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 Сидорова У.Ю.

Проверил: ассистент кафедры информатики Гриценко Н.Ю.

Минск 2023

**СОДЕРЖАНИЕ**

[1 Цель работы 3](#_Toc145669228)

[2 Теоретические сведения 3](#_Toc145669228)

[3 Результат работы программы 4](#_Toc145669229)

[Список использованных источников 6](#_Toc145669230)

[Приложение A 7](#_Toc145669231)

1 ЦЕЛЬ РАБОТЫ

Целью выполнения лабораторной работы является создание приложения на *Win32 API*, которое использует функции *API* подсистемы памяти *Win32,* управляет памятью и выводом, организует и контролирует асинхронные операции вывода.

В качестве задачи необходимо реализовать приложение для визуализации структуры файловой системы и отслеживания изменений в реальном времени.

**2** ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

В лабораторной работе используются общие элементы управления, реализованные в общей библиотеке элементов управления, *Comctl32.dll*, которая является библиотекой *DLL*, входящей в состав операционной системы *Windows*. Как и другие окна элементов управления, общим элементом управления является дочернее окно, которое приложение использует вместе с другим окном для взаимодействия с пользователем [1].

Для выбора папки используется функция *SHBrowseForFolder,* которая отображает диалоговое окно, позволяющее пользователю выбрать папку оболочки [2 ].

Вывод результата происходит с помощью элемента *List View*. Элемент управления представлением списка — это окно, в котором отображается коллекция элементов. Элементы управления представлением списка предоставляют несколько способов упорядочивания и отображения элементов и являются гораздо более гибкими, чем простые списки [3]. Отслеживание действий в выбранной папке осуществляется в отдельном потоке, который создается с помощью функции *CreateThread*. Она создает поток для выполнения в виртуальном адресном пространстве вызывающего процесса и, если функция выполняется успешно, возвращаемое значение является дескриптором для нового потока. Если функция завершается сбоем, возвращается значение *NULL* [4].

При остановке отслеживания поток переводится в состояние ожидания функцией *WaitForSingleObject.* Она проверяет текущее состояние указанного объекта. Если состояние объекта не сигнализировано, вызывающий поток переходит в состояние ожидания до тех пор, пока объект не получит сигнал или не истечет интервал времени ожидания [5]. А так же очищается *Notify Event List* функцией *InterlockedFlushSList*, которая удаляет все элементы из односвязного списка и синхронизирует доступ к списку в многопроцессорной системе [6].

3 РЕЗУЛЬТАТ РАБОТЫ ПРОГРАММЫ

При запуске программы открывается оконное приложение (рисунок 1).

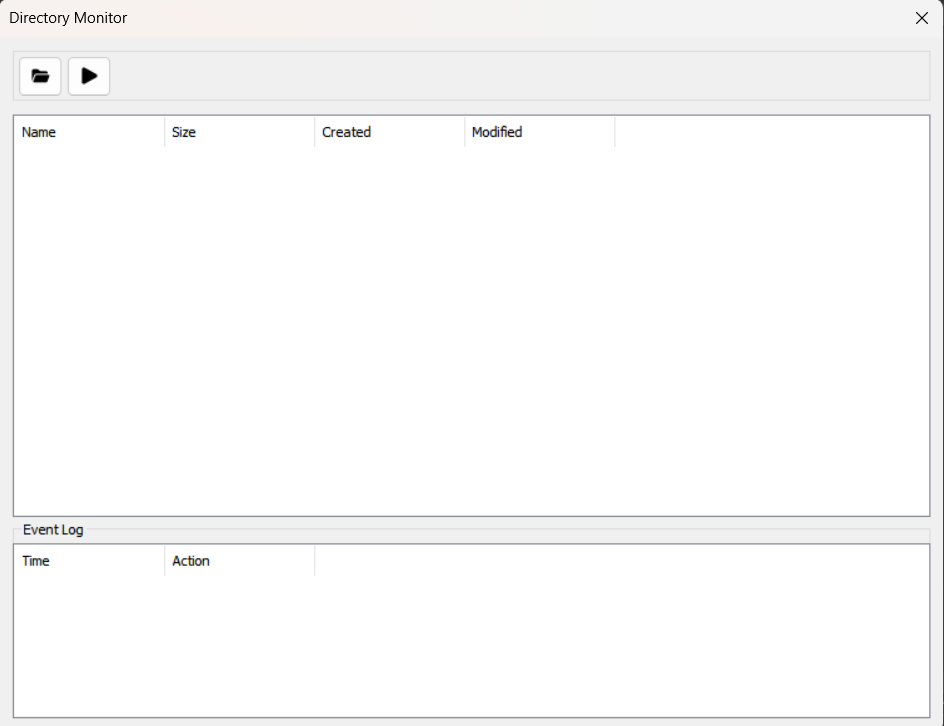


Рисунок 1 – Графический интерфейс программы

Функциональные возможности приложения:

– выбрать каталог;

При нажатии на верхнюю левую кнопку открывается диалоговое окно для обзора папок (рисунок 2), которое позволяет открыть существующую директорию.

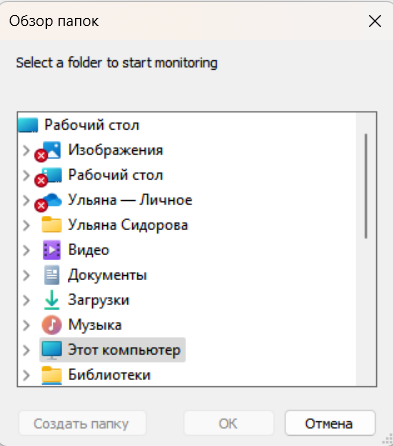


Рисунок 2 – Открытие обзора папок

– просмотр файлов и папок, находящихся в выбранном каталоге;

После выбора текущей папки появляется список файлов и папок, входящих в директорию (рисунок 3).

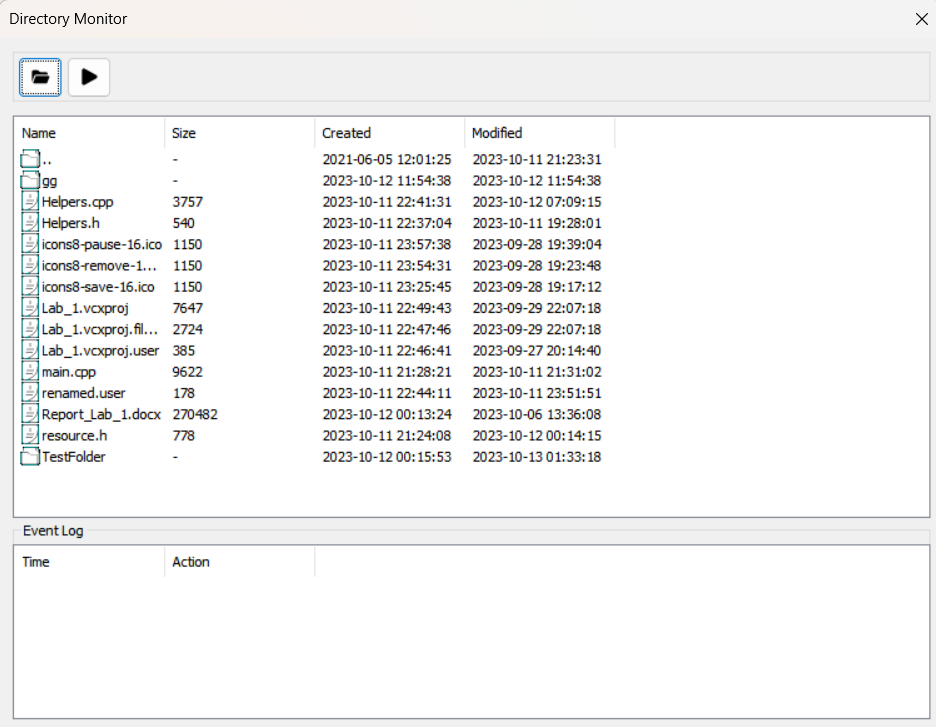


Рисунок 3 – Отображение существующих файлов и папок

– навигация по каталогу;

При помощи двойного нажатия на папку, есть возможность переходить в другие директории. Родительский каталог обозначается двумя точками.

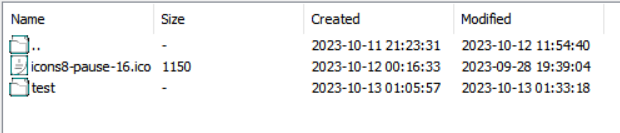


Рисунок 4 – Переход в другой каталог

– отслеживание каталога;

При нажатии на кнопку *play*, начинается отслеживание выбранного каталога, а кнопка *play* меняется на *pause*. В данном режиме изменения фиксируются в реальном времени, а также журналируются. При повторном нажатии отслеживание завершится, а кнопка вернется в первоначальное состояние (рисунок 5).

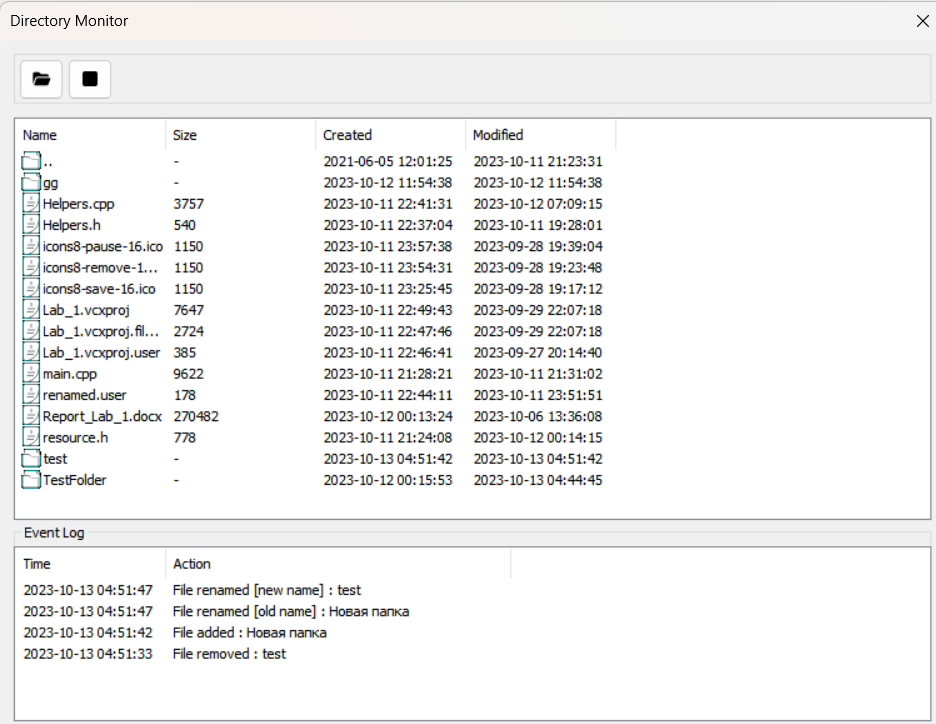


Рисунок 5 – Журналирование изменений

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Общие сведения об элементах управления [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://learn.microsoft.com/ru-ru/windows/win32/controls/common-controls-intro> – Дата доступа 10.10.2023.

[2] SHBrowseForFolderA function (shlobj\_core.h) [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://learn.microsoft.com/en-us/windows/win32/api/shlobj_core/nf-shlobj_core-shbrowseforfoldera> – Дата доступа 10.10.2023.

[3] About List-View Controls [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://learn.microsoft.com/en-us/windows/win32/controls/list-view-controls-overview> – Дата доступа 10.10.2023.

[4] Функция CreateThread (processthreadsapi.h) [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://learn.microsoft.com/ru-ru/windows/win32/api/processthreadsapi/nf-processthreadsapi-createthread> – Дата доступа 10.10.2023.

[5] WaitForSingleObject function (synchapi.h) [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://learn.microsoft.com/en-us/windows/win32/api/synchapi/nf-synchapi-waitforsingleobject> – Дата доступа 10.10.2023.

[6] InterlockedFlushSList function (interlockedapi.h) [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://learn.microsoft.com/en-us/windows/win32/api/interlockedapi/nf-interlockedapi-interlockedflushslist> – Дата доступа 10.10.2023.

ПРИЛОЖЕНИЕ А

**(обязательное)**

**Исходный код программы**

**Файл main.cpp**

#include "resource.h"

#include <windows.h>

#include <CommCtrl.h>

#include <strsafe.h>

#include <string>

#include <list>

#include "Helpers.h"

#pragma comment( lib, "comctl32.lib" )

enum

{

WM\_FILE\_NOTIFY\_EVENT = WM\_APP + 1

};

struct FILE\_NOTIFY\_LIST\_ITEM

{

SLIST\_ENTRY ItemEntry;

DWORD Action;

WCHAR FileName[MAX\_PATH];

};

static HINSTANCE hAppInstance = NULL;

static HWND hMainWindow = NULL;

static std::wstring WatchedDirPath;

static HANDLE hWatchThread = NULL;

static HANDLE hStopWatchingEvent = NULL;

static PSLIST\_HEADER pFileNotifyEventListHead;

INT\_PTR CALLBACK MainDialogProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam);

bool ReadDirectory(std::wstring const& dirPath, HWND hDirListView);

DWORD WINAPI WatchDirectoryProc(LPVOID);

void StartWatching(LPCWSTR dirPath);

void StopWatching();

void UpdatePlayerControls();

std::wstring FileNotifyActionToString(DWORD action);

int WINAPI wWinMain(\_In\_ HINSTANCE hInstance, \_In\_opt\_ HINSTANCE hPrevInstance, \_In\_ PWSTR pCmdLine, \_In\_ int nCmdShow)

{

hAppInstance = hInstance;

INITCOMMONCONTROLSEX iCC = { sizeof(INITCOMMONCONTROLSEX), ICC\_LISTVIEW\_CLASSES | ICC\_PROGRESS\_CLASS };

::InitCommonControlsEx(&iCC);

pFileNotifyEventListHead = (PSLIST\_HEADER)\_aligned\_malloc(sizeof(SLIST\_HEADER),

MEMORY\_ALLOCATION\_ALIGNMENT);

if (NULL == pFileNotifyEventListHead)

{

printf("Memory allocation failed.\n");

return -1;

}

::InitializeSListHead(pFileNotifyEventListHead);

hMainWindow = ::CreateDialogParam(hInstance, MAKEINTRESOURCE(IDD\_MAIN\_DIALOG), NULL, MainDialogProc, 0);

if (hMainWindow == NULL)

{

return -1;

}

::ShowWindow(hMainWindow, nCmdShow);

::UpdateWindow(hMainWindow);

MSG msg = { };

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

{

if (!IsDialogMessage(hMainWindow, &msg))

{

::TranslateMessage(&msg);

::DispatchMessage(&msg);

}

}

return 0;

}

INT\_PTR CALLBACK MainDialogProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam)

{

switch (uMsg)

{

case WM\_DESTROY:

::PostQuitMessage(0);

return TRUE;

case WM\_CLOSE:

::DestroyWindow(hwnd);

return TRUE;

case WM\_FILE\_NOTIFY\_EVENT:

{

std::list<FILE\_NOTIFY\_LIST\_ITEM\*> itemList;

PSLIST\_ENTRY nextEntry = ::InterlockedPopEntrySList(pFileNotifyEventListHead);

while (NULL != nextEntry)

{

itemList.push\_front((FILE\_NOTIFY\_LIST\_ITEM\*)nextEntry);

nextEntry = ::InterlockedPopEntrySList(pFileNotifyEventListHead);

}

for (auto const& nextItem : itemList)

{

FILETIME actionTime;

::GetSystemTimeAsFileTime(&actionTime);

std::wstring actionTimeString = Helpers::FileTimeToString(actionTime);

LVITEMW lvi = { };

lvi.mask = LVIF\_TEXT;

lvi.iItem = 0;

lvi.pszText = &actionTimeString[0];

HWND hLogListView = ::GetDlgItem(hwnd, IDC\_LOG\_LIST);

ListView\_InsertItem(hLogListView, &lvi);

std::wstring logText = FileNotifyActionToString(nextItem->Action) + L" : " + std::wstring(nextItem->FileName);

ListView\_SetItemText(hLogListView, lvi.iItem, 1, &logText[0]);

HWND hDirListView = ::GetDlgItem(hwnd, IDC\_DIR\_LIST);

ListView\_DeleteAllItems(hDirListView);

ReadDirectory(WatchedDirPath + L"\\\*", hDirListView);

\_aligned\_free(nextItem);

}

}

break;

case WM\_COMMAND:

switch (LOWORD(wParam))

{

case IDC\_OPEN\_DIR:

{

std::wstring dirPath = Helpers::BrowseFolder(hwnd, L"Select a folder to start monitoring", L"");

if (!dirPath.empty())

{

if (NULL != hWatchThread)

{

StopWatching();

}

HWND hDirListView = ::GetDlgItem(hwnd, IDC\_DIR\_LIST);

ListView\_DeleteAllItems(hDirListView);

HWND hLogListView = ::GetDlgItem(hwnd, IDC\_LOG\_LIST);

ListView\_DeleteAllItems(hLogListView);

ReadDirectory(dirPath + L"\\\*", hDirListView);

StartWatching(dirPath.c\_str());

}

return TRUE;

}

break;

case IDC\_START\_STOP\_MONITORING:

return TRUE;

break;

}

break;

case WM\_INITDIALOG:

{

Helpers::SetButtonIcon(hAppInstance, hwnd, IDC\_OPEN\_DIR, IDI\_OPEN\_ICON);

Helpers::SetButtonIcon(hAppInstance, hwnd, IDC\_START\_STOP\_MONITORING, IDI\_PLAY\_ICON);

LVCOLUMNW lvc;

lvc.mask = LVCF\_FMT | LVCF\_WIDTH | LVCF\_TEXT | LVCF\_SUBITEM;

lvc.cx = 120;

lvc.fmt = LVCFMT\_LEFT;

static WCHAR szDirListColumnString[4][64] = { L"Name", L"Size", L"Created", L"Modified" };

HWND hDirListView = ::GetDlgItem(hwnd, IDC\_DIR\_LIST);

ListView\_SetExtendedListViewStyle(hDirListView, LVS\_EX\_AUTOSIZECOLUMNS | LVS\_EX\_FULLROWSELECT);

for (auto column = 0; column < 4; ++column)

{

lvc.pszText = szDirListColumnString[column];

if (ListView\_InsertColumn(hDirListView, column, &lvc) == -1)

{

return FALSE;

}

}

static TCHAR szLogListColumnString[4][64] = { L"Time", L"Action" };

HWND hLogListView = ::GetDlgItem(hwnd, IDC\_LOG\_LIST);

ListView\_SetExtendedListViewStyle(hLogListView, LVS\_EX\_AUTOSIZECOLUMNS | LVS\_EX\_FULLROWSELECT);

for (auto column = 0; column < 2; ++column)

{

lvc.pszText = szLogListColumnString[column];

if (ListView\_InsertColumn(hLogListView, column, &lvc) == -1)

{

return FALSE;

}

}

}

break;

}

return FALSE;

}

bool ReadDirectory(std::wstring const& dirPath, HWND hDirListView)

{

WIN32\_FIND\_DATAW ffd;

HANDLE hFind = INVALID\_HANDLE\_VALUE;

hFind = FindFirstFileW(dirPath.c\_str(), &ffd);

if (INVALID\_HANDLE\_VALUE == hFind)

{

return false;

}

LVITEMW lvi = { };

lvi.mask = LVIF\_TEXT;

do

{

if (::CompareStringW(LOCALE\_SYSTEM\_DEFAULT, NORM\_IGNORECASE, ffd.cFileName, -1, L".", -1) == CSTR\_EQUAL ||

::CompareStringW(LOCALE\_SYSTEM\_DEFAULT, NORM\_IGNORECASE, ffd.cFileName, -1, L"..", -1) == CSTR\_EQUAL)

{

continue;

}

lvi.pszText = ffd.cFileName;

ListView\_InsertItem(hDirListView, &lvi);

if (ffd.dwFileAttributes & FILE\_ATTRIBUTE\_DIRECTORY)

{

static WCHAR emptyFileSize[2] = { L"-" };

ListView\_SetItemText(hDirListView, lvi.iItem, 1, emptyFileSize);

}

else

{

LARGE\_INTEGER fileSize = { 0, 0 };

fileSize.LowPart = ffd.nFileSizeLow;

fileSize.HighPart = ffd.nFileSizeHigh;

WCHAR fileSizeString[256];

wsprintfW(fileSizeString, L"%ld", fileSize.QuadPart);

ListView\_SetItemText(hDirListView, lvi.iItem, 1, fileSizeString);

}

std::wstring fileCreatedString = Helpers::FileTimeToString(ffd.ftCreationTime);

ListView\_SetItemText(hDirListView, lvi.iItem, 2, &fileCreatedString[0]);

std::wstring fileModifiedString = Helpers::FileTimeToString(ffd.ftLastWriteTime);

ListView\_SetItemText(hDirListView, lvi.iItem, 3, &fileModifiedString[0]);

++lvi.iItem;

}

while (FindNextFileW(hFind, &ffd) != 0);

DWORD dwError = GetLastError();

if (dwError != ERROR\_NO\_MORE\_FILES)

{

return false;

}

FindClose(hFind);

return true;

}

void StartWatching(LPCWSTR dirPath)

{

WatchedDirPath = dirPath;

hStopWatchingEvent = ::CreateEvent(NULL, FALSE, FALSE, NULL);

hWatchThread = ::CreateThread(NULL, 0, WatchDirectoryProc, 0, 0, NULL);

}

void StopWatching()

{

if (NULL != hWatchThread)

{

::SetEvent(hStopWatchingEvent);

::WaitForSingleObject(hWatchThread, INFINITE);

}

::InterlockedFlushSList(pFileNotifyEventListHead);

hStopWatchingEvent = NULL;

hWatchThread = NULL;

}

DWORD WINAPI WatchDirectoryProc(LPVOID)

{

static const UINT CHANGES\_BUFFER\_SIZE = 1024;

LPCWSTR dirPath = WatchedDirPath.c\_str();

HANDLE hDirHandle = CreateFile(dirPath, FILE\_LIST\_DIRECTORY, FILE\_SHARE\_READ | FILE\_SHARE\_WRITE | FILE\_SHARE\_DELETE, NULL, OPEN\_EXISTING,

FILE\_FLAG\_BACKUP\_SEMANTICS | FILE\_FLAG\_OVERLAPPED, NULL);

if (INVALID\_HANDLE\_VALUE == hDirHandle)

{

return ::GetLastError();

}

OVERLAPPED overlapped;

overlapped.hEvent = CreateEvent(NULL, FALSE, 0, NULL);

if (INVALID\_HANDLE\_VALUE == overlapped.hEvent)

{

return ::GetLastError();

}

while (true)

{

DWORD changesBuffer[CHANGES\_BUFFER\_SIZE];

if (!::ReadDirectoryChangesW(hDirHandle, changesBuffer, sizeof(changesBuffer), TRUE,

FILE\_NOTIFY\_CHANGE\_FILE\_NAME | FILE\_NOTIFY\_CHANGE\_DIR\_NAME | FILE\_NOTIFY\_CHANGE\_CREATION | FILE\_NOTIFY\_CHANGE\_LAST\_WRITE, NULL, &overlapped, NULL))

{

return ::GetLastError();

}

HANDLE events[2] = { overlapped.hEvent, hStopWatchingEvent };

DWORD result = ::WaitForMultipleObjects(2, events, FALSE, INFINITE);

if (result == WAIT\_OBJECT\_0)

{

DWORD bytesTransferred;

::GetOverlappedResult(hDirHandle, &overlapped, &bytesTransferred, FALSE);

FILE\_NOTIFY\_INFORMATION\* event = (FILE\_NOTIFY\_INFORMATION\*)changesBuffer;

for (;;) {

FILE\_NOTIFY\_LIST\_ITEM\* pNotifyItem = (FILE\_NOTIFY\_LIST\_ITEM\*)\_aligned\_malloc(sizeof(FILE\_NOTIFY\_LIST\_ITEM),

MEMORY\_ALLOCATION\_ALIGNMENT);

::ZeroMemory(pNotifyItem, sizeof(FILE\_NOTIFY\_LIST\_ITEM));

if (NULL == pNotifyItem)

{

return -1;

}

pNotifyItem->Action = event->Action;

::StringCchCopyN(pNotifyItem->FileName, MAX\_PATH, event->FileName, event->FileNameLength / sizeof(WCHAR));

::InterlockedPushEntrySList(pFileNotifyEventListHead, &pNotifyItem->ItemEntry);

if (event->NextEntryOffset)

{

\*((uint8\_t\*\*)&event) += event->NextEntryOffset;

}

else

{

break;

}

}

::PostMessage(hMainWindow, WM\_FILE\_NOTIFY\_EVENT, 0, 0);

}

else if (result == WAIT\_OBJECT\_0 + 1)

{

break;

}

}

::CloseHandle(hStopWatchingEvent);

hStopWatchingEvent = NULL;

hWatchThread = NULL;

return 0;

}

std::wstring FileNotifyActionToString(DWORD action)

{

switch (action)

{

case FILE\_ACTION\_ADDED:

{

return L"File added";

}

break;

case FILE\_ACTION\_REMOVED:

{

return L"File removed";

}

break;

case FILE\_ACTION\_MODIFIED:

{

return L"File modified";

}

break;

case FILE\_ACTION\_RENAMED\_OLD\_NAME:

{

return L"File renamed [old name]";

}

break;

case FILE\_ACTION\_RENAMED\_NEW\_NAME:

{

return L"File renamed [new name]";

}

break;

default:

{

return L"Unknown action";

}

break;

}

}

**Файл Helpers.h**

#pragma once

#include <windows.h>

#include <string>

namespace Helpers

{

bool SetButtonIcon(HINSTANCE hInstance, HWND hDlg, int buttonID, int iconID);

std::wstring BrowseForFile(HINSTANCE hInstance, HWND hParentWindow, LPCWSTR title,

LPCWSTR filter, LPCWSTR defExtention, bool existingOnly);

std::wstring BrowseFolder(HWND hParentWindow, LPCWSTR title, LPCWSTR startFromPath);

void SetControlEnabled(HWND hDlg, int controlID, bool enabled);

std::wstring FileTimeToString(FILETIME ftime);

}

**Файл Helpers.cpp**

#include "Helpers.h"

#include <codecvt>

#include <shlobj.h>

#include <sstream>

#include <iomanip>

namespace Helpers

{

bool SetButtonIcon(HINSTANCE hInstance, HWND hDlg, int buttonID, int iconID)

{

HICON hIcon = static\_cast<HICON>(::LoadImage(hInstance, MAKEINTRESOURCE(iconID), IMAGE\_ICON,

16, 16, LR\_DEFAULTCOLOR));

if (hIcon == NULL)

{

return FALSE;

}

::SendDlgItemMessage(hDlg, buttonID, BM\_SETIMAGE, (WPARAM)IMAGE\_ICON, (LPARAM)hIcon);

return TRUE;

}

std::wstring BrowseForFile(HINSTANCE hInstance, HWND hParentWindow, LPCTSTR title,

LPCTSTR filter, LPCTSTR defExtention, bool existingOnly)

{

static const int MaxFilePath = 1024;

OPENFILENAME of = {};

of.lStructSize = sizeof(OPENFILENAME);

of.hwndOwner = hParentWindow;

of.hInstance = hInstance;

of.lpstrFilter = filter;

of.lpstrDefExt = defExtention;

of.lpstrTitle = title;

of.Flags = OFN\_EXPLORER;

if (existingOnly)

{

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

}

TCHAR selectedFilePath[MaxFilePath] = { 0 };

of.lpstrFile = selectedFilePath;

of.nMaxFile = MaxFilePath;

BOOL ok = ::GetOpenFileName(&of);

if (ok)

{

return selectedFilePath;

}

return std::wstring();

}

static int CALLBACK BrowseCallbackProc(HWND hwnd, UINT uMsg, LPARAM lParam, LPARAM lpData)

{

if (uMsg == BFFM\_INITIALIZED)

{

SendMessage(hwnd, BFFM\_SETSELECTION, TRUE, lpData);

}

return 0;

}

std::wstring BrowseFolder(HWND hParentWindow, LPCWSTR title, LPCWSTR startFromPath)

{

WCHAR path[MAX\_PATH];

BROWSEINFO bi = { 0 };

bi.hwndOwner = hParentWindow;

bi.lpszTitle = title;

bi.ulFlags = BIF\_RETURNONLYFSDIRS | BIF\_NEWDIALOGSTYLE;

bi.lpfn = BrowseCallbackProc;

bi.lParam = (LPARAM)startFromPath;

LPITEMIDLIST pidl = SHBrowseForFolder(&bi);

if (pidl != 0)

{

SHGetPathFromIDListW(pidl, path);

IMalloc\* imalloc = 0;

if (SUCCEEDED(SHGetMalloc(&imalloc)))

{

imalloc->Free(pidl);

imalloc->Release();

}

return std::wstring(path);

}

return std::wstring();

}

void SetControlEnabled(HWND hDlg, int controlID, bool enabled)

{

:: EnableWindow(GetDlgItem(hDlg, controlID), enabled);

}

std::wstring FileTimeToString(FILETIME ftime)

{

SYSTEMTIME utc;

::FileTimeToSystemTime(std::addressof(ftime), std::addressof(utc));

std::wostringstream os;

os << std::setfill<wchar\_t>('0') << std::setw(4) << utc.wYear << '-' << std::setw(2) << utc.wMonth

<< '-' << std::setw(2) << utc.wDay << ' ' << std::setw(2) << utc.wHour

<< ':' << std::setw(2) << utc.wMinute << ':' << std::setw(2) << utc.wSecond;

return os.str();

}

}