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 гр.153502 Леоненко А.О.

Проверил ассистент кафедры информатики Гриценко Н.Ю.

Минск 2023

**СОДЕРЖАНИЕ**

[1 Теоретические сведения 3](#_Toc148047681)

[2 Формулировка задачи 4](#_Toc148047682)

[3 Описание функций программы 5](#_Toc148047683)

[3.1 Интерфейс программы 5](#_Toc148047684)

[3.2 Начало копирования 5](#_Toc148047685)

[3.3 Приостановка и возобновление копирования 6](#_Toc148047686)

[3.4 Сообщения программы 6](#_Toc148047687)

[Список использованных источников 8](#_Toc148047688)

[Приложение а 9](#_Toc148047689)

1 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, почтовый слопот 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].

Объект t*hread* можно использовать для наблюдения за потоком выполнения в приложении и управления этим потоком. Объект *thread*, созданный с помощью конструктора по умолчанию, не связан с потоком выполнения. Объект *thread*, созданный с помощью вызываемого объекта, создает новый поток выполнения и вызывает вызываемый объект в этом *thread*. *Thread* объекты можно перемещать, но не копировать, поэтому поток выполнения может быть связан только с одним *thread* объектом.

Каждый поток выполнения имеет уникальный идентификатор типа *thread*::*id*. Функция *this\_thread::get\_id* возвращает идентификатор вызывающего потока. Функция-член *thread::get\_id* возвращает идентификатор потока, управляемого thread объектом. Для созданного *thread* по умолчанию объекта *thread::get\_id* метод возвращает объект с одинаковым значением для всех созданных по умолчанию *thread* объектов и отличается от значения *this\_thread::get\_id*, возвращаемого для любого потока выполнения, который может быть присоединен во время вызова [3].

2 ФОРМУЛИРОВКА ЗАДАЧИ

Целью выполнения лабораторной работы является создание оконного приложения на *Win32 API*, обладающее минимальным функционалом, позволяющим отработать базовые навыки написания программы на *Win32* *API*.

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

3 ОПИСАНИЕ ФУНКЦИЙ ПРОГРАММЫ

Согласно формулировке задачи, были спроектированы следующие функции программы:

– Возможность ввода адреса файла-источника;

– Возможность ввода конечного пути копирования;

– Копирование файла;

– Приостановка и возобновление копирования.

3.1 Интерфейс программы

Интерфейс программы изображен на рисунке 1. В окне программы расположены поле для ввода адреса файла для копирования, поле для ввода конечного пути копирования, а также кнопки начала копирования и остановки или возобновления копирования.

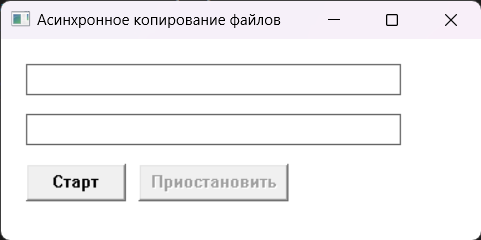


Рисунок 1 – Интерфейс программы

3.2 Начало копирования

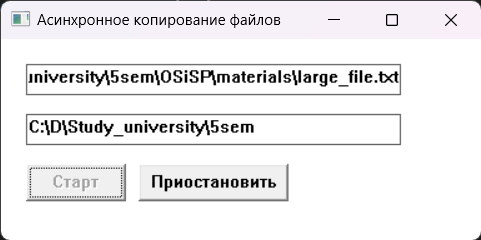


Рисунок 2 – Интерфейс во время операции копирования

Для начала процесса копирования необходимо в соответствующие поля ввести адрес файла и конечный путь копирования. После следует нажать кнопу «Старт». Интерфейс после выполнения данных действий изображен на рисунке 2. На время выполнения операции копирования кнопка «Старт» блокируется, в то время как кнопка «Приостановить» становится доступной для нажатия.

3.3 Приостановка и возобновление копирования

Для приостановки операции копирования необходимо нажать кнопку «Приостановить». Интерфейс после нажатия данной кнопки изображен на рисунке 3. Для возобновления операции копирования необходимо нажать на кнопку «Возобновить».

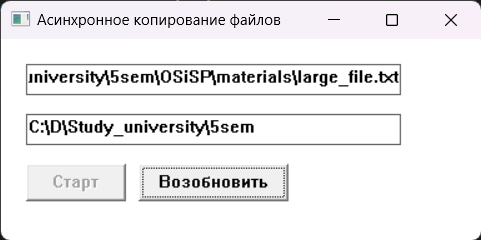


Рисунок 3 – Интерфейс после приостановки копирования

3.4 Сообщения программы

После успешного копирования файла программа уведомляет об осуществленной операции с помощью специального окна. Пример изображен на рисунке 4.

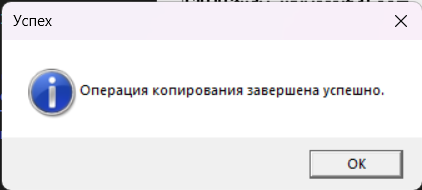


Рисунок 4 – Сообщение об успешном окончании копирования

В связи с некорректным вводом пользователя программа также уведомляет об ошибке с помощью специального окна. Примеры изображены на рисунках 5 и 6.

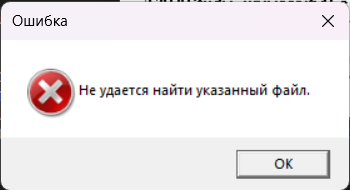


Рисунок 5 – Сообщение при некорректном вводе адреса файла

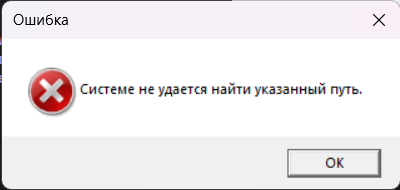


Рисунок 6 – Сообщение при некорректном вводе пути копирования

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

ПРИЛОЖЕНИЕ А

Листинг кода

Файл main.cpp

#include <Windows.h>

#include <tchar.h>

#include <thread>

#include <mutex>

#include <condition\_variable>

#include <filesystem>

namespace fs = std::filesystem;

#define ID\_BUTTON\_START 1001

#define ID\_BUTTON\_PAUSE 1002

HWND hwndButtonStart, hwndButtonPauseResume, hwndEditSource, hwndEditDestination;

HANDLE hThread;

std::mutex mtx;

std::condition\_variable cv;

bool isPaused = false;

TCHAR sourcePath[MAX\_PATH], destinationPath[MAX\_PATH];

void SetButtonsState(BOOL isCopying) {

EnableWindow(hwndButtonStart, !isCopying);

EnableWindow(hwndButtonPauseResume, isCopying);

}

void CopyFileAsync(LPCTSTR sourcePath, LPCTSTR destinationPath) {

SetButtonsState(TRUE);

HANDLE hSourceFile = CreateFile(sourcePath, GENERIC\_READ, 0, NULL, OPEN\_EXISTING, FILE\_ATTRIBUTE\_NORMAL, NULL);

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

DWORD error = GetLastError();

LPWSTR errorMessage;

FormatMessage(

FORMAT\_MESSAGE\_ALLOCATE\_BUFFER | FORMAT\_MESSAGE\_FROM\_SYSTEM,

NULL,

error,

0,

(LPWSTR)&errorMessage,

0,

NULL

);

MessageBox(NULL, errorMessage, \_T("Ошибка"), MB\_OK | MB\_ICONERROR);

LocalFree(errorMessage);

SetButtonsState(FALSE);

return;

}

fs::path path(sourcePath);

std::wstring fileName = path.filename().wstring();

fs::path destinationDirectory = fs::path(destinationPath) / fileName;

destinationPath = destinationDirectory.c\_str();

HANDLE hDestinationFile = CreateFile(destinationPath, GENERIC\_WRITE, 0, NULL, CREATE\_ALWAYS, FILE\_ATTRIBUTE\_NORMAL, NULL);

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

DWORD error = GetLastError();

LPWSTR errorMessage;

FormatMessage(

FORMAT\_MESSAGE\_ALLOCATE\_BUFFER | FORMAT\_MESSAGE\_FROM\_SYSTEM,

NULL,

error,

0,

(LPWSTR)&errorMessage,

0,

NULL

);

MessageBox(NULL, errorMessage, \_T("Ошибка"), MB\_OK | MB\_ICONERROR);

LocalFree(errorMessage);

CloseHandle(hSourceFile);

SetButtonsState(FALSE);

return;

}

BYTE buffer[4096];

DWORD bytesRead, bytesWritten;

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

std::unique\_lock<std::mutex> lock(mtx);

cv.wait(lock, [] { return !isPaused; });

WriteFile(hDestinationFile, buffer, bytesRead, &bytesWritten, NULL);

}

CloseHandle(hSourceFile);

CloseHandle(hDestinationFile);

MessageBox(NULL, \_T("Операция копирования завершена успешно."), \_T("Успех"), MB\_OK | MB\_ICONINFORMATION);

SetButtonsState(FALSE);

}

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

switch (uMsg) {

case WM\_CREATE:

hwndEditSource = CreateWindow(\_T("EDIT"), \_T(""), WS\_CHILD | WS\_VISIBLE | WS\_BORDER | ES\_AUTOHSCROLL, 20, 20, 300, 25, hwnd, NULL, NULL, NULL);

hwndEditDestination = CreateWindow(\_T("EDIT"), \_T(""), WS\_CHILD | WS\_VISIBLE | WS\_BORDER | ES\_AUTOHSCROLL, 20, 60, 300, 25, hwnd, NULL, NULL, NULL);

hwndButtonStart = CreateWindow(\_T("BUTTON"), \_T("Старт"), WS\_CHILD | WS\_VISIBLE, 20, 100, 80, 30, hwnd, (HMENU)ID\_BUTTON\_START, NULL, NULL);

hwndButtonPauseResume = CreateWindow(\_T("BUTTON"), \_T("Приостановить"), WS\_CHILD | WS\_VISIBLE, 110, 100, 120, 30, hwnd, (HMENU)ID\_BUTTON\_PAUSE, NULL, NULL);

SetButtonsState(FALSE);

break;

case WM\_COMMAND:

switch (LOWORD(wParam)) {

case ID\_BUTTON\_START:

isPaused = FALSE;

GetWindowText(hwndEditSource, sourcePath, MAX\_PATH);

GetWindowText(hwndEditDestination, destinationPath, MAX\_PATH);

std::thread(CopyFileAsync, sourcePath, destinationPath).detach();

break;

case ID\_BUTTON\_PAUSE:

{

std::lock\_guard<std::mutex> lock(mtx);

isPaused = !isPaused;

}

if (!isPaused)

cv.notify\_one();

SetWindowText(hwndButtonPauseResume, isPaused ? \_T("Возобновить") : \_T("Приостановить"));

break;

}

break;

case WM\_DESTROY:

PostQuitMessage(0);

break;

default:

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

}

return 0;

}

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nCmdShow) {

WNDCLASS wc = { 0 };

wc.lpfnWndProc = WindowProc;

wc.hInstance = hInstance;

wc.lpszClassName = \_T("AsyncCopyAppClass");

RegisterClass(&wc);

HWND hwnd = CreateWindow(\_T("AsyncCopyAppClass"), \_T("Асинхронное копирование файлов"), WS\_OVERLAPPEDWINDOW, 100, 100, 400, 200, NULL, NULL, hInstance, NULL);

ShowWindow(hwnd, nCmdShow);

MSG msg;

while (GetMessage(&msg, NULL, 0, 0)) {

TranslateMessage(&msg);

DispatchMessage(&msg);

}

return 0;

}