom - rcClient.top;*

*int WINDOW\_WIDTH\_PIX = rcClient.right - rcClient.left;*

*MoveWindow(compressButton,*

*MARGIN\_XPix,*

*MARGIN\_YPix,*

*WINDOW\_WIDTH\_PIX - MARGIN\_XPix \* 2,*

*(WINDOW\_HEIGHT\_PIX - MARGIN\_YPix \* 3) / 2,*

*FALSE);*

*MoveWindow(decompressButton,*

*MARGIN\_XPix,*

*MARGIN\_YPix \* 2 + (WINDOW\_HEIGHT\_PIX - MARGIN\_YPix \* 3) / 2,*

*WINDOW\_WIDTH\_PIX - MARGIN\_XPix \* 2,*

*(WINDOW\_HEIGHT\_PIX - MARGIN\_YPix \* 3) / 2,*

*FALSE);*

*InvalidateRect(m\_hwnd, NULL, FALSE);*

*}*

*bool MainWindow::compress()*

*{*

*double result = 0;*

*HANDLE decompressedFile = openFile();*

*if (decompressedFile != INVALID\_HANDLE\_VALUE)*

*{*

*LARGE\_INTEGER fileSize;*

*if (GetFileSizeEx(decompressedFile, &fileSize) && fileSize.QuadPart < 0xffffffff)*

*{*

*DWORD realFileSize = fileSize.LowPart;*

*HANDLE alertEvent = CreateEventA(NULL, FALSE, FALSE, NULL);*

*if (alertEvent != INVALID\_HANDLE\_VALUE)*

*{*

*OVERLAPPED overlapped = OVERLAPPED();*

*overlapped.hEvent = alertEvent;*

*LPVOID buffer = (LPVOID)malloc(realFileSize);*

*if (buffer)*

*{*

*BOOL readResult = ReadFile(decompressedFile, buffer, realFileSize, NULL, &overlapped);*

*bool isOverlapped = GetLastError() == ERROR\_IO\_PENDING;*

*if (readResult || isOverlapped)*

*{*

*COMPRESSOR\_HANDLE compressor;*

*if (CreateCompressor(COMPRESS\_ALGORITHM\_MSZIP, NULL, &compressor))*

*{*

*bool readSuccessfull = true;*

*if (isOverlapped)*

*{*

*MsgWaitForMultipleObjects(1, &alertEvent, TRUE, INFINITE, QS\_ALLINPUT);*

*DWORD numberOfBytesTransfered;*

*readSuccessfull = GetOverlappedResult(decompressedFile, &overlapped, &numberOfBytesTransfered, FALSE);*

*}*

*if (readSuccessfull)*

*{*

*SIZE\_T compressBufferSize;*

*if (!Compress(compressor, buffer, realFileSize, NULL, 0, &compressBufferSize))*

*{*

*DWORD compressError = GetLastError();*

*if (compressError == ERROR\_INSUFFICIENT\_BUFFER)*

*{*

*DWORD ErrorCode = GetLastError();*

*PBYTE compressBuffer = (PBYTE)malloc(compressBufferSize);*

*if (compressBuffer)*

*{*

*SIZE\_T compressDataSize;*

*ULONGLONG StartTime = GetTickCount64();*

*if (Compress(compressor, buffer, realFileSize, compressBuffer, compressBufferSize, &compressDataSize))*

*{*

*double duration = (GetTickCount64() - StartTime) / (double)1000;*

*HANDLE compressedFile = saveFile(ARRAYSIZE(FILE\_TYPES), FILE\_TYPES, DEFAULT\_EXTENSIONS);*

*if (compressedFile != INVALID\_HANDLE\_VALUE)*

*{*

*HANDLE alertEvent = CreateEventA(NULL, FALSE, FALSE, NULL);*

*if (alertEvent != INVALID\_HANDLE\_VALUE)*

*{*

*OVERLAPPED overlapped = OVERLAPPED();*

*overlapped.hEvent = alertEvent;*

*BOOL writeResult = WriteFile(compressedFile, compressBuffer, compressDataSize, NULL, &overlapped);*

*bool isOverlapped = GetLastError() == ERROR\_IO\_PENDING;*

*if (writeResult || isOverlapped)*

*{*

*bool writeSuccessfull = true;*

*if (isOverlapped)*

*{*

*MsgWaitForMultipleObjects(1, &alertEvent, TRUE, INFINITE, QS\_ALLINPUT);*

*DWORD numberOfBytesTransfered;*

*writeSuccessfull = GetOverlappedResult(decompressedFile, &overlapped, &numberOfBytesTransfered, FALSE);*

*}*

*if (writeSuccessfull)*

*{*

*result = duration;*

*}*

*}*

*}*

*CloseHandle(compressedFile);*

*}*

*}*

*free(compressBuffer);*

*}*

*}*

*}*

*}*

*CloseCompressor(compressor);*

*}*

*}*

*free(buffer);*

*}*

*CloseHandle(alertEvent);*

*}*

*}*

*CloseHandle(decompressedFile);*

*}*

*return result;*

*}*

*bool MainWindow::decompress()*

*{*

*double result = 0;*

*HANDLE compressedFile = openFile(ARRAYSIZE(FILE\_TYPES), FILE\_TYPES, DEFAULT\_EXTENSIONS);*

*if (compressedFile != INVALID\_HANDLE\_VALUE)*

*{*

*LARGE\_INTEGER fileSize;*

*if (GetFileSizeEx(compressedFile, &fileSize) && fileSize.QuadPart < 0xffffffff)*

*{*

*DWORD realFileSize = fileSize.LowPart;*

*HANDLE alertEvent = CreateEventA(NULL, FALSE, FALSE, NULL);*

*if (alertEvent != INVALID\_HANDLE\_VALUE)*

*{*

*OVERLAPPED overlapped = OVERLAPPED();*

*overlapped.hEvent = alertEvent;*

*LPVOID buffer = (LPVOID)malloc(realFileSize);*

*if (buffer)*

*{*

*BOOL readResult = ReadFile(compressedFile, buffer, realFileSize, NULL, &overlapped);*

*bool isOverlapped = GetLastError() == ERROR\_IO\_PENDING;*

*if (readResult || isOverlapped)*

*{*

*DECOMPRESSOR\_HANDLE decompressor;*

*if (CreateDecompressor(COMPRESS\_ALGORITHM\_MSZIP, NULL, &decompressor))*

*{*

*bool readSuccessfull = true;*

*if (isOverlapped)*

*{*

*MsgWaitForMultipleObjects(1, &alertEvent, TRUE, INFINITE, QS\_ALLINPUT);*

*DWORD numberOfBytesTransfered;*

*readSuccessfull = GetOverlappedResult(compressedFile, &overlapped, &numberOfBytesTransfered, FALSE);*

*}*

*if (readSuccessfull)*

*{*

*SIZE\_T decompressBufferSize;*

*if (!Decompress(decompressor, buffer, realFileSize, NULL, 0, &decompressBufferSize))*

*{*

*DWORD decompressError = GetLastError();*

*if (decompressError == ERROR\_INSUFFICIENT\_BUFFER)*

*{*

*DWORD ErrorCode = GetLastError();*

*PBYTE decompressBuffer = (PBYTE)malloc(decompressBufferSize);*

*if (decompressBuffer)*

*{*

*SIZE\_T decompressDataSize;*

*ULONGLONG StartTime = GetTickCount64();*

*if (Decompress(decompressor, buffer, realFileSize, decompressBuffer, decompressBufferSize, &decompressDataSize))*

*{*

*double duration = (GetTickCount64() - StartTime) / (double)1000;*

*HANDLE decompressedFile = saveFile();*

*if (decompressedFile != INVALID\_HANDLE\_VALUE)*

*{*

*HANDLE alertEvent = CreateEventA(NULL, FALSE, FALSE, NULL);*

*if (alertEvent != INVALID\_HANDLE\_VALUE)*

*{*

*OVERLAPPED overlapped = OVERLAPPED();*

*overlapped.hEvent = alertEvent;*

*BOOL writeResult = WriteFile(decompressedFile, decompressBuffer, decompressDataSize, NULL, &overlapped);*

*bool isOverlapped = GetLastError() == ERROR\_IO\_PENDING;*

*if (writeResult || isOverlapped)*

*{*

*bool writeSuccessfull = true;*

*if (isOverlapped)*

*{*

*MsgWaitForMultipleObjects(1, &alertEvent, TRUE, INFINITE, QS\_ALLINPUT);*

*DWORD numberOfBytesTransfered;*

*writeSuccessfull = GetOverlappedResult(compressedFile, &overlapped, &numberOfBytesTransfered, FALSE);*

*}*

*if (writeSuccessfull)*

*{*

*result = duration;*

*}*

*}*

*}*

*CloseHandle(decompressedFile);*

*}*

*}*

*free(decompressBuffer);*

*}*

*}*

*}*

*}*

*CloseCompressor(decompressor);*

*}*

*}*

*free(buffer);*

*}*

*CloseHandle(alertEvent);*

*}*

*}*

*CloseHandle(compressedFile);*

*}*

*return result;*

*}*

*HANDLE MainWindow::openFile(UINT fileTypesSize, const COMDLG\_FILTERSPEC\* fileTypes, LPWSTR defaultExtension)*

*{*

*HANDLE result = INVALID\_HANDLE\_VALUE;*

*IFileDialog\* fileDialog = NULL;*

*HRESULT hr = CoCreateInstance(CLSID\_FileOpenDialog,*

*NULL,*

*CLSCTX\_INPROC\_SERVER,*

*IID\_PPV\_ARGS(&fileDialog));*

*if (SUCCEEDED(hr))*

*{*

*if (fileTypes)*

*{*

*hr = fileDialog->SetFileTypes(fileTypesSize, fileTypes);*

*}*

*if (SUCCEEDED(hr))*

*{*

*if (defaultExtension)*

*{*

*hr = fileDialog->SetDefaultExtension(defaultExtension);*

*}*

*if (SUCCEEDED(hr))*

*{*

*hr = fileDialog->Show(NULL);*

*if (SUCCEEDED(hr))*

*{*

*IShellItem\* shellItem;*

*hr = fileDialog->GetResult(&shellItem);*

*if (SUCCEEDED(hr))*

*{*

*PWSTR filePath = NULL;*

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

*if (SUCCEEDED(hr))*

*{*

*result = CreateFile(filePath, GENERIC\_READ, NULL, NULL, OPEN\_EXISTING, FILE\_ATTRIBUTE\_NORMAL | FILE\_FLAG\_OVERLAPPED, NULL);*

*}*

*shellItem->Release();*

*}*

*}*

*fileDialog->Release();*

*}*

*}*

*}*

*return result;*

*}*

*HANDLE MainWindow::saveFile(UINT fileTypesSize, const COMDLG\_FILTERSPEC\* fileTypes, LPWSTR defaultExtension)*

*{*

*HANDLE result = INVALID\_HANDLE\_VALUE;*

*IFileDialog\* fileDialog = NULL;*

*HRESULT hr = CoCreateInstance(CLSID\_FileSaveDialog,*

*NULL,*

*CLSCTX\_INPROC\_SERVER,*

*IID\_PPV\_ARGS(&fileDialog));*

*if (SUCCEEDED(hr))*

*{*

*if (fileTypes)*

*{*

*hr = fileDialog->SetFileTypes(fileTypesSize, fileTypes);*

*}*

*if (SUCCEEDED(hr))*

*{*

*if (defaultExtension)*

*{*

*hr = fileDialog->SetDefaultExtension(defaultExtension);*

*}*

*if (SUCCEEDED(hr))*

*{*

*hr = fileDialog->Show(NULL);*

*if (SUCCEEDED(hr))*

*{*

*IShellItem\* shellItem;*

*hr = fileDialog->GetResult(&shellItem);*

*if (SUCCEEDED(hr))*

*{*

*PWSTR filePath = NULL;*

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

*if (SUCCEEDED(hr))*

*{*

*result = CreateFile(filePath, GENERIC\_WRITE, NULL, NULL, CREATE\_ALWAYS, FILE\_ATTRIBUTE\_NORMAL | FILE\_FLAG\_OVERLAPPED, NULL);*

*}*

*shellItem->Release();*

*}*

*}*

*}*

*}*

*fileDialog->Release();*

*}*

*return result;*

*}*