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, ОТОБРАЖЕНИЕ ФАЙЛОВ В ПАМЯТЬ**

Студент О. Л. Дайнович

Преподаватель Н. Ю. Гриценко

Минск 2023

**СОДЕРЖАНИЕ**

[1 Цель работы 3](#_Toc148078232)

[2 Теоретические сведения 4](#_Toc148078233)

[3 Описание функций программы 5](#_Toc148078234)

[Заключение 6](#_Toc148078235)

[Список использованных источников 7](#_Toc148078236)

[Приложение А (обязательное) Листинг кода 8](#_Toc148078237)

# **1 ЦЕЛЬ РАБОТЫ**

Целью выполнения лабораторной работы является создание оконного приложения на Win32 API, обладающее возможностью отображения активных процессов и информации об используемой ими памятью. Данное приложение будет запрашивать информацию об активных процессах и отображать ей в виде списка в окне приложения. Также, данное приложение должно иметь возможность записывать полученную информацию в файл.

# **2 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

Управление памятью в Windows осуществляется через Win32 API и позволяет эффективно управлять выделением и освобождением памяти. Функции API подсистемы памяти, такие как VirtualAlloc и VirtualFree, предоставляют возможность выделения и освобождения виртуальной памяти. Это особенно важно при работе с большими объемами данных и динамическими структурами. [1]

Ввод-вывод в Windows обеспечивается разнообразными средствами, включая файловый ввод-вывод, сокеты и межпроцессное взаимодействие. Функции API, такие как ReadFile и WriteFile, позволяют работать с файлами и устройствами. Операции ввода-вывода могут быть синхронными и асинхронными, позволяя создавать быстрые и отзывчивые приложения. [2]

Windows предоставляет множество расширенных возможностей ввода-вывода, включая работу с многозадачностью, потоками, асинхронными операциями и виртуальными файлами. Это позволяет разрабатывать мощные и эффективные приложения с возможностью параллельной обработки ввода-вывода.

API подсистемы памяти Windows, такие как VirtualAlloc и VirtualQuery, предоставляют доступ к различным функциям, включая выделение резервных областей памяти, управление атрибутами доступа и контроль регионов памяти.

Windows предоставляет механизмы для асинхронных операций ввода-вывода, такие как механизмы перекрытия и использование асинхронных функций API. Это позволяет приложениям выполнять операции ввода-вывода параллельно без блокировки пользовательского интерфейса. [3]

Все эти возможности делают Windows мощной и гибкой платформой для разработки приложений, требующих эффективного управления памятью и высокоэффективного ввода-вывода.

# **3 ОПИСАНИЕ ФУНКЦИЙ ПРОГРАММЫ**

В ходе выполнения лабораторной работы было разработано оконное приложение для визуализации структуры файловой системы и отслеживания изменений в реальном времени (рисунок 1).

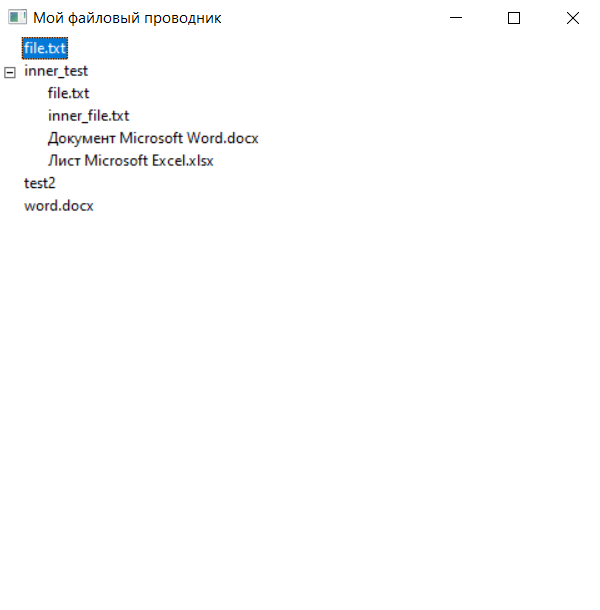


Рисунок 1 — Графический интерфейс приложения

Приложение изображает файловое дерево и асинхронно следит за изменениями в директориях в реальном времени. В случае изменения в файловой системе приложение отреагирует на них и изменит вид файлового дерева.

# **ЗАКЛЮЧЕНИЕ**

В результате лабораторной работы были изучены принципы работы с памятью Win32 API. Было создано оконное приложение, позволяющее получать важную информацию о файлах и корректировать ее в реальном времени.

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Microsoft Learn [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/learnwin32/.
2. Начало работы с классическими приложениями для Windows, которые используют API Win32 [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/desktop-programming.

[3] Сообщения окна [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/learnwin32/window-messages.

# **ПРИЛОЖЕНИЕ А**

# **(обязательное)**

# **Листинг кода**

Листинг 1 – Файл main.cpp

#include <Windows.h>

#include <CommCtrl.h>

#include <thread>

#include <vector>

#include <string>

#pragma comment(lib, "Comctl32.lib")

HWND g\_hTreeView;

const wchar\_t\* initialPath = L"D:\\\\test";

void PopulateTreeView(HWND hTreeView, const wchar\_t\* path, HTREEITEM hParentItem);

DWORD WINAPI WatchFileSystem(LPVOID lpParam);

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

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

{

// Регистрация класса окна

const wchar\_t CLASS\_NAME[] = L"MyFileExplorerClass";

WNDCLASS wc = {};

wc.lpfnWndProc = WindowProc;

wc.hInstance = hInstance;

wc.lpszClassName = CLASS\_NAME;

RegisterClass(&wc);

// Создание окна

HWND hwnd = CreateWindowEx(

0, // дополнительные стили окна

CLASS\_NAME, // имя класса окна

L"Мой файловый проводник", // заголовок окна

WS\_OVERLAPPEDWINDOW, // стиль окна

CW\_USEDEFAULT, CW\_USEDEFAULT, // позиция окна

500, 500, // размер окна

NULL, // родительское окно

NULL, // меню

hInstance, // дескриптор экземпляра

NULL // дополнительные данные

);

if (hwnd == NULL)

{

return 0;

}

ShowWindow(hwnd, nCmdShow);

// Цикл обработки сообщений

MSG msg = {};

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

{

TranslateMessage(&msg);

DispatchMessage(&msg);

}

UpdateWindow(hwnd);

return 0;

}

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

{

switch (uMsg)

{

case WM\_CREATE:

{

// Создание TreeView

g\_hTreeView = CreateWindowEx(0, WC\_TREEVIEW, L"", WS\_VISIBLE | WS\_CHILD | TVS\_HASBUTTONS | TVS\_LINESATROOT | TVS\_SHOWSELALWAYS,

0, 0, 400, 400, hwnd, NULL, NULL, NULL);

// Заполнение TreeView файловой системой

PopulateTreeView(g\_hTreeView, initialPath, TVI\_ROOT);

// Запуск потока отслеживания изменений в файловой системе

DWORD threadId;

HANDLE hThread = CreateThread(NULL, 0, WatchFileSystem, NULL, 0, &threadId);

if (hThread != NULL)

CloseHandle(hThread);

}

return 0;

case WM\_DESTROY:

PostQuitMessage(0);

return 0;

default:

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

}

}

// Функция заполнения TreeView

void PopulateTreeView(HWND hTreeView, const wchar\_t\* path, HTREEITEM hParentItem)

{

WIN32\_FIND\_DATA findData;

HANDLE hFind = INVALID\_HANDLE\_VALUE;

wchar\_t searchPath[MAX\_PATH];

wcscpy\_s(searchPath, path);

wcscat\_s(searchPath, L"\\\*");

hFind = FindFirstFile(searchPath, &findData);

if (hFind != INVALID\_HANDLE\_VALUE)

{

do

{

if (wcscmp(findData.cFileName, L".") != 0 && wcscmp(findData.cFileName, L"..") != 0)

{

if (findData.dwFileAttributes & FILE\_ATTRIBUTE\_DIRECTORY)

{

wchar\_t newPath[MAX\_PATH];

wcscpy\_s(newPath, path);

wcscat\_s(newPath, L"\\");

wcscat\_s(newPath, findData.cFileName);

TVINSERTSTRUCT insertStruct;

insertStruct.hParent = hParentItem;

insertStruct.hInsertAfter = TVI\_LAST;

insertStruct.item.mask = TVIF\_TEXT | TVIF\_IMAGE | TVIF\_SELECTEDIMAGE;

insertStruct.item.pszText = findData.cFileName;

insertStruct.item.iImage = 0;

insertStruct.item.iSelectedImage = 0;

HTREEITEM hNewItem = TreeView\_InsertItem(hTreeView, &insertStruct);

// Рекурсивно заполняем поддиректории

PopulateTreeView(hTreeView, newPath, hNewItem);

}

else

{

TVINSERTSTRUCT insertStruct;

insertStruct.hParent = hParentItem;

insertStruct.hInsertAfter = TVI\_LAST;

insertStruct.item.mask = TVIF\_TEXT | TVIF\_IMAGE | TVIF\_SELECTEDIMAGE;

insertStruct.item.pszText = findData.cFileName;

insertStruct.item.iImage = 1;

insertStruct.item.iSelectedImage = 1;

TreeView\_InsertItem(hTreeView, &insertStruct);

}

}

} while (FindNextFile(hFind, &findData));

FindClose(hFind);

}

}

// Функция отслеживания изменений в файловой системе

DWORD WINAPI WatchFileSystem(LPVOID lpParam)

{

std::wstring g\_rootPath(initialPath);

HANDLE hDirectory = CreateFile(g\_rootPath.c\_str(), FILE\_LIST\_DIRECTORY, FILE\_SHARE\_READ | FILE\_SHARE\_WRITE | FILE\_SHARE\_DELETE,

NULL, OPEN\_EXISTING, FILE\_FLAG\_BACKUP\_SEMANTICS, NULL);

if (hDirectory == INVALID\_HANDLE\_VALUE)

return 1;

DWORD bufferLength = 1024 \* 1024;

std::vector<BYTE> buffer(bufferLength);

while (true)

{

DWORD bytesReturned = 0;

BOOL result = ReadDirectoryChangesW(hDirectory, &buffer[0], bufferLength, TRUE, FILE\_NOTIFY\_CHANGE\_FILE\_NAME | FILE\_NOTIFY\_CHANGE\_DIR\_NAME,

&bytesReturned, NULL, NULL);

if (!result)

break;

// Очищаем TreeView перед перерисовкой

TreeView\_DeleteAllItems(g\_hTreeView);

// Перерисовываем TreeView

PopulateTreeView(g\_hTreeView, initialPath, TVI\_ROOT);

}

CloseHandle(hDirectory);

return 0;

}