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 группы 153502

Кузнецов Е. А.

Проверил: Гриценко Н.Ю.

Минск 2023

**CОДЕРЖАНИЕ**

1Цель работы...............................................................................................................3

2 Теоретические сведения...........................................................................................4

3 Описание функций программы................................................................................5

Список использованных источников..........................................................................7

Приложение А...............................................................................................................8

**1 ЦЕЛЬ РАБОТЫ**

Целью выполнения лабораторной работы является 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.

В качестве задачи необходимо разработать асинхронное приложение для копирования больших файлов с возможностью приостановки и возобновления операции.

**2 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

Функция *CreateFile()* создает или открывает файл или устройство ввода-вывода. Наиболее часто используемые устройства ввода-вывода: файл, файловый поток, каталог, физический диск, том, буфер консоли, ленточный накопитель, ресурс связи и канал. Функция возвращает дескриптор, который можно использовать для доступа к файлу или устройству для различных типов операций ввода-вывода в зависимости от файла или устройства, а также указанных флагов и атрибутов [1].

Функция *ReadFile()* считывает данные из указанного файла или устройства ввода-вывода. Операции чтения выполняются в позиции, указанной указателем файла, если устройство поддерживает.

Эта функция предназначена как для синхронных, так и для асинхронных операций [2].

Функция *CreateThread()* создает поток для выполнения в виртуальном адресном пространстве вызывающего процесса [3].

**3 ОПИСАНИЕ ФУНКЦИЙ ПРОГРАММЫ**

Согласно формулировке задачи, были спроектированы следующие функции программы:

– копирование файла;

– приостановка и возобновление копирования.

Для начала работы необходимо нажать на кнопку «Старт» (рисунок 1).

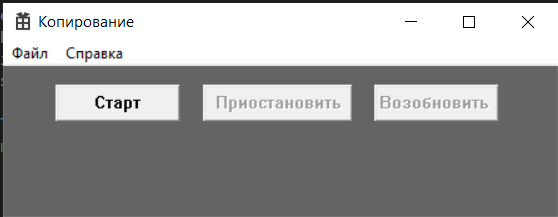


Рисунок 1 – Интерфейс программы

Далее следует выбрать файл в открывшемся проводнике (рисунок 2) и нажать кнопку «Открыть».

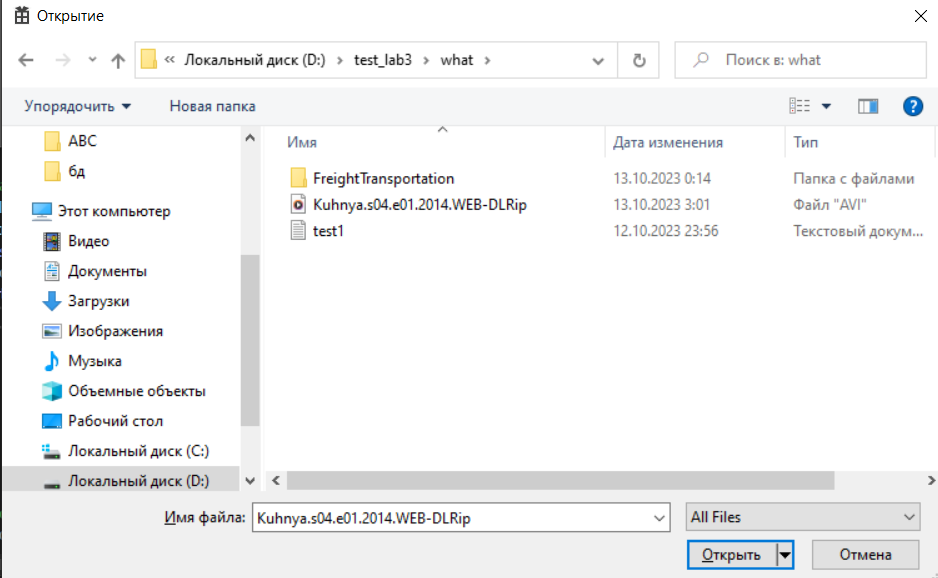


Рисунок 2 – Проводник выбора файла

Далее для выбора директории для копирования необходимо нажать на кнопку «Сохранить» (рисунок 3).

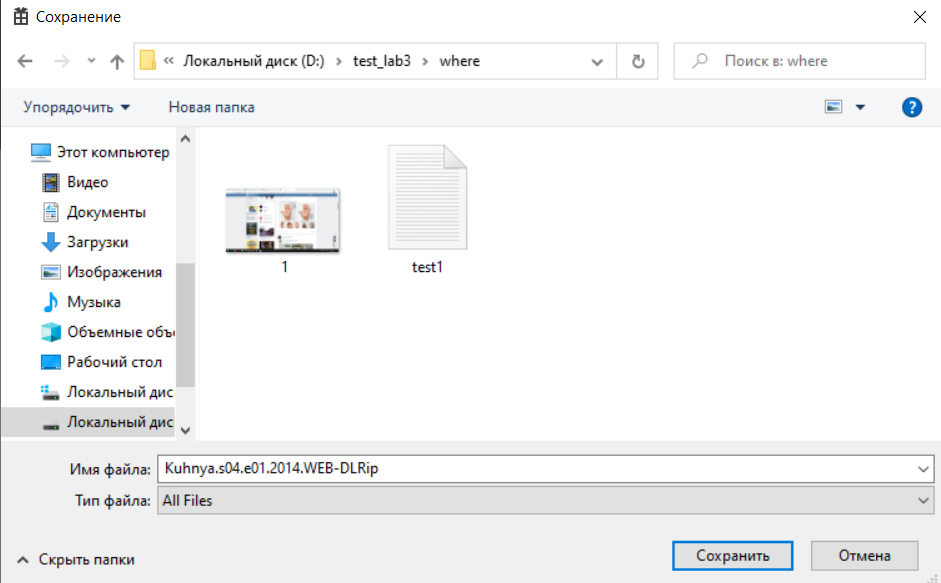


Рисунок 3 – Проводник сохранения файла

Для приостановки копирования необходимо нажать на кнопку «Приостановить». Для возобновления копирования необходимо нажать на кнопку «Возобновить». При завершении копирования появляется сообщение об успешном окончании копирования(рисунок 4).

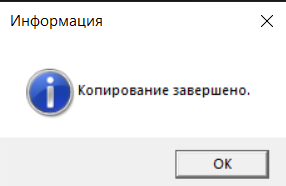


Рисунок 4 – Сообщение об успешном окончании копирования

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

[1] Функция CreateFileW [Электронный ресурс]. – Электронные данные. – Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/api/fileapi/nf-fileapi-createfilew

[2] Функция ReadFile [Электронный ресурс]. – Электронные данные. – Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/api/fileapi/nf-fileapi-readfile

[3] Функция CreateThread() [Электронный ресурс]. – Электронные данные. – Режим доступа: https://learn.microsoft.com/ru-ru/cpp/standard-library/thread-class?view=msvc-170

ПРИЛОЖЕНИЕ АЛистинг кода

**Файл LABA3.h**

#include <thread>

#include <atomic>

#include "framework.h"

#include "laba3.h"

#include <string>

#include <iostream>

#include <commdlg.h>

#include <fstream>

#include <sstream>

#include <commctrl.h>

#define MAX\_LOADSTRING 100

#define IDM\_START 1001

#define IDM\_PAUSE 1002

#define IDM\_RESUME 1003

#define WM\_APP\_UPDATE\_UI (WM\_APP + 1)

//HANDLE hPauseEvent = CreateEvent(NULL, FALSE, FALSE, NULL);

HANDLE hPauseEvent = CreateEvent(NULL, FALSE, FALSE, NULL);

HANDLE hCopyEvent = CreateEvent(NULL, FALSE, FALSE, NULL);

HWND hWnd;

struct CopyFileParams {

UINT msgId;

std::wstring source;

std::wstring target;

};

// Глобальные переменные:

HINSTANCE hInst; // текущий экземпляр

WCHAR szTitle[MAX\_LOADSTRING] = L"Копирование"; // Текст строки заголовка

WCHAR szWindowClass[MAX\_LOADSTRING]; // имя класса главного окна

std::atomic<bool> isCopying{ false }; // Флаг копирования

std::atomic<bool> isPaused{ false }; // Флаг приостановки

std::wstring sourceFilePath;

std::wstring targetFilePath;

//void CopyFileAsync(HWND hWnd, const std::wstring& source, const std::wstring& target);

//DWORD WINAPI CopyFileAsyncThread(LPVOID lpParam);

//void CopyFileAsync(HWND hWnd, const std::wstring& source, const std::wstring& target);

DWORD WINAPI CopyFileAsync(LPVOID lpParam);

// Отправить объявления функций, включенных в этот модуль кода:

ATOM MyRegisterClass(HINSTANCE hInstance);

BOOL InitInstance(HINSTANCE, int);

LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);

INT\_PTR CALLBACK About(HWND, UINT, WPARAM, LPARAM);

int APIENTRY wWinMain(\_In\_ HINSTANCE hInstance,

\_In\_opt\_ HINSTANCE hPrevInstance,

\_In\_ LPWSTR lpCmdLine,

\_In\_ int nCmdShow)

{

UNREFERENCED\_PARAMETER(hPrevInstance);

UNREFERENCED\_PARAMETER(lpCmdLine);

//LoadStringW(hInstance, IDS\_APP\_TITLE, szTitle, MAX\_LOADSTRING);

LoadStringW(hInstance, IDC\_LABA3, szWindowClass, MAX\_LOADSTRING);

MyRegisterClass(hInstance);

// Выполнить инициализацию приложения:

if (!InitInstance (hInstance, nCmdShow))

{

return FALSE;

}

HACCEL hAccelTable = LoadAccelerators(hInstance, MAKEINTRESOURCE(IDC\_LABA3));

MSG msg;

// Цикл основного сообщения:

while (GetMessage(&msg, nullptr, 0, 0))

{

if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))

{

TranslateMessage(&msg);

DispatchMessage(&msg);

}

}

return (int) msg.wParam;

}

//

// ФУНКЦИЯ: MyRegisterClass()

//

// ЦЕЛЬ: Регистрирует класс окна.

//

ATOM MyRegisterClass(HINSTANCE hInstance)

{

WNDCLASSEXW wcex;

wcex.cbSize = sizeof(WNDCLASSEX);

wcex.style = CS\_HREDRAW | CS\_VREDRAW;

wcex.lpfnWndProc = WndProc;

wcex.cbClsExtra = 0;

wcex.cbWndExtra = 0;

wcex.hInstance = hInstance;

wcex.hIcon = LoadIcon(hInstance, MAKEINTRESOURCE(IDI\_LABA3));

wcex.hCursor = LoadCursor(nullptr, IDC\_ARROW);

wcex.hbrBackground = (HBRUSH)(COLOR\_WINDOW + 2);

wcex.lpszMenuName = MAKEINTRESOURCEW(IDC\_LABA3);

wcex.lpszClassName = szWindowClass;

wcex.hIconSm = LoadIcon(wcex.hInstance, MAKEINTRESOURCE(IDI\_SMALL));

return RegisterClassExW(&wcex);

}

// ФУНКЦИЯ: InitInstance(HINSTANCE, int)

// ЦЕЛЬ: Сохраняет маркер экземпляра и создает главное окно

// КОММЕНТАРИИ:

// В этой функции маркер экземпляра сохраняется в глобальной переменной, а также

// создается и выводится главное окно программы.

BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)

{

hInst = hInstance; // Сохранить маркер экземпляра в глобальной переменной

hWnd = CreateWindowW(szWindowClass, szTitle, WS\_OVERLAPPEDWINDOW,

CW\_USEDEFAULT, CW\_USEDEFAULT, 460, 180, nullptr, nullptr, hInstance, nullptr);

if (!hWnd)

{

return FALSE;

}

ShowWindow(hWnd, nCmdShow);

UpdateWindow(hWnd);

return TRUE;

}

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)

{

switch (message)

{

case WM\_CREATE: {

CreateWindowW(L"BUTTON", L"Старт", WS\_VISIBLE | WS\_CHILD | BS\_PUSHBUTTON,

42, 15, 100, 30, hWnd, (HMENU)IDM\_START, hInst, nullptr);

CreateWindowW(L"BUTTON", L"Приостановить", WS\_VISIBLE | WS\_CHILD | BS\_PUSHBUTTON,

160, 15, 120, 30, hWnd, (HMENU)IDM\_PAUSE, hInst, nullptr);

CreateWindowW(L"BUTTON", L"Возобновить ", WS\_VISIBLE | WS\_CHILD | BS\_PUSHBUTTON,

297, 15, 100, 30, hWnd, (HMENU)IDM\_RESUME, hInst, nullptr);

SendMessage(hWnd, WM\_APP\_UPDATE\_UI, 0, 0);

}

break;

case WM\_COMMAND:

{

int wmId = LOWORD(wParam);

switch (wmId)

{

case IDM\_START:

if (!isCopying) {

OPENFILENAME ofn;

TCHAR szFile[MAX\_PATH] = L"";

ZeroMemory(&ofn, sizeof(ofn));

ofn.lStructSize = sizeof(ofn);

ofn.hwndOwner = hWnd;

ofn.lpstrFile = szFile;

ofn.nMaxFile = sizeof(szFile);

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

ofn.nFilterIndex = 1;

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

if (GetOpenFileName(&ofn)) {

sourceFilePath = szFile;

// Вызов диалога выбора целевого файла

ZeroMemory(&ofn, sizeof(ofn));

ofn.lStructSize = sizeof(ofn);

ofn.hwndOwner = hWnd;

ofn.lpstrFile = szFile;

ofn.nMaxFile = sizeof(szFile);

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

ofn.nFilterIndex = 1;

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

if (GetSaveFileName(&ofn)) {

targetFilePath = szFile;

isCopying = true;

SendMessage(hWnd, WM\_APP\_UPDATE\_UI, 0, 0);

CopyFileParams params;

//params.hWnd = hWnd;

params.msgId = WM\_APP\_UPDATE\_UI;

params.source = sourceFilePath;

params.target = targetFilePath;

// Создаем поток

HANDLE hThread = CreateThread(NULL, 0, CopyFileAsync, &params, 0, NULL);

if (hThread == NULL) {

MessageBox(hWnd, L"Не удается создать поток.", L"Ошибка", MB\_OK | MB\_ICONERROR);

isCopying = false;

PostMessage(hWnd, WM\_APP\_UPDATE\_UI, 0, 0);

}

}

}

}

break;

case IDM\_PAUSE:

if (isCopying) {

isPaused = true;

// Отправить сообщение об изменении состояния

SendMessage(hWnd, WM\_APP\_UPDATE\_UI, 0, 0);

}

break;

case IDM\_RESUME:

if (isCopying) {

isPaused = false;

// Отправить сообщение об изменении состояния

SendMessage(hWnd, WM\_APP\_UPDATE\_UI, 0, 0);

}

break;

case IDM\_ABOUT:

DialogBox(hInst, MAKEINTRESOURCE(IDD\_ABOUTBOX), hWnd, About);

break;

case IDM\_EXIT:

if (MessageBox(hWnd, L"Вы действительно хотите закрыть окно?", L"Выход", MB\_OKCANCEL) == IDOK) {

DestroyWindow(hWnd);

}

return 0;

break;

default:

return DefWindowProc(hWnd, message, wParam, lParam);

}

}

break;

case WM\_APP\_UPDATE\_UI:

EnableWindow(GetDlgItem(hWnd, IDM\_START), !isCopying);

EnableWindow(GetDlgItem(hWnd, IDM\_PAUSE), isCopying && !isPaused);

EnableWindow(GetDlgItem(hWnd, IDM\_RESUME), isCopying && isPaused);

break;

case WM\_CLOSE:

if (MessageBox(hWnd, L"Вы действительно хотите закрыть окно?", L"Выход", MB\_OKCANCEL) == IDOK) {

DestroyWindow(hWnd);

}

return 0;

break;

case WM\_DESTROY:

PostQuitMessage(0);

break;

default:

return DefWindowProc(hWnd, message, wParam, lParam);

}

return 0;

}

// Обработчик сообщений для окна "О программе".

INT\_PTR CALLBACK About(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)

{

UNREFERENCED\_PARAMETER(lParam);

switch (message)

{

case WM\_INITDIALOG:

return (INT\_PTR)TRUE;

case WM\_COMMAND:

if (LOWORD(wParam) == IDOK || LOWORD(wParam) == IDCANCEL)

{

EndDialog(hDlg, LOWORD(wParam));

return (INT\_PTR)TRUE;

}

break;

}

return (INT\_PTR)FALSE;

}

// Функция для асинхронного копирования файла

DWORD WINAPI CopyFileAsync(LPVOID lpParam) {

isCopying = true;

isPaused = false;

CopyFileParams\* params = reinterpret\_cast<CopyFileParams\*>(lpParam);

HANDLE hSourceFile = CreateFileW(sourceFilePath.c\_str(), GENERIC\_READ, FILE\_SHARE\_READ, NULL, OPEN\_EXISTING, FILE\_ATTRIBUTE\_NORMAL, NULL);

if (hSourceFile == INVALID\_HANDLE\_VALUE) {

isCopying = false;

PostMessage(hWnd, params->msgId, 0, 0);

MessageBox(hWnd, L"Не удается открыть исходный файл.", L"Ошибка", MB\_OK | MB\_ICONERROR);

return 1;

}

HANDLE hTargetFile = CreateFileW(targetFilePath.c\_str(), GENERIC\_WRITE, 0, NULL, CREATE\_ALWAYS, FILE\_ATTRIBUTE\_NORMAL, NULL);

if (hTargetFile == INVALID\_HANDLE\_VALUE) {

CloseHandle(hSourceFile);

isCopying = false;

PostMessage(hWnd, WM\_APP\_UPDATE\_UI, 0, 0);

MessageBox(hWnd, L"Не удается создать целевой файл.", L"Ошибка", MB\_OK | MB\_ICONERROR);

return 1;

}

char buffer[4096];

DWORD bytesRead, bytesWritten;

while (ReadFile(hSourceFile, buffer, sizeof(buffer), &bytesRead, NULL) && bytesRead > 0) {

if (isPaused) {

// Если операция приостановлена, ждем, пока не будет возобновлена

while (isPaused) {

Sleep(100);

}

}

if (!WriteFile(hTargetFile, buffer, bytesRead, &bytesWritten, NULL) || bytesRead != bytesWritten) {

// Произошла ошибка записи

CloseHandle(hSourceFile);

CloseHandle(hTargetFile);

isCopying = false;

PostMessage(hWnd, WM\_APP\_UPDATE\_UI, 0, 0);

MessageBox(hWnd, L"Произошла ошибка записи в целевой файл.", L"Ошибка", MB\_OK | MB\_ICONERROR);

return 1;

}

}

isCopying = false;

PostMessage(hWnd, WM\_APP\_UPDATE\_UI, 0, 0);

MessageBox(hWnd, L"Копирование завершено.", L"Информация", MB\_OK | MB\_ICONINFORMATION);

CloseHandle(hSourceFile);

CloseHandle(hTargetFile);

ExitThread(0);

return 0;

}