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 № 1

на тему «Основы программирования в Win 32 API. Оконное приложение Win 32 с минимальной достаточной функциональностью. Обработка основных оконных сообщений»

Выполнил:

студент гр. 153504

Князев Н. Д.

Проверил:

Гриценко Н.Ю.

Минск 2023

СОДЕРЖАНИЕ

[Цель работы 3](#_Toc146680819)

[Краткие теоретические сведения 4](#_Toc146680820)

[Реализация программного продукта 6](#_Toc146680821)

[Результат выполнения программы 7](#_Toc146680822)

[Список использованных источников 9](#_Toc146680823)

[Приложение А 10](#_Toc146680824)

# 1 ЦЕЛЬ РАБОТЫ

Необходимо реализовать игру "Сапер" с графическим интерфейсом, позволяющим пользователю открывать ячейки поля и помечать мины. Для разработки нужно научиться использовать Win 32 API, создавать оконные приложения и обрабатывать основные оконные сообщения.

# 2 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Win32 API (Application Programming Interface) представляет собой набор функций и подпрограмм, созданных для взаимодействия с операционной системой Windows. Этот набор программных средств обеспечивает доступ к различным функциям и сервисам операционной системы Windows, что позволяет приложениям взаимодействовать с операционной системой и использовать ее возможности.

Окна играют важнейшую роль в разработке приложений. Они являются ключевыми элементами пользовательского интерфейса и обеспечивают взаимодействие приложения с пользователем. Win32 API предоставляет обширный набор функций и возможностей для создания, управления и настройки окон. Эти функции позволяют создавать главные окна приложений, диалоговые окна, элементы управления, такие как кнопки и текстовые поля, а также обрабатывать пользовательские события, такие как клики мыши и нажатия клавиш.

Функция wWinMain является точкой входа для Windows-приложений, написанных с использованием Win32 API. Она выполняет инициализацию приложения, создание окна и запуск цикла обработки сообщений.

Дескриптор окна (hWnd) — это указатель на структуру данных внутри операционной системы, которая содержит информацию о состоянии и характеристиках окна. Этот дескриптор обычно возвращается функцией CreateWindow при создании окна. Для выполнения операций с окном разработчики передают этот дескриптор в соответствующие функции.

Для создания окон используется функция CreateWindowW. Данная функция позволяет задавать параметры окна, такие как его размер, местоположение и стиль, и создавать различные элементы интерфейса, такие как кнопки и текстовые поля.

В wWinMain происходит цикл обработки сообщений. Он начинается с вызова функции GetMessage, которая ожидает появления новых сообщений в очереди сообщений Windows. Как только сообщение появляется, оно извлекается и передается функции TranslateMessage, которая выполняет необходимую предобработку сообщения, например, преобразование клавиш в символы. Затем сообщение передается функции DispatchMessage, которая направляет его на обработку в оконную процедуру, связанную с конкретным окном приложения. Здесь происходит фактическая обработка событий и выполнение соответствующих действий, например, обновление интерфейса, обработка пользовательского ввода и многое другое.

Функции оконных процедур, также известные как оконные процедуры или оконные обработчики, представляют собой специальные функции, которые связываются с каждым окном в приложении и отвечают за обработку сообщений и событий, связанных с этим окном.

Оконные процедуры получают на вход сообщения от операционной системы и обрабатывают их в соответствии с логикой приложения. Сообщения могут быть разнообразными: от событий пользовательского ввода, таких как нажатия клавиш и перемещения мыши, до системных сообщений, связанных с жизненным циклом окон и изменением их состояния.

Оконные процедуры обычно содержат инструкции для обновления графического интерфейса приложения, обработки ввода пользователя, валидации данных, реагирования на системные команды и многое другое. Каждое окно в приложении имеет свою собственную оконную процедуру.

WM-команды (Windows Messages) представляют собой набор стандартных сообщений, используемых в Windows-приложениях для обмена информацией и управления окнами и элементами интерфейса. Эти сообщения передаются между различными компонентами приложения, включая оконные процедуры (оконные обработчики) и систему Windows, и позволяют реализовать множество функциональных возможностей.

Примеры часто используемых WM-команд:

* WM\_CREATE: Это сообщение отправляется при создании окна и предоставляет возможность для настройки и инициализации оконных данных и ресурсов.
* WM\_DESTROY: Это сообщение отправляется перед уничтожением окна. Здесь происходит освобождение ресурсов и завершение работы с окном.
* WM\_COMMAND: Сообщение WM\_COMMAND генерируется в ответ на действия пользователя, такие как нажатие кнопок и выбор элементов меню. Оно позволяет приложению реагировать на команды пользователя.
* WM\_PAINT: Это сообщение генерируется, когда окно требует перерисовки, например, после изменения размера или при необходимости отображения новой информации.

# 3 РЕАЛИЗАЦИЯ ПРОГРАММНОГО ПРОДУКТА

Данный код представляет собой приложение для проигрывания аудиофайлов с использованием графического интерфейса на операционной системе Windows. Код разделен на несколько ключевых частей:

1. Определение структуры **AudioTrack**, которая представляет аудио-трек и хранит путь к файлу и его отображаемое имя.
2. Создание глобальных переменных и элементов управления оконного интерфейса, таких как кнопки воспроизведения, паузы, остановки, переключения на следующий и предыдущий треки, плейлист и кнопка добавления музыкальных файлов.
3. Функция **PlayTrack**, которая воспроизводит аудио-трек по указанному индексу в плейлисте.
4. Функция **AddTrackToPlaylist**, которая добавляет аудио-файл в плейлист, извлекая имя файла из его пути.
5. Функция **AddMusicFileFromDialog**, которая открывает диалоговое окно для выбора аудио-файла и добавляет его в плейлист.
6. Основная оконная процедура **WndProc**, в которой обрабатываются сообщения от пользовательского интерфейса, такие как нажатия кнопок и двойные клики на треках в плейлисте.
7. Функция **WinMain**, которая инициализирует оконное приложение, регистрирует класс окна, создает окно и запускает цикл обработки сообщений.

Общий результат - это приложение, позволяющее управлять воспроизведением аудио-треков, создавать плейлист и добавлять новые аудио-файлы для воспроизведения."

**4 РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ ПРОГРАММЫ**

В выполненном приложении имеется структура кнопок, таких как Add Music File, Play, Pause, Stop, Next и Prev. Кнопки отвечают за проигрывание аудио, остановку аудио, и переключение аудио, также, можно добавлять различные аудио форматов .wav и .mp3 в плейлист (рисунок 4.1).

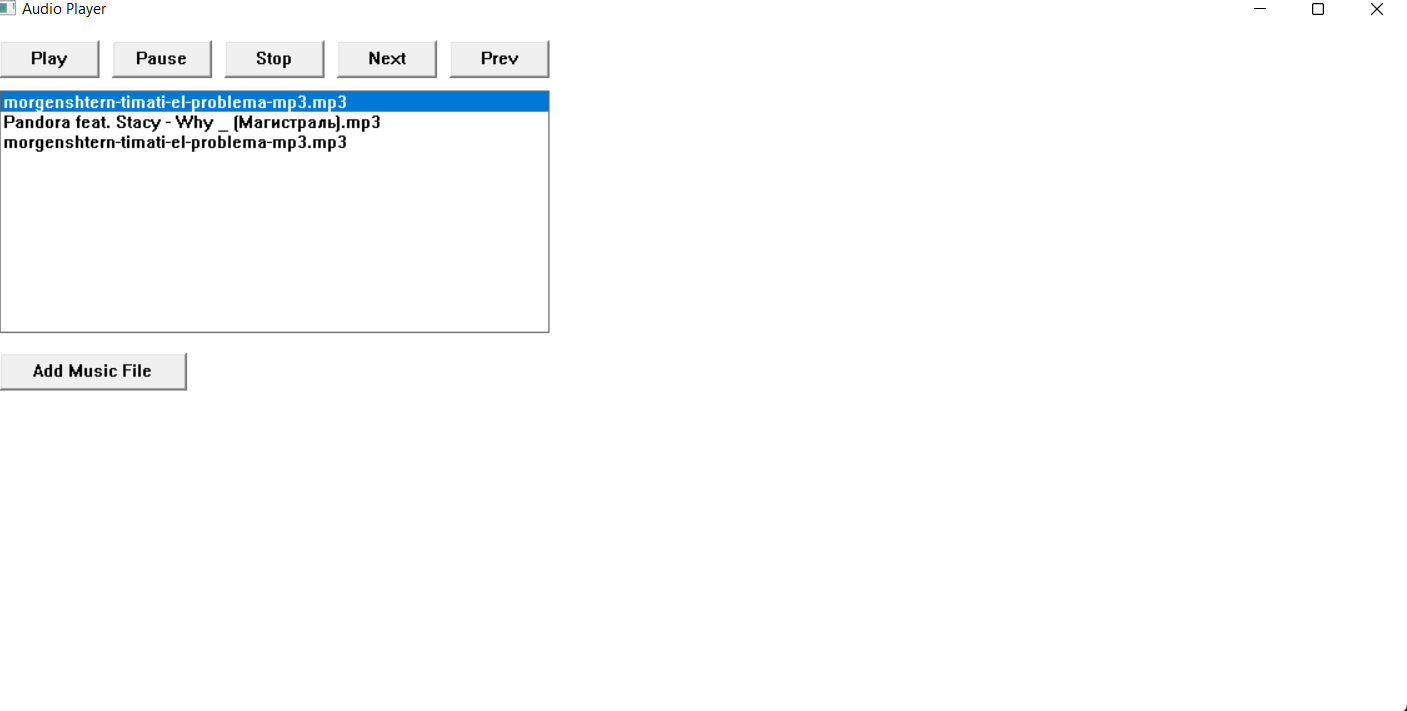


Рисунок 4.1 – Главное окно приложения

При нажатии на кнопку Add Music File появляется окно проводника, (рисунок 4.2).

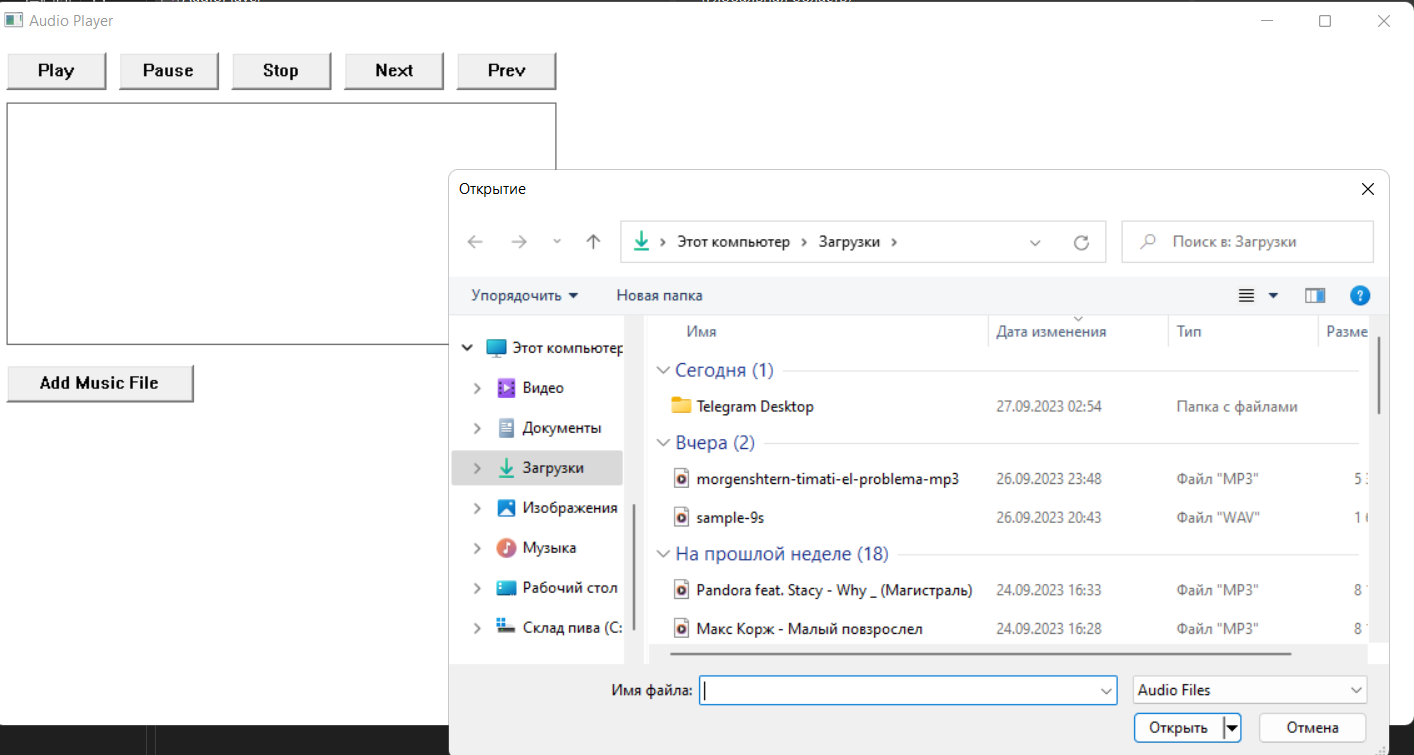


Рисунок 4.2 – Текстовое сообщение, информирующее о выигрыше

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Get Started with Win32 and C++ [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/ru-ru/windows/win32/LearnWin32/learn-to-program-for-windows>.

[2] Окна на чистом WinAPI. Или просто о сложном [Электронный ресурс]. – Режим доступа: <https://habr.com/ru/articles/352096/>.

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

# ПРИЛОЖЕНИЕ А

**(обязательное)**

**Исходный код программы**

Листинг 1 – Файл Lab1.h:

#pragma once

#include "resource.h"

Листинг 2 – Файл AudioPlayer.cpp:

#include <Windows.h>

#include <mmsystem.h>

#include <vector>

#include <string>

#include <tchar.h>

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

HWND hwnd;

HINSTANCE hInst;

static TCHAR szWindowClass[] = \_T("AudioPlayer");

static TCHAR szTitle[] = \_T("Audio Player");

struct AudioTrack {

std::wstring filePath;

std::wstring displayName;

};

std::vector<AudioTrack> playlist;

int currentTrackIndex = 0;

HWND hPlayButton;

HWND hPauseButton;

HWND hStopButton;

HWND hNextButton;

HWND hPrevButton;

HWND hPlaylist;

HWND hAddMusicButton;

void PlayTrack(int trackIndex) {

if (trackIndex >= 0 && trackIndex < playlist.size()) {

currentTrackIndex = trackIndex;

mciSendString(L"close all", NULL, 0, NULL);

std::wstring command = L"open \"" + playlist[currentTrackIndex].filePath + L"\" type mpegvideo alias MyAudio";

mciSendString(command.c\_str(), NULL, 0, NULL);

mciSendString(L"play MyAudio", NULL, 0, NULL);

SendMessage(hPlaylist, LB\_SETCURSEL, (WPARAM)currentTrackIndex, 0);

}

}

void AddTrackToPlaylist(const std::wstring& filePath) {

AudioTrack track;

track.filePath = filePath;

size\_t lastSlash = filePath.find\_last\_of(L"\\");

if (lastSlash != std::wstring::npos) {

track.displayName = filePath.substr(lastSlash + 1);

}

else {

track.displayName = filePath;

}

playlist.push\_back(track);

SendMessage(hPlaylist, LB\_ADDSTRING, 0, (LPARAM)track.displayName.c\_str());

}

void AddMusicFileFromDialog(HWND hWnd) {

OPENFILENAME ofn;

wchar\_t szFile[MAX\_PATH] = { 0 };

ZeroMemory(&ofn, sizeof(ofn));

ofn.lStructSize = sizeof(ofn);

ofn.lpstrFile = szFile;

ofn.nMaxFile = sizeof(szFile);

ofn.lpstrFilter = L"Audio Files\0\*.mp3;\*.wav;\*.ogg\0All Files\0\*.\*\0";

ofn.nFilterIndex = 1;

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

if (GetOpenFileName(&ofn)) {

AddTrackToPlaylist(szFile);

}

}

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

switch (message) {

case WM\_CREATE:

hPlayButton = CreateWindow(\_T("BUTTON"), \_T("Play"), WS\_CHILD | WS\_VISIBLE | BS\_PUSHBUTTON, 10, 10, 80, 30, hWnd, (HMENU)1, hInst, NULL);

hPauseButton = CreateWindow(\_T("BUTTON"), \_T("Pause"), WS\_CHILD | WS\_VISIBLE | BS\_PUSHBUTTON, 100, 10, 80, 30, hWnd, (HMENU)2, hInst, NULL);

hStopButton = CreateWindow(\_T("BUTTON"), \_T("Stop"), WS\_CHILD | WS\_VISIBLE | BS\_PUSHBUTTON, 190, 10, 80, 30, hWnd, (HMENU)3, hInst, NULL);

hNextButton = CreateWindow(\_T("BUTTON"), \_T("Next"), WS\_CHILD | WS\_VISIBLE | BS\_PUSHBUTTON, 280, 10, 80, 30, hWnd, (HMENU)4, hInst, NULL);

hPrevButton = CreateWindow(\_T("BUTTON"), \_T("Prev"), WS\_CHILD | WS\_VISIBLE | BS\_PUSHBUTTON, 370, 10, 80, 30, hWnd, (HMENU)5, hInst, NULL);

hPlaylist = CreateWindow(\_T("LISTBOX"), NULL, WS\_CHILD | WS\_VISIBLE | WS\_BORDER | LBS\_NOTIFY | LBS\_HASSTRINGS, 10, 50, 440, 200, hWnd, (HMENU)6, hInst, NULL);

hAddMusicButton = CreateWindow(\_T("BUTTON"), \_T("Add Music File"), WS\_CHILD | WS\_VISIBLE | BS\_PUSHBUTTON, 10, 260, 150, 30, hWnd, (HMENU)7, hInst, NULL);

break;

case WM\_COMMAND:

switch (LOWORD(wParam)) {

case 1:

PlayTrack(currentTrackIndex);

break;

case 2:

mciSendString(L"pause MyAudio", NULL, 0, NULL);

break;

case 3:

mciSendString(L"stop MyAudio", NULL, 0, NULL);

break;

case 4:

PlayTrack(currentTrackIndex + 1);

break;

case 5:

PlayTrack(currentTrackIndex - 1);

break;

case 6:

if (HIWORD(wParam) == LBN\_DBLCLK) {

int selectedTrackIndex = SendMessage(hPlaylist, LB\_GETCURSEL, 0, 0);

if (selectedTrackIndex != LB\_ERR) {

PlayTrack(selectedTrackIndex);

}

}

break;

case 7:

AddMusicFileFromDialog(hWnd);

break;

default:

break;

}

break;

case WM\_PAINT: {

PAINTSTRUCT ps;

HDC hdc = BeginPaint(hWnd, &ps);

EndPaint(hWnd, &ps);

}

break;

case WM\_DESTROY:

mciSendString(L"close all", NULL, 0, NULL);

PostQuitMessage(0);

break;

default:

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

break;

}

return 0;

}

int APIENTRY WinMain(HINSTANCE hInstance,

HINSTANCE hPrevInstance,

LPSTR lpCmdLine,

int nCmdShow) {

WNDCLASSEX 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, IDI\_APPLICATION);

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

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

wcex.lpszMenuName = NULL;

wcex.lpszClassName = szWindowClass;

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

if (!RegisterClassEx(&wcex)) {

MessageBox(NULL, \_T("Call to RegisterClassEx failed!"), \_T("Audio Player"), 0);

return 1;

}

hInst = hInstance;

hwnd = CreateWindow(szWindowClass, szTitle, WS\_OVERLAPPEDWINDOW,

CW\_USEDEFAULT, 0, CW\_USEDEFAULT, 0, NULL, NULL, hInstance, NULL);

if (!hwnd) {

MessageBox(NULL, \_T("Call to CreateWindow failed!"), \_T("Audio Player"), 0);

return 1;

}

ShowWindow(hwnd, nCmdShow);

UpdateWindow(hwnd);

MSG msg;

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

TranslateMessage(&msg);

DispatchMessage(&msg);

}

return (int)msg.wParam;

}