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

Михалевич М.П.

Проверил:

Гриценко Н.Ю.

Минск 2023

СОДЕРЖАНИЕ

[1 Цели работы 3](#_Toc147863804)

[2 Kраткие теоретические сведения 4](#_Toc147863805)

[3 Результаты выполнения лабораторной работы 5](#_Toc147863806)

[Вывод 6](#_Toc147863807)

[Список использованных источников 7](#_Toc147863808)

[Приложение А 8](#_Toc147863809)

## 1 ЦЕЛИ РАБОТЫ

Целью данной лабораторной работы является разработка асинхронного приложения, предназначенного для копирования больших файлов с встроенной функцией приостановки и возобновления операции копирования.

## 2 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Win32 API - это набор функций и процедур, предоставляемых операционной системой Windows для разработки приложений. Он служит связующим звеном между приложением и аппаратным обеспечением компьютера, обеспечивая возможность создания разнообразных оконных приложений для Windows. В рамках этого интерфейса разработчики получают доступ к широкому спектру возможностей, включая обработку событий, взаимодействие с системными ресурсами (например, файлами и памятью), и многие другие задачи.

Работа с файлами в Win32 API предоставляет разработчикам мощные инструменты для управления файловой системой операционной системы Windows. Основными функциями для работы с файлами являются CreateFile(), ReadFile(), WriteFile(), и CloseHandle(). Сначала необходимо открыть файл с помощью CreateFile(), получив дескриптор файла. Затем можно читать и записывать данные с использованием соответствующих функций ReadFile() и WriteFile(). После завершения операций с файлом, важно закрыть файл с помощью CloseHandle(), чтобы освободить ресурсы.[1]

В процессе выполнения данной лабораторной работы, были применены следующие теоретические сведения:

1. Оконное приложение: Kод представляет собой графическое оконное приложение, разработанное с использованием Win32 API. Главное окно приложения содержит элементы управления, такие как панель прогресса (Progress Bar), метка состояния (Status Label) и кнопки управления операцией копирования.

2. Использование элементов управления: Для создания интерфейса пользователя использованы элементы управления, такие как кнопки (Button), панель прогресса и метку состояния. Эти элементы обеспечивают взаимодействие пользователя с приложением

3. Многопоточное программирование: Для асинхронного копирования файлов создан новый поток выполнения с использованием стандартной библиотеки C++ (std::thread). Это позволяет приложению выполнять операции копирования в фоновом режиме, не блокируя интерфейс пользователя.

4. Работа с файлами: Kод выполняет операции копирования файлов из одной директории в другую. Для этого используются классы std::ifstream и std::ofstream для чтения и записи файлов соответственно.

5. Обработка событий: Приложение реагирует на события, такие как нажатие кнопок "Выбрать файл", "Выбрать директорию", "Старт" и "Пауза". Каждое из этих событий вызывает соответствующие функции и операции, обеспечивая удобное взаимодействие пользователя с приложением.

## 3 РЕЗУЛЬТАТЫ ВЫПОЛНЕНИЯ ЛАБОРАТОРНОЙ РАБОТЫ

В рамках проведения лабораторной работы было создано асинхронное приложение, предназначенное для эффективного копирования больших файлов. Пользователю предоставлена возможность выбора как исходного файла, так и целевой директории, в которую необходимо выполнить копирование. После нажатия кнопки "Старт", пользователь информирован о ходе операции копирования с помощью отображения процентного выполнения. Подробности работы программы представлены на рисунке 3.1.

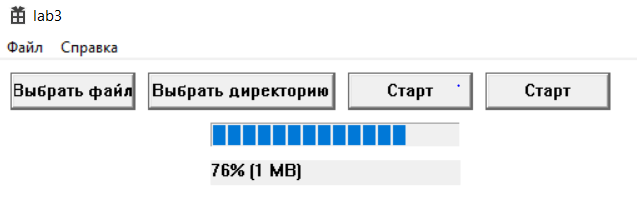


Рисунок 3.1 – Результат работы программы

## ВЫВОД

В ходе выполнения лабораторной работы было разработано асинхронное приложение, способное эффективно копировать большие файлы с возможностью приостановки и возобновления операции. Программа предоставляет удобный графический интерфейс для выбора исходного файла и целевой директории, а также динамически отображает текущий прогресс копирования. Это позволяет пользователям управлять процессом копирования и следить за его выполнением, делая работу с большими файлами более удобной и контролируемой.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Щупак Ю. Win32 API. Разработка приложений для Windows. – СПб: Питер, 2008. – 592 с.: ип.

[2] [Электронный ресурс]. – Режим доступа: [https://learn.microsoft.com/ru-ru/windows/win32/api – Дата доступа 08.10.2023](https://learn.microsoft.com/ru-ru/windows/win32/api%20–%20Дата%20доступа%2020.09.2023)

[3] [Электронный ресурс]. – Режим доступа:  [https://learn.microsoft.com/ru-ru/windows/win32/fileio/opening-a-file-for-reading-or-writing – Дата доступа 08.10.2023](https://stackoverflow.com/questions/17187265/how-to-group-radio-box-buttons-using-win32-api%20–%20Дата%20доступа%2023.09.2023)

[4] [Электронный ресурс]. – Режим доступа: https://ru.stackoverflow.com/questions/836063/ – Дата доступа 08.10.2023

## ПРИЛОЖЕНИЕ А

**(обязательное)**

**Листинг кода**

#include "framework.h"

#include "lab3.h"

#include <commdlg.h>

#include <shlobj.h>

#include <string>

#include <thread>

#include <fstream>

HWND hwndButtonPause, hwndProgress, hwndStatusLabel;

std::wstring sourceFilePath, destFolderPath;

bool paused = false, copying = false;

#define MAX\_LOADSTRING 100

// Глобальные переменные:

HINSTANCE hInst;

WCHAR szTitle[MAX\_LOADSTRING];

WCHAR szWindowClass[MAX\_LOADSTRING];

ATOM MyRegisterClass(HINSTANCE hInstance);

BOOL InitInstance(HINSTANCE, int);

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

void UpdateProgress(float percentage, const std::wstring& status) {

SendMessage(hwndProgress, PBM\_SETPOS, (int)(percentage \* 100), 0);

SetWindowText(hwndStatusLabel, status.c\_str());

}

void CopyFileAsync() {

std::ifstream source(sourceFilePath, std::ios::binary);

std::ofstream dest(destFolderPath + L"\\" + sourceFilePath.substr(sourceFilePath.find\_last\_of(L"\\") + 1), std::ios::binary);

if (!source.is\_open() || !dest.is\_open()) {

MessageBox(NULL, L"Error opening files!", L"Error", MB\_OK | MB\_ICONERROR);

return;

}

source.seekg(0, std::ios::end);

size\_t fileSize = source.tellg();

source.seekg(0, std::ios::beg);

const size\_t bufferSize = 4096;

char buffer[bufferSize];

size\_t copiedBytes = 0;

while (source.read(buffer, bufferSize) && !source.eof()) {

while (paused) {

std::this\_thread::sleep\_for(std::chrono::milliseconds(100));

}

dest.write(buffer, source.gcount());

copiedBytes += source.gcount();

float progress = static\_cast<float>(copiedBytes) / fileSize;

std::wstring status = std::to\_wstring((int)(progress \* 100)) + L"% (" + std::to\_wstring(copiedBytes / (1024 \* 1024)) + L" MB)";

UpdateProgress(progress, status);

std::this\_thread::sleep\_for(std::chrono::milliseconds(1)); // Simulate slower copying for demonstration

}

source.close();

dest.close();

copying = false;

UpdateProgress(0.0f, L"");

MessageBox(NULL, L"Copy completed!", L"Success", MB\_OK);

}

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\_LAB3, szWindowClass, MAX\_LOADSTRING);

MyRegisterClass(hInstance);

if (!InitInstance(hInstance, nCmdShow))

{

return FALSE;

}

HACCEL hAccelTable = LoadAccelerators(hInstance, MAKEINTRESOURCE(IDC\_LAB3));

MSG msg;

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

{

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

{

TranslateMessage(&msg);

DispatchMessage(&msg);

}

}

return (int)msg.wParam;

}

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\_LAB3));

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

wcex.hbrBackground = (HBRUSH)(COLOR\_WINDOW + 1);

wcex.lpszMenuName = MAKEINTRESOURCEW(IDC\_LAB3);

wcex.lpszClassName = szWindowClass;

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

return RegisterClassExW(&wcex);

}

BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)

{

hInst = hInstance;

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

CW\_USEDEFAULT, 0, CW\_USEDEFAULT, 0, 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:

hwndProgress = CreateWindowEx(0, PROGRESS\_CLASS, NULL, WS\_CHILD | WS\_VISIBLE, 170, 50, 200, 20, hWnd, NULL, hInst, NULL);

hwndStatusLabel = CreateWindow(L"STATIC", L"", WS\_CHILD | WS\_VISIBLE, 170, 80, 200, 20, hWnd, NULL, hInst, NULL);

CreateWindow(L"BUTTON", L"Выбрать файл", WS\_TABSTOP | WS\_VISIBLE | WS\_CHILD | BS\_DEFPUSHBUTTON, 10, 10, 100, 30, hWnd, (HMENU)IDC\_BUTTONSOURCE, hInst, NULL);

CreateWindow(L"BUTTON", L"Выбрать директорию", WS\_TABSTOP | WS\_VISIBLE | WS\_CHILD | BS\_DEFPUSHBUTTON, 120, 10, 150, 30, hWnd, (HMENU)IDC\_BUTTONDEST, hInst, NULL);

CreateWindow(L"BUTTON", L"Старт", WS\_TABSTOP | WS\_VISIBLE | WS\_CHILD | BS\_DEFPUSHBUTTON, 280, 10, 100, 30, hWnd, (HMENU)IDC\_BUTTONSTART, hInst, NULL);

hwndButtonPause = CreateWindow(L"BUTTON", L"Пауза", WS\_CHILD | WS\_VISIBLE | BS\_DEFPUSHBUTTON, 390, 10, 100, 30, hWnd, (HMENU)IDC\_BUTTONPAUSE, hInst, NULL);

break;

case WM\_COMMAND:

{

int wmId = LOWORD(wParam);

switch (wmId)

{

case IDC\_BUTTONSOURCE:

{

OPENFILENAME ofn = {};

wchar\_t fileName[MAX\_PATH] = {};

ofn.lStructSize = sizeof(ofn);

ofn.hwndOwner = hWnd;

ofn.lpstrFile = fileName;

ofn.nMaxFile = sizeof(fileName);

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

ofn.nFilterIndex = 1;

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

if (GetOpenFileName(&ofn)) {

sourceFilePath = fileName;

}

}

break;

case IDC\_BUTTONDEST:

{

BROWSEINFO bi = {};

bi.lpszTitle = L"Choose Destination Folder";

LPITEMIDLIST pidl = SHBrowseForFolder(&bi);

if (pidl != 0) {

wchar\_t path[MAX\_PATH];

SHGetPathFromIDList(pidl, path);

IMalloc\* imalloc = 0;

if (SUCCEEDED(SHGetMalloc(&imalloc))) {

imalloc->Free(pidl);

imalloc->Release();

}

destFolderPath = path;

}

}

break;

case IDC\_BUTTONSTART:

{

if (!sourceFilePath.empty() && !destFolderPath.empty() && !copying) {

copying = true;

std::thread(CopyFileAsync).detach();

}

}

break;

case IDC\_BUTTONPAUSE:

{

if (copying) {

paused = !paused;

SetWindowText(hwndButtonPause, paused ? L"Старт" : L"Пауза");

}

}

break;

}

}

break;

case WM\_DESTROY:

PostQuitMessage(0);

break;

default:

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

}

return 0;

}