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. Функции API подсистемы памяти Win 32. Организация и контроль асинхронных операций ввода-вывода. Отображение файлов в память.»

Выполнил:

студент гр. 153504

Скроба А.П.

Проверил:

Гриценко Н.Ю.

Минск 2023

**СОДЕРЖАНИЕ**

[1 Цель работы 3](#_Toc146675523)

[2 Краткие теоретические сведения 4](#_Toc146675524)

[3 Результат выполнения программы 5](#_Toc146675525)

[Список использованных источников 9](#_Toc146675526)

[Приложение А 10](#_Toc146675527)

# 1 ЦЕЛЬ РАБОТЫ

Разработать файловый менеджер на платформе Win32 с поддержкой многозадачности и возможностью одновременной копии/перемещения файлов. Файловый менеджер должен обеспечивать управление файлами и каталогами, предоставлять функциональность копирования и перемещения файлов, а также поддерживать асинхронные операции ввода-вывода для повышения производительности и отзывчивости приложения.

**2 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

Многопоточность — свойство платформы или приложения, состоящее в том, что процесс, порождённый в операционной системе, может состоять из нескольких потоков, выполняющихся «параллельно», то есть без предписанного порядка во времени. При выполнении некоторых задач такое разделение может достичь более эффективного использования ресурсов вычислительной машины.

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.

Асинхронность в программировании — выполнение процесса в неблокирующем режиме системного вызова, что позволяет потоку программы продолжить обработку.

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.

# 3 РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ ПРОГРАММЫ

Результатом выполнения лабораторной работы является оконное приложение с интуитивно понятным пользовательским интерфейсом (рисунок 3.1).

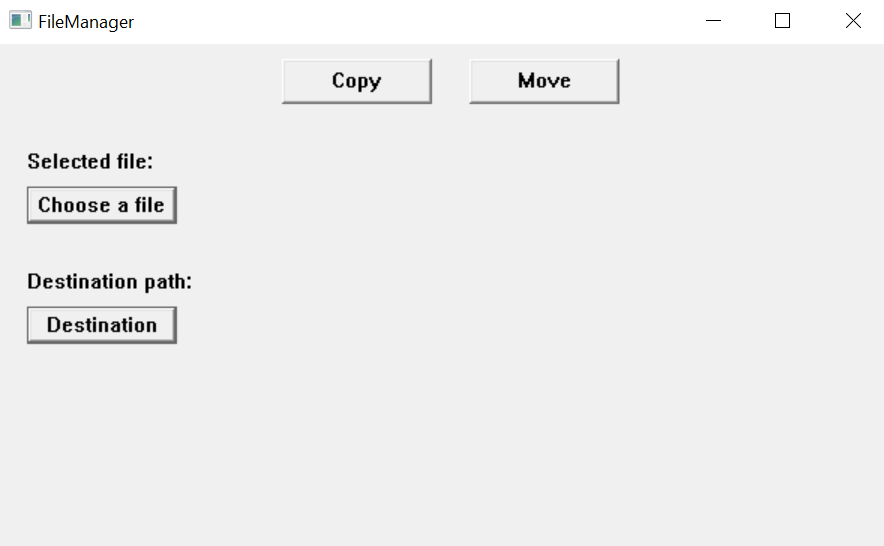
****

Рисунок 3.1 – Основной интерфейс приложения

При нажатии на кнопку «Choose a file» открывается файловый менеджер, отображающий и позволяющий выбрать файлы со всеми видами расширения. Аналогично при нажатии на «Destination» открывается файловый менеджер, позволяющий выбрать люблю папку, в которую будет происходить копирование или перемещение выбранного файла (рисунок 3.2, рисунок 3.3).

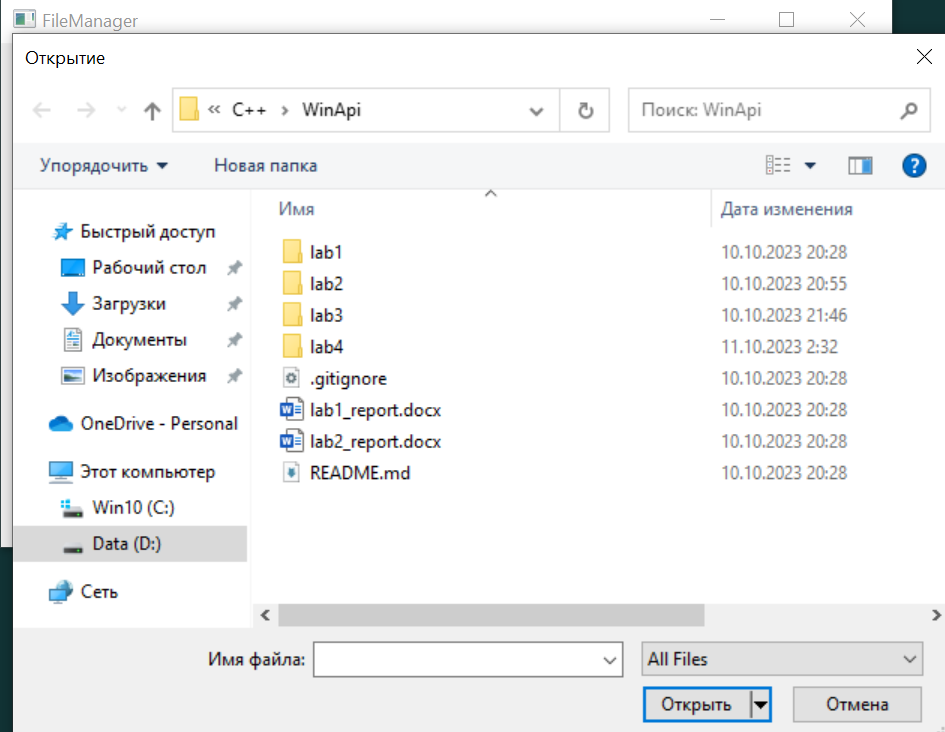


Рисунок 3.2 – Выбор файла для загрузки в программу

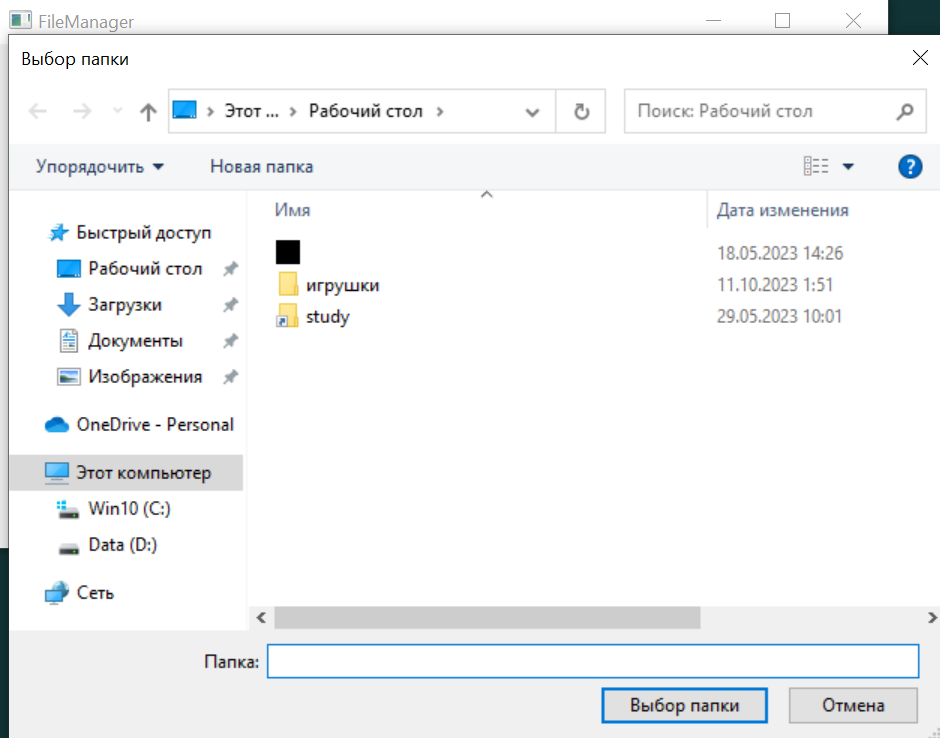


Рисунок 3.3 – Выбор папки для дальнейших действий

После выбора файла и папки в программе отображается их путь(рисунок 3.4).

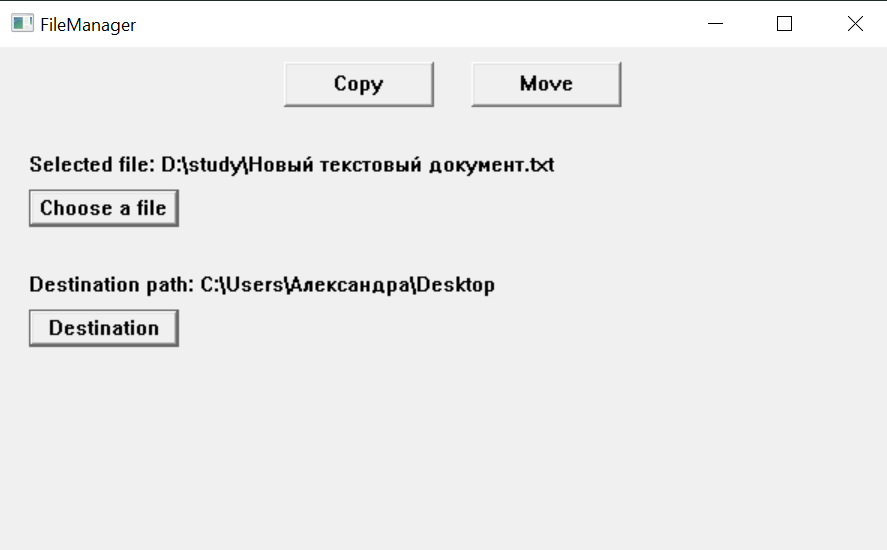
****

Рисунок 3.4 – Отображение путей выбранного файла и папки

После выбора файла и папки пользователь может либо скопировать данный файл в выбранную папку, нажав кнопку «Copy», либо переместить, нажав на кнопку «Move». При этом для удобства и наглядности асинхронности, реализована история действий программы (рисунок 3.5).

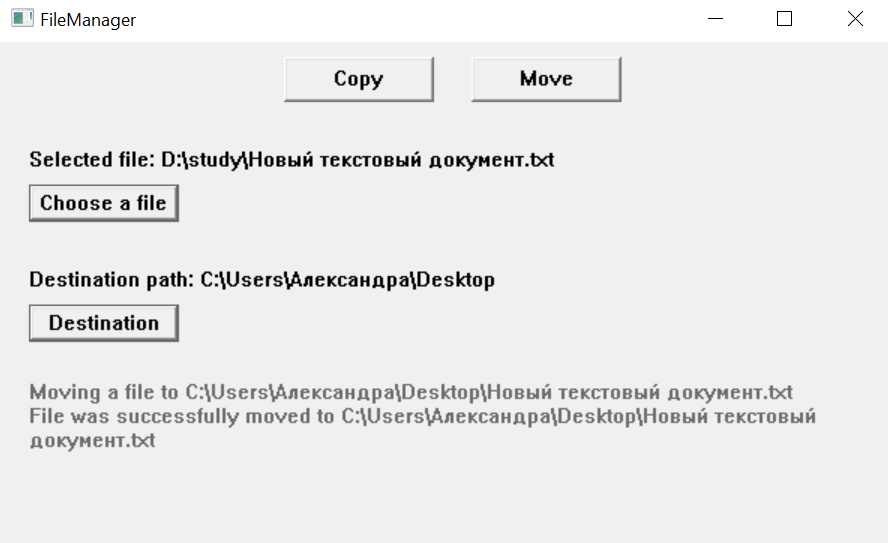


Рисунок 3.5 – История действий программы

Так же релизованна возможноть одновременной копии/перемещения файлов (рисунок 3.6).

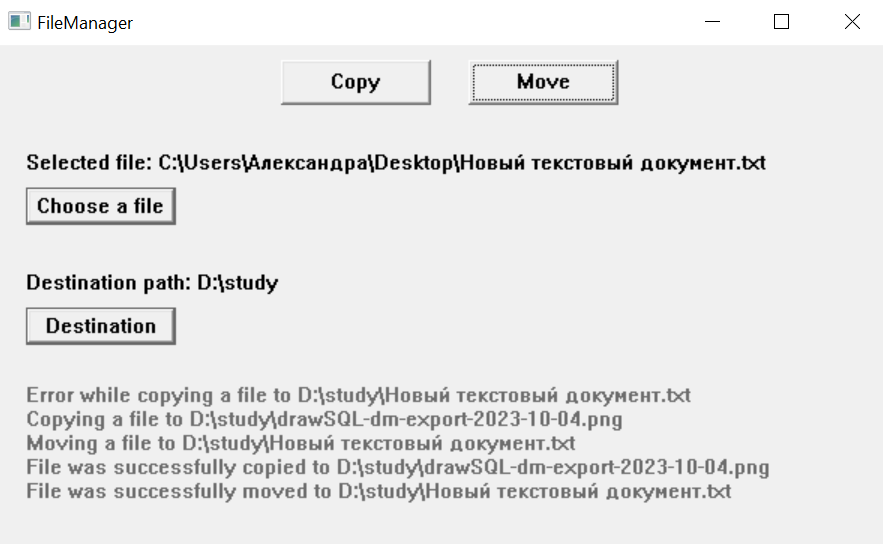


Рисунок 3.6 – Диалоговое окно об успешном сохранении файла

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Марапулец Ю. В. Системное программирование в WIN API. Учебное пособие, 2021. – 256 с

[2] Модель COM [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/com/the-component-object-model.

[3] Пример простейшей многопоточной программы на WinAPI [Электронный ресурс]. – Режим доступа: https://eax.me/winapi-threads.

# ПРИЛОЖЕНИЕ А

**(обязательное)**

**Исходный код программы**

Листинг 1 – Файл Resource.h:

#pragma once

#define OnFileButtonClicked 1

#define OnFolderButtonClicked 2

#define OnCopyButtonClicked 3

#define OnMoveButtonClicked 4

int currentProccesses = 0;

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

WNDCLASS WindowClass(HBRUSH BGColor, HCURSOR Cursor, HINSTANCE hInst, LPCWSTR Name, WNDPROC wc);

IFileDialog\* pFileDialog = NULL;

WCHAR fileName[MAX\_PATH];

OPENFILENAMEW ofn;

LPWSTR folderName;

HRESULT folderHR;

DWORD folderOptions;

HWND selectedText;

HWND destinationText;

HWND hEditControl;

void MainWndAddWidget(HWND hWnd);

void SetOpenFileParams(HWND hWnd);

bool GetFolder(HWND hwnd);

void ChangeSelectedTextText();

void ChangeDestinationText();

DWORD CopyFunction(LPVOID lpParam);

DWORD MoveFunction(LPVOID lpParam);

Листинг 2 **–** Файл Source.cpp:

#include <Windows.h>

#include <string.h>

#include <shlwapi.h>

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

#include <shlobj.h>

#include "Resource.h"

int WINAPI WinMain(\_In\_ HINSTANCE hInstance, \_In\_opt\_ HINSTANCE hPrevInstance, \_In\_ LPSTR lpCmdLine, \_In\_ int nShowCmd) {

WNDCLASS MainClass = WindowClass((HBRUSH)COLOR\_WINDOW, LoadCursor(NULL, IDC\_ARROW), hInstance, L"MainWndClass", WndProc);

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

MSG SoftwareMainMessage = { 0 };

CreateWindow(L"MainWndClass", L"FileManager", WS\_OVERLAPPEDWINDOW | WS\_VISIBLE, 200, 100, 610, 375, NULL, NULL, NULL, NULL);

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

TranslateMessage(&SoftwareMainMessage);

DispatchMessage(&SoftwareMainMessage);

}

return 0;

}

WNDCLASS WindowClass(HBRUSH BGColor, HCURSOR Cursor, HINSTANCE hInst, LPCWSTR Name, WNDPROC wc) {

WNDCLASS nwc = { };

nwc.hbrBackground = BGColor;

nwc.hCursor = Cursor;

nwc.hInstance = hInst;

nwc.lpszClassName = Name;

nwc.lpfnWndProc = wc;

return nwc;

}

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

switch (uMsg) {

case WM\_COMMAND:

switch (wParam) {

case OnFileButtonClicked:

if (GetOpenFileNameW(&ofn)) {

ChangeSelectedTextText();

}

break;

case OnFolderButtonClicked:

if (GetFolder(hwnd)) {

ChangeDestinationText();

}

break;

case OnCopyButtonClicked:

if (!CreateThread(

NULL, // Атрибуты безопасности потока

0, // Размер стека потока

CopyFunction, // Функция, которую будет выполнять поток

NULL, // Параметры, передаваемые в CopyFunction

0, // Флаги создания потока

NULL)) { // Указатель на переменную для сохранения ID потока

MessageBox(hwnd, L"Oops. Something went wrong.", L"Error!", MB\_OK | MB\_ICONERROR);

}

else {

currentProccesses++;

}

break;

case OnMoveButtonClicked:

if (!CreateThread(NULL, 0, MoveFunction, NULL, 0, NULL)) {

MessageBox(hwnd, L"Oops. Something went wrong.", L"Error!", MB\_OK | MB\_ICONERROR);

}

else {

currentProccesses++;

}

break;

}

break;

case WM\_CREATE: {

MainWndAddWidget(hwnd);

SetOpenFileParams(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\_CLOSE:

if (currentProccesses > 0) {

MessageBox(hwnd, L"Сannot close the application until the program execution is finished.", L"Error!", MB\_OK | MB\_ICONERROR);

}

else {

DestroyWindow(hwnd);

}

break;

case WM\_DESTROY:

pFileDialog->Release();

CoUninitialize();

PostQuitMessage(0);

break;

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

}

}

void MainWndAddWidget(HWND hwnd) {

CreateWindow(L"button", L"Copy", WS\_VISIBLE | WS\_CHILD , 190, 10, 100, 30, hwnd, (HMENU)OnCopyButtonClicked, NULL, NULL);

CreateWindow(L"button", L"Move", WS\_VISIBLE | WS\_CHILD , 315, 10, 100, 30, hwnd, (HMENU)OnMoveButtonClicked, NULL, NULL);

selectedText = CreateWindow(L"static", L"Selected file: ", WS\_VISIBLE | WS\_CHILD, 20, 70, 580, 20, hwnd, NULL, NULL, NULL);

CreateWindow(L"button", L"Сhoose a file", WS\_VISIBLE | WS\_CHILD | WS\_BORDER, 20, 95, 100, 25, hwnd, (HMENU)OnFileButtonClicked, NULL, NULL);

destinationText = CreateWindow( L"static", L"Destination path: ", WS\_VISIBLE | WS\_CHILD, 20, 150, 580, 20, hwnd, NULL, NULL, NULL);

CreateWindow(L"button", L"Destination", WS\_VISIBLE | WS\_CHILD | WS\_BORDER, 20, 175, 100, 25, hwnd, (HMENU)OnFolderButtonClicked, NULL, NULL);

hEditControl = CreateWindow(L"edit", L" ", WS\_VISIBLE | WS\_CHILD | ES\_AUTOVSCROLL | WS\_DISABLED | ES\_MULTILINE, 20, 225, 560, 100, hwnd, NULL, NULL, NULL);

}

void SetOpenFileParams(HWND hwnd) {

ZeroMemory(&ofn, sizeof(ofn));

ofn.lStructSize = sizeof(ofn);

ofn.hwndOwner = hwnd;

ofn.lpstrFile = fileName;

ofn.nMaxFile = sizeof(fileName);

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

ofn.lpstrFileTitle = NULL;

ofn.nMaxFileTitle = 0;

ofn.lpstrInitialDir = L"D:\\С++\\WinApi";

ofn.lpstrDefExt = NULL;

ofn.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 ChangeSelectedTextText() {

WCHAR newText[300];

wcscpy\_s(newText, L"Selected file: ");

wcscat\_s(newText, fileName);

SetWindowText(selectedText, (LPWSTR)newText);

}

void ChangeDestinationText() {

WCHAR newText[300];

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

wcscat\_s(newText, folderName);

SetWindowText(destinationText, (LPWSTR)newText);

}

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(hEditControl, EM\_SETSEL, (WPARAM)-1, (LPARAM)-1);

WCHAR message[MAX\_PATH];

wcscpy\_s(message, L"Copying a file to ");

wcscat\_s(message, newPath);

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

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

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

Sleep(10000);

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

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

wcscpy\_s(message, L"File was successfully copied to ");

wcscat\_s(message, newPath);

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

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

currentProccesses--;

ExitThread(0);

return 0;

}

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

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

wcscat\_s(message, newPath);

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

SendMessage(hEditControl, 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"Moving a file to ");

wcscat\_s(message, newPath);

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

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

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

Sleep(10000);

if (MoveFileW(path, newPath)) {

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

wcscpy\_s(message, L"File was successfully moved to ");

wcscat\_s(message, newPath);

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

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

currentProccesses--;

ExitThread(0);

return 0;

}

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

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

wcscat\_s(message, newPath);

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

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

currentProccesses--;

ExitThread(1);

return 1;

}