Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет   
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Операционные системы 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

Кирзнер А. П.

Проверил: Гриценко Н.Ю.

Минск 2023

**СОДЕРЖАНИЕ**

1 Цель работы 3

2 Теоретические сведения 4

3 Результат выполнения программы 5

Список использованных источников 7

Приложение А 8

# 1 ЦЕЛЬ РАБОТЫ

Построить файловый менеджер с поддержкой многозадачности и возможностью одновременной копии/перемещения файлов.

# 2 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Функция *CopyFile* копируетсуществующий файл в новый файл [1].

Функция *MoveFile* перемещает существующий файл или каталог, включая его дочерние элементы [2].

Интерфейс *IFileDialog* предоставляет методы, которые инициализируют, отображают и получают результаты из диалогового окна общих файлов. Интерфейс содержит следующие методы: *GetOptions* – возвращает текущие флаги, для которых задано управление поведением диалога, *SetOptions* – задает флаги для управления поведением диалогового окна, *GetFolder* – возвращает папку, выбранную в диалоговом окне, или, если диалоговое окно не отображается, папку, которую необходимо выбрать при открытии диалогового окна и многие другие методы [3].

Поток – это сущность в процессе, которую можно запланировать для выполнения. Все потоки процесса совместно используют его виртуальное адресное пространство и системные ресурсы. Каждый процесс запускается одним потоком, но может создавать дополнительные потоки из любого из его потоков [4].

Функция *CreateThread* создает новый поток для процесса. Создаваемый поток должен указать начальный адрес кода, который должен выполнить новый поток. Как правило, начальный адрес — это имя функции, определенной в коде программы. Эта функция принимает один параметр и возвращает значение *DWORD*. Процесс может содержать несколько потоков, одновременно выполняющих одну и ту же функцию [5].

Функция *ExitThread* завершает вызывающий поток. При вызове этой функции (явным образом или путем возврата из процедуры потока) стек текущего потока освобождается, все ожидающие операции ввода-вывода, инициированные потоком, который не связан с портом завершения, отменяется, а поток завершается. Если поток является последним потоком в процессе при вызове этой функции, процесс потока также завершается [6].

# 3 РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ ПРОГРАММЫ

При запуске приложения можно увидеть две кнопки для выбора файла для дальнейших манипуляций с ним (*Choose source file*) и для выбора папки, в которую будет произведено копирование или перемещение выбранного файла (*Choose destination folder*). Также в окне отображаются текущие выбранные файл и папка. Внизу окна располагаются две кнопки для произведения операций с файлом (*Copy chosen file*, *Move chosen file*). Изначально эти кнопки не доступны, пока пользователь не выберет всё необходимое для операции. Кроме всего прочего на окне расположено окно для вывода текущего состояния программы (рисунок 1).

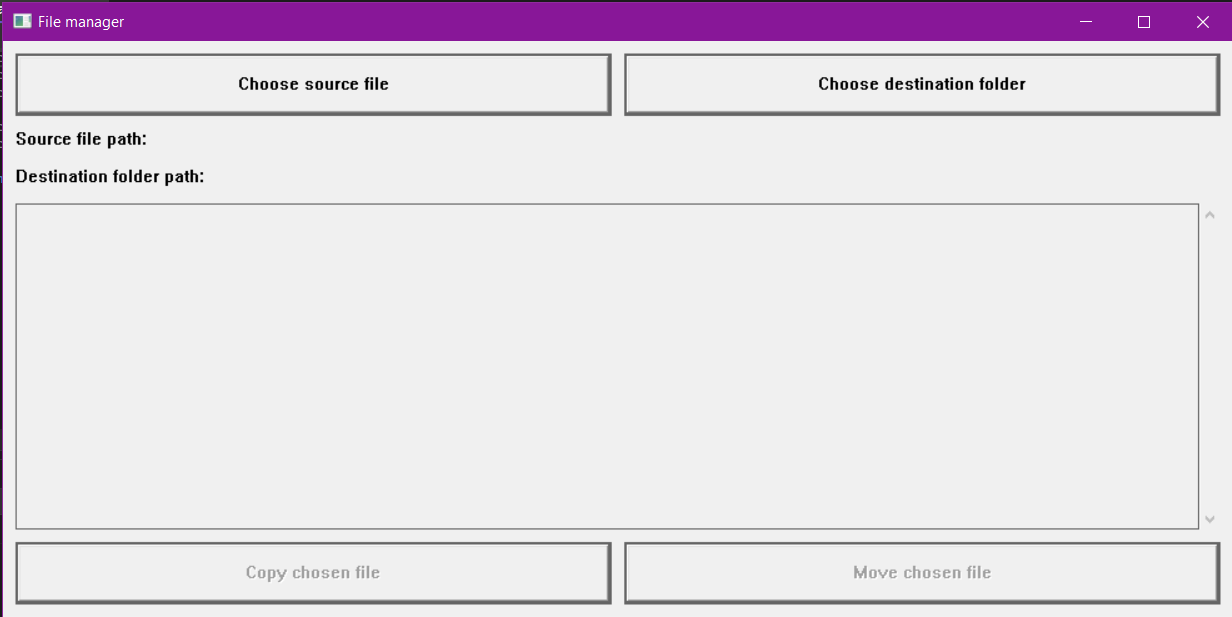


Рисунок 1 – Графический интерфейс программы

В соответствии с рисунком 2 можно заметить, что после выбора файла и папки кнопки для произведения операций стали доступны.

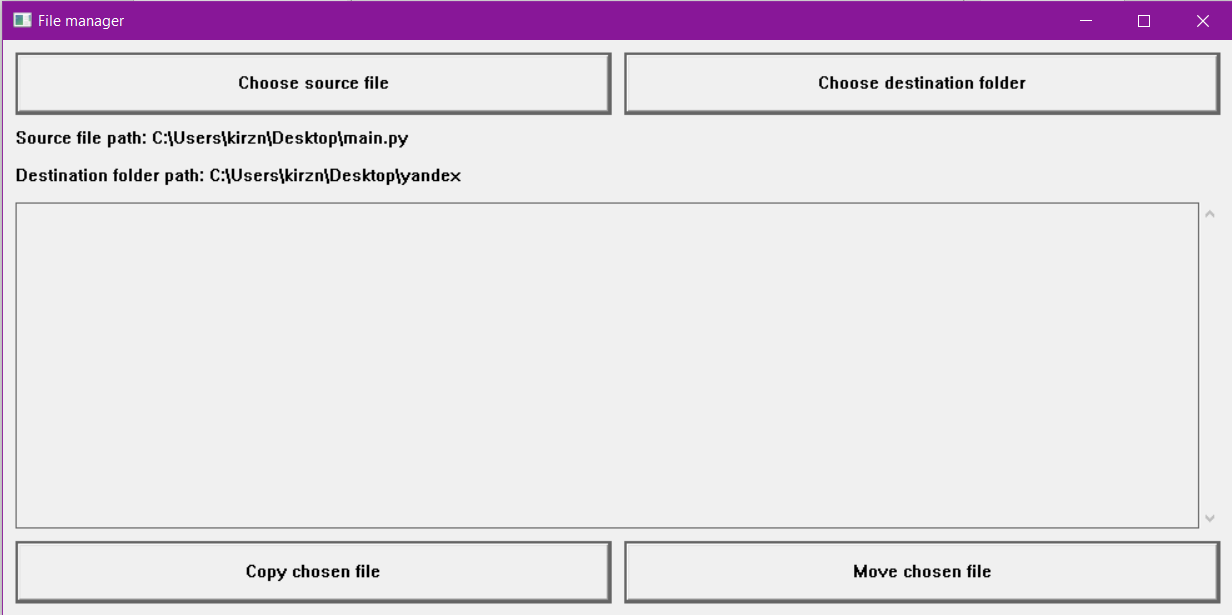


Рисунок 2 – Обновлённое состояние приложения

При нажатии кнопки *Copy chosen file* или *Move chosen file* будет происходить копирование или перемещение соответственно выбранного файла в указанную папку. В окне вывода отображается состояние программы (рисунок 3).

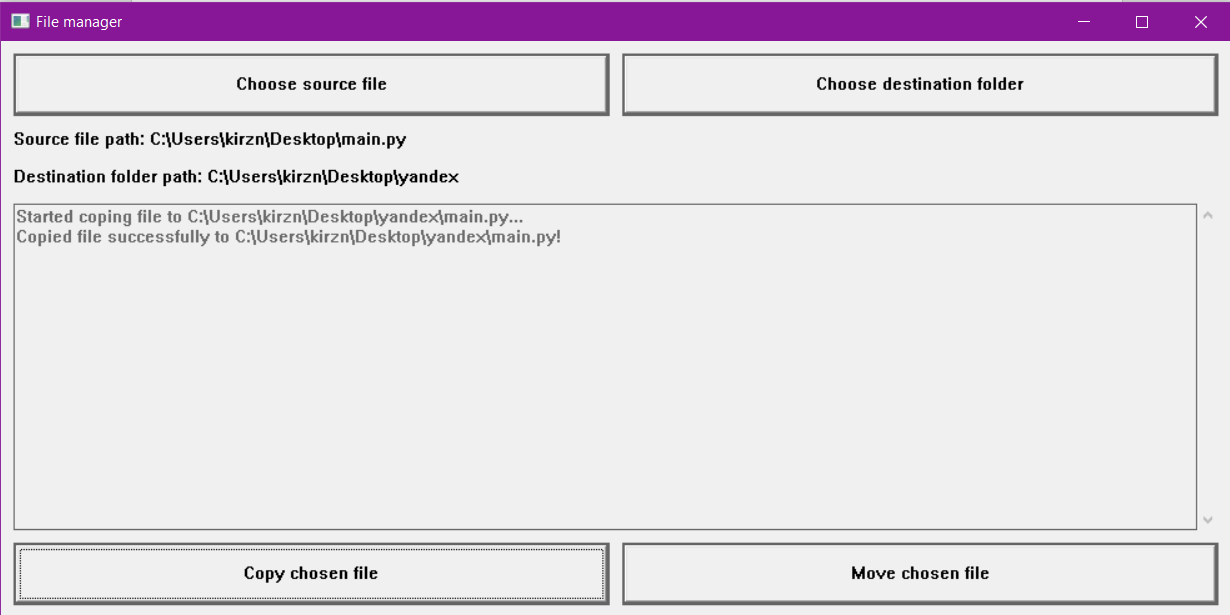


Рисунок 3 – Отображение состояния во время копирования

Программа поддерживает одновременное копирование/перемещение файлов. Это подтверждает окно вывода на рисунке 4.

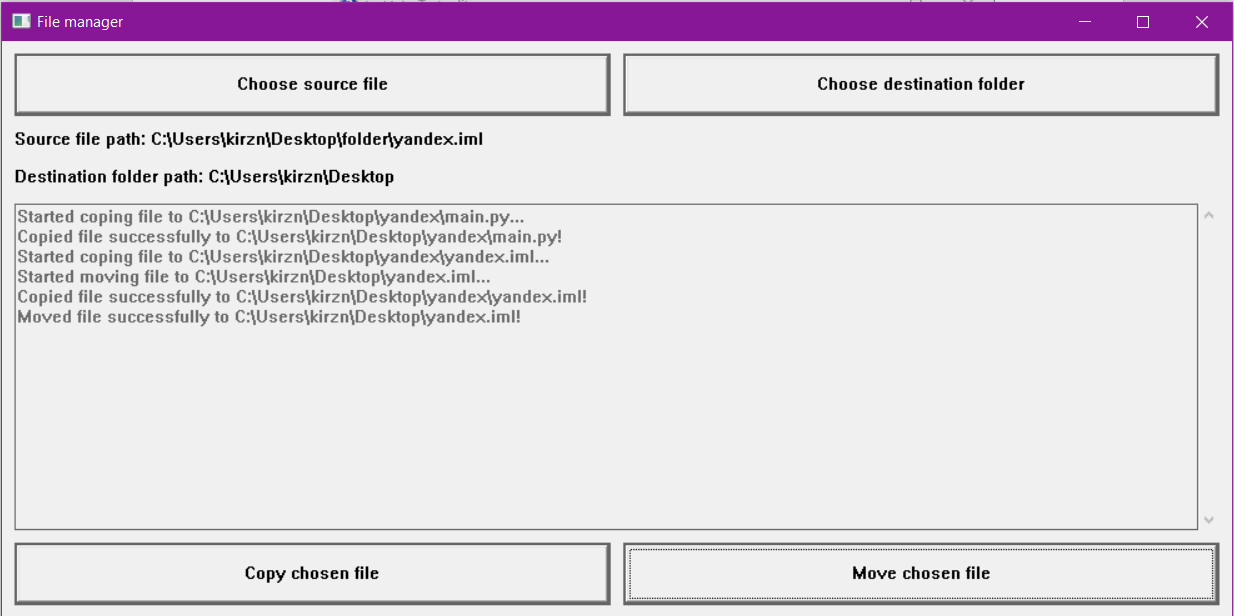


Рисунок 4 – Отображение многозадачности программы

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Функция CopyFile (winbase.h) [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/en-us/windows/win32/api/winbase/nf-winbase-copyfile>.

[2] Функция MoveFile (winbase.h) [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/en-us/windows/win32/api/winbase/nf-winbase-movefile>.

[3] Интерфейс IFileDialog (shobjidl\_core.h) [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/ru-ru/windows/win32/api/shobjidl_core/nn-shobjidl_core-ifiledialog>.

[4] Несколько потоков [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/ru-ru/windows/win32/procthread/multiple-threads>.

[5] Создание потоков [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/ru-ru/windows/win32/procthread/creating-threads>.

[6] Функция ExitThread (processthreadsapi.h) [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/ru-ru/windows/win32/api/processthreadsapi/nf-processthreadsapi-exitthread>.

# ПРИЛОЖЕНИЕ А

**(обязательное)**

**Исходный код программы**

**Файл Lab3.h**

#pragma once

#define OnFileButtonClicked 1

#define OnFolderButtonClicked 2

#define OnCopyButtonClicked 3

#define OnMoveButtonClicked 4

LRESULT CALLBACK SoftwareMainProcedure(HWND hWnd, UINT msg, WPARAM wp, LPARAM lp);

WNDCLASS NewWindowClass(HBRUSH BGColor, HCURSOR Cursor, HINSTANCE hInst, HICON Icon, LPCWSTR Name, WNDPROC Procedure);

HWND fileButton;

HWND folderButton;

IFileDialog\* pFileDialog = NULL;

WCHAR fileName[MAX\_PATH];

OPENFILENAMEW ofnFile;

LPWSTR folderName;

HRESULT folderHR;

DWORD folderOptions;

LPWSTR sourcePath;

LPWSTR destinationPath;

HWND sourceText;

HWND destinationText;

HWND copyButton;

HWND moveButton;

bool isFileChosen = false;

bool isFolderChosen = false;

HWND hEdit;

int currentProccesses = 0;

void CreateButtons(HWND hWnd);

void ChangeButtonsSizes(HWND hWnd);

void SetOpenFilesParams(HWND hWnd);

bool GetFolder(HWND hwnd);

void ChangeSourceText();

void ChangeDestinationText();

void CheckCopyAndMoveButtons();

DWORD CopyFunction(LPVOID lpParam);

DWORD MoveFunction(LPVOID lpParam);

**Файл Lab3.cpp**

#include <Windows.h>

#include <shlwapi.h>

#include <shlobj.h>

#include <stdio.h>

#include <thread>

#include "Lab3.h"

#include <string>

using std::string;

int WINAPI WinMain(HINSTANCE hInst, HINSTANCE hPrevInst, LPSTR args, int ncmdshows) {

WNDCLASS SoftwareMainClass = NewWindowClass((HBRUSH)COLOR\_WINDOW, LoadCursor(NULL, IDC\_ARROW), hInst, LoadIcon(NULL, IDI\_APPLICATION), L"MainWndClass", SoftwareMainProcedure);

if (!RegisterClass(&SoftwareMainClass)) { return -1; }

MSG SoftwareMainMessage = { 0 };

CreateWindow(L"MainWndClass", L"File manager", WS\_OVERLAPPEDWINDOW | WS\_VISIBLE, 100, 100, 500, 250, NULL, NULL, NULL, NULL);

while (GetMessage(&SoftwareMainMessage, NULL, NULL, NULL)) {

TranslateMessage(&SoftwareMainMessage);

DispatchMessage(&SoftwareMainMessage);

}

return 0;

}

WNDCLASS NewWindowClass(HBRUSH BGColor, HCURSOR Cursor, HINSTANCE hInst, HICON Icon, LPCWSTR Name, WNDPROC Procedure) {

WNDCLASS NWC = { 0 };

NWC.hCursor = Cursor;

NWC.hIcon = Icon;

NWC.hInstance = hInst;

NWC.lpszClassName = Name;

NWC.hbrBackground = BGColor;

NWC.lpfnWndProc = Procedure;

return NWC;

}

LRESULT CALLBACK SoftwareMainProcedure(HWND hWnd, UINT msg, WPARAM wp, LPARAM lp) {

switch (msg) {

case WM\_COMMAND:

switch (wp) {

case OnFileButtonClicked:

if (GetOpenFileNameW(&ofnFile)) {

ChangeSourceText();

isFileChosen = true;

CheckCopyAndMoveButtons();

}

break;

case OnFolderButtonClicked:

if (GetFolder(hWnd)) {

ChangeDestinationText();

isFolderChosen = true;

CheckCopyAndMoveButtons();

}

break;

case OnCopyButtonClicked:

if (!

CreateThread(

NULL,

0,

CopyFunction,

NULL,

0,

NULL)) {

MessageBox(hWnd, L"Can't handle operation!", L"Error", MB\_OK | MB\_ICONERROR);

}

else {

currentProccesses++;

}

break;

case OnMoveButtonClicked:

if (!

CreateThread(

NULL,

0,

MoveFunction,

NULL,

0,

NULL)) {

MessageBox(hWnd, L"Can't handle operation!", L"Error", MB\_OK | MB\_ICONERROR);

}

else {

currentProccesses++;

}

break;

}

break;

case WM\_CREATE: {

CreateButtons(hWnd);

SetOpenFilesParams(hWnd);

HRESULT hr = CoInitialize(NULL);

if (SUCCEEDED(hr)) {

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

if (SUCCEEDED(hr)) {

folderHR = pFileDialog->GetOptions(&folderOptions);

if (SUCCEEDED(folderHR)) {

pFileDialog->SetOptions(folderOptions | FOS\_PICKFOLDERS);

}

}

}

else {

MessageBox(hWnd, L"Can't create file dialog", L"Error", MB\_OK | MB\_ICONERROR);

PostQuitMessage(0);

}

break;

}

case WM\_SIZE:

ChangeButtonsSizes(hWnd);

break;

case WM\_GETMINMAXINFO:

{

LPMINMAXINFO lpMMI = (LPMINMAXINFO)lp;

lpMMI->ptMinTrackSize.x = 1000;

lpMMI->ptMinTrackSize.y = 500;

break;

}

case WM\_CLOSE:

if (currentProccesses > 0) {

MessageBox(hWnd, L"Can't close application while proccessing operation!", L"Error", MB\_OK | MB\_ICONERROR);

}

else {

DestroyWindow(hWnd);

}

break;

case WM\_DESTROY:

pFileDialog->Release();

CoUninitialize();

PostQuitMessage(0);

break;

default: return DefWindowProc(hWnd, msg, wp, lp);

}

}

void CreateButtons(HWND hWnd) {

RECT clientRect;

GetClientRect(hWnd, &clientRect);

fileButton = CreateWindow(

L"BUTTON",

L"Choose source file",

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

10,

10,

(clientRect.right - 30) / 2,

50,

hWnd,

(HMENU)OnFileButtonClicked,

(HINSTANCE)GetWindowLongPtr(hWnd, GWLP\_HINSTANCE),

NULL);

folderButton = CreateWindow(

L"BUTTON",

L"Choose destination folder",

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

20 + (clientRect.right - 30) / 2,

10,

(clientRect.right - 30) / 2,

50,

hWnd,

(HMENU)OnFolderButtonClicked,

(HINSTANCE)GetWindowLongPtr(hWnd, GWLP\_HINSTANCE),

NULL);

sourceText = CreateWindow(

L"STATIC",

L"Source file path: ",

WS\_VISIBLE | WS\_CHILD,

10,

70,

(clientRect.right - 20),

20,

hWnd,

NULL,

(HINSTANCE)GetWindowLongPtr(hWnd, GWLP\_HINSTANCE),

NULL);

destinationText = CreateWindow(

L"STATIC",

L"Destination folder path: ",

WS\_VISIBLE | WS\_CHILD,

10,

100,

(clientRect.right - 20),

20,

hWnd,

NULL,

(HINSTANCE)GetWindowLongPtr(hWnd, GWLP\_HINSTANCE),

NULL);

copyButton = CreateWindow(

L"BUTTON",

L"Copy chosen file",

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

10,

clientRect.bottom - 60,

(clientRect.right - 30) / 2,

50,

hWnd,

(HMENU)OnCopyButtonClicked,

(HINSTANCE)GetWindowLongPtr(hWnd, GWLP\_HINSTANCE),

NULL);

moveButton = CreateWindow(

L"BUTTON",

L"Move chosen file",

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

20 + (clientRect.right - 30) / 2,

clientRect.bottom - 60,

(clientRect.right - 30) / 2,

50,

hWnd,

(HMENU)OnMoveButtonClicked,

(HINSTANCE)GetWindowLongPtr(hWnd, GWLP\_HINSTANCE),

NULL);

hEdit = CreateWindow(

L"EDIT",

L"",

WS\_VISIBLE | WS\_CHILD | WS\_BORDER | ES\_AUTOVSCROLL | WS\_DISABLED | ES\_MULTILINE | WS\_VSCROLL,

10,

130,

(clientRect.right - 20),

clientRect.bottom - 200,

hWnd,

NULL,

(HINSTANCE)GetWindowLongPtr(hWnd, GWLP\_HINSTANCE),

NULL

);

}

void ChangeButtonsSizes(HWND hWnd) {

RECT clientRect;

GetClientRect(hWnd, &clientRect);

MoveWindow(

fileButton,

10,

10,

(clientRect.right - 30) / 2,

50,

TRUE);

MoveWindow(

folderButton,

20 + (clientRect.right - 30) / 2,

10,

(clientRect.right - 30) / 2,

50,

TRUE);

MoveWindow(

copyButton,

10,

clientRect.bottom - 60,

(clientRect.right - 30) / 2,

50,

TRUE);

MoveWindow(

moveButton,

20 + (clientRect.right - 30) / 2,

clientRect.bottom - 60,

(clientRect.right - 30) / 2,

50,

TRUE);

MoveWindow(

hEdit,

10,

130,

(clientRect.right - 20),

clientRect.bottom - 200,

TRUE);

}

void SetOpenFilesParams(HWND hWnd) {

ZeroMemory(&ofnFile, sizeof(ofnFile));

ofnFile.lStructSize = sizeof(ofnFile);

ofnFile.hwndOwner = hWnd;

ofnFile.lpstrFile = fileName;

ofnFile.nMaxFile = sizeof(fileName);

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

ofnFile.lpstrFileTitle = NULL;

ofnFile.nMaxFileTitle = 0;

ofnFile.lpstrInitialDir = NULL;

ofnFile.lpstrDefExt = NULL;

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

}

bool GetFolder(HWND hwnd) {

bool isSuccess = false;

folderHR = pFileDialog->Show(NULL);

if (SUCCEEDED(folderHR)) {

IShellItem\* pItem;

folderHR = pFileDialog->GetResult(&pItem);

if (SUCCEEDED(folderHR)) {

folderHR = pItem->GetDisplayName(SIGDN\_FILESYSPATH, &folderName);

if (SUCCEEDED(folderHR)) {

isSuccess = true;

}

pItem->Release();

}

}

return isSuccess;

}

void ChangeSourceText() {

WCHAR newSource[300];

wcscpy\_s(newSource, L"Source file path: ");

wcscat\_s(newSource, fileName);

SetWindowText(sourceText, (LPWSTR)newSource);

}

void ChangeDestinationText() {

WCHAR newDestination[300];

wcscpy\_s(newDestination, L"Destination folder path: ");

wcscat\_s(newDestination, folderName);

SetWindowText(destinationText, (LPWSTR)newDestination);

}

void CheckCopyAndMoveButtons() {

if (isFileChosen && isFolderChosen) {

EnableWindow(copyButton, TRUE);

EnableWindow(moveButton, TRUE);

}

else {

EnableWindow(copyButton, FALSE);

EnableWindow(moveButton, FALSE);

}

}

DWORD CopyFunction(LPVOID lpParam) {

WCHAR path[MAX\_PATH];

WCHAR filePath[MAX\_PATH];

WCHAR folder[MAX\_PATH];

wcscpy\_s(filePath, fileName);

wcscpy\_s(path, fileName);

wcscpy\_s(folder, folderName);

WCHAR name[MAX\_PATH];

PathStripPath(filePath);

lstrcpyW(name, filePath);

WCHAR newPath[MAX\_PATH];

PathCombine(newPath, folder, name);

SendMessage(hEdit, EM\_SETSEL, (WPARAM)-1, (LPARAM)-1);

WCHAR message[MAX\_PATH];

wcscpy\_s(message, L"Started coping file to ");

wcscat\_s(message, newPath);

wcscat\_s(message, L"...\r\n");

SendMessage(hEdit, EM\_SETSEL, (WPARAM)-1, (LPARAM)-1);

SendMessage(hEdit, EM\_REPLACESEL, 0, (LPARAM)message);

Sleep(10000);

if (CopyFile(path, newPath, TRUE)) {

SendMessage(hEdit, EM\_SETSEL, (WPARAM)-1, (LPARAM)-1);

wcscpy\_s(message, L"Copied file successfully to ");

wcscat\_s(message, newPath);

wcscat\_s(message, L"!\r\n");

SendMessage(hEdit, EM\_REPLACESEL, 0, (LPARAM)message);

currentProccesses--;

ExitThread(0);

return 0;

}

SendMessage(hEdit, EM\_SETSEL, (WPARAM)-1, (LPARAM)-1);

wcscpy\_s(message, L"Error while coping file to ");

wcscat\_s(message, newPath);

wcscat\_s(message, L"! :(\r\n");

SendMessage(hEdit, EM\_REPLACESEL, 0, (LPARAM)message);

currentProccesses--;

ExitThread(1);

return 1;

}

DWORD MoveFunction(LPVOID lpParam) {

WCHAR path[MAX\_PATH];

WCHAR folder[MAX\_PATH];

WCHAR filePath[MAX\_PATH];

wcscpy\_s(filePath, fileName);

wcscpy\_s(path, fileName);

wcscpy\_s(folder, folderName);

WCHAR name[MAX\_PATH];

PathStripPath(filePath);

lstrcpyW(name, filePath);

WCHAR newPath[MAX\_PATH];

PathCombine(newPath, folder, name);

WCHAR message[MAX\_PATH];

wcscpy\_s(message, L"Started moving file to ");

wcscat\_s(message, newPath);

wcscat\_s(message, L"...\r\n");

SendMessage(hEdit, EM\_SETSEL, (WPARAM)-1, (LPARAM)-1);

SendMessage(hEdit, EM\_REPLACESEL, 0, (LPARAM)message);

Sleep(10000);

if (MoveFileW(path, newPath)) {

SendMessage(hEdit, EM\_SETSEL, (WPARAM)-1, (LPARAM)-1);

wcscpy\_s(message, L"Moved file successfully to ");

wcscat\_s(message, newPath);

wcscat\_s(message, L"!\r\n");

SendMessage(hEdit, EM\_REPLACESEL, 0, (LPARAM)message);

currentProccesses--;

ExitThread(0);

return 0;

}

SendMessage(hEdit, EM\_SETSEL, (WPARAM)-1, (LPARAM)-1);

wcscpy\_s(message, L"Error while moving file to ");

wcscat\_s(message, newPath);

wcscat\_s(message, L"! :(\r\n");

SendMessage(hEdit, EM\_REPLACESEL, 0, (LPARAM)message);

currentProccesses--;

ExitThread(1);

return 1;

}