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 №2

на тему

**РАСШИРЕННОЕ ИСПОЛЬЗОВАНИЕ ОКОННОГО ИНТЕРФЕЙСА WIN32 И GDI**

Студент Е. С. Кахновский

Преподаватель Н. Ю. Гриценко

Минск 2023

СОДЕРЖАНИЕ

[1 Цель работы 3](#_Toc146752068)

[2 Теоретические сведения 4](#_Toc146752069)

[3 Результат выполнения 6](#_Toc146752070)

[Заключение 8](#_Toc146752071)

[Список использованных источников 9](#_Toc146752072)

[Приложение А (обязательное) Листинг кода 10](#_Toc146752073)

1. **ЦЕЛЬ РАБОТЫ**

Целью выполнения лабораторной работы является расширенное использование оконного интерфейса Win32 и GDI, формирование сложных изображений, создание и использование элементов управления, обработка различных сообщений, использование механизма перехвата сообщений (winhook). Чтобы продемонстрировать все условия, будет создано оконное приложение «Airplane», позволяющее управлять моделью самолета, изображенного с помощью функций GDI.

1. **ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

Win32 API (Windows API) - это набор функций и интерфейсов прикладного программирования, предоставляемых операционной системой Microsoft Windows для разработки приложений под эту операционную систему. Win32 API предоставляет доступ к широкому спектру функциональности операционной системы, включая создание окон, работу с файлами и директориями, многозадачность, ввод и вывод данных, сетевое взаимодействие и многое другое.

Разработчики используют Win32 API на языках программирования C и C++ для создания приложений под операционной системой Windows.

GDI (Graphics Device Interface) - это компонент операционной системы Windows, который предоставляет API для рисования 2D графики и взаимодействия с графическими устройствами, такими как мониторы, принтеры и другие выводящие устройства. GDI обеспечивает базовую функциональность для работы с графическими ресурсами, такими как кисти, перья, шрифты и регионы, и предоставляет множество функций для создания и манипулирования изображениями.

Для выполнения данной лабораторной работы, были использованы следующие теоретические сведения и концепции:

– WndProc - это процедура обработки сообщений для окна приложения. Она обрабатывает сообщения, поступающие от операционной системы, такие как WM\_PAINT, WM\_KEYDOWN, WM\_TIMER и другие.

– WNDCLASSEX: это структура, используемая для описания класса окна. Включает в себя настройки, такие как стиль, функция обратного вызова (WndProc), иконки, курсоры и т.д. Зарегистрированный класс используется для создания окон.

– CreateWindow: функция, используемая для создания окна или элемента управления в окне. Принимает параметры, такие как класс окна, заголовок, стиль окна и позицию.

– Создание окна: Сначала регистрируется класс окна с помощью RegisterClassEx, который описывает параметры окна. Затем с помощью CreateWindow создается главное окно приложения.

– Обработка сообщений: В главном цикле сообщений (WinMain) приложение ожидает и обрабатывает сообщения с помощью GetMessage, TranslateMessage и DispatchMessage. WndProc обрабатывает различные типы сообщений, такие как рисование, клавиатурные события и таймеры.

– Рисование: Рисование выполняется в процедуре WM\_PAINT. Окно разделено на несколько частей с разными цветами, представляющими землю, фон и т.д. Функция DrawAirplane отрисовывает "самолет" с использованием различных элементов: фюзеляж, хвост, крыло, окна и шасси (в зависимости от состояния шасси).

– Управление: Пользователь может управлять "самолетом" с помощью клавиатуры. Стрелки влево и вправо увеличивают/уменьшают скорость "самолета", а стрелки вверх и вниз изменяют его вертикальное положение. Пробел переключает шасси "самолета" между выдвинутым и убранным состоянием.

– Хук клавиатуры: Приложение также демонстрирует использование глобального хука клавиатуры с помощью функций SetWindowsHookEx и UnhookWindowsHookEx. Хук отслеживает нажатие клавиши пробела и выводит сообщение в отладочное окно о состоянии шасси. Таймер: Приложение использует таймер для периодического обновления положения "самолета" в окне и его отрисовки.

– Обработка ошибок: В коде предусмотрена обработка ошибок с использованием сообщений MessageBox, чтобы уведомлять пользователя о проблемах.

1. **РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ**

В ходе выполнения лабораторной работы было создано оконное приложение, позволяющее управлять модель самолета (взаимодействие происходит по нажатию стрелок на клавиатуре, пробел – открыть/закрыть шасси)(рисунок 1).

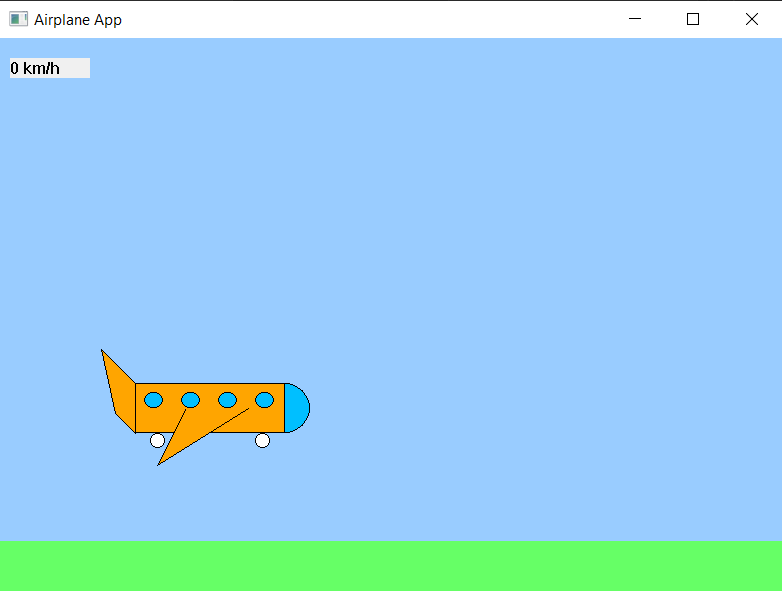


Рисунок 1 ─ Основное окно приложения

Так же присутствует обработка ошибок, таких как, приземление на землю без выпущенных шасси (рисунок 2).

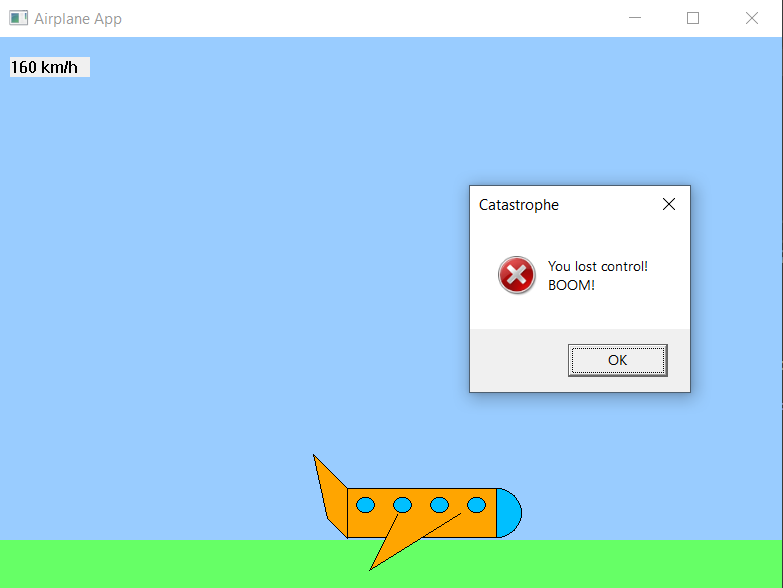


Рисунок 2 ─ Окно потери контроля над самолетом

# ЗАКЛЮЧЕНИЕ

В ходе выполнения лабораторной работы было разработано оконное приложение, представляющее различные возможности управления моделью самолета, на WIN32 API с использованем расширенного оконного интерфейса Win32 и GDI, формирование сложных изображений, создание и использование элементов управления, обработка различных сообщений, использование механизма перехвата сообщений (winhook).

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Щупак Ю. Win32 API. Разработка приложений для Windows. ─ СПб: Питер, 2008. ─ 592 с.: ил.
2. Создание классических приложений для Windows с использованием API Win32 [Электронный ресурс]. ─ Режим доступа: <https://learn.microsoft.com/ru-ru/windows/win32/api>.
3. Графический интерфейс GDI в Microsoft Windows ─ Александр Фролов, Григорий Фролов Том 4, М.: Диалог-МИФИ, 1993, 288 стр.

# ПРИЛОЖЕНИЕ А (обязательное) Листинг кода

Листинг 1 ­– Файл Airplane.cpp

#include <windows.h>

#include <corecrt\_wstdio.h>

#include <string>

#include "AirplaneDef.h"

HINSTANCE hInst;

HWND hMainWnd;

// Variables

HHOOK landingGear\_hKeyboardHook = NULL;

bool isLandingGear = true;

bool isAirplaneMoving = true;

bool isCrashed = false;

// Start position

int airplaneX = 70;

int airplaneY = 510;

// Start speed

int airplaneSpeed = 0;

// Func

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

void DrawAirplane(HDC hdc, int x, int y, int width, int height);

void UpdateAirplanePosition(int deltaX, int deltaY);

void SwitchLandingGear();

void StartAirplaneMovement();

void StopAirplaneMovement();

// Hook func

LRESULT CALLBACK KeyboardHookProc(int nCode, WPARAM wParam, LPARAM lParam);

void SetKeyboardHook();

void UnhookKeyboardHook();

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

hInst = hInstance;

WNDCLASSEX wcex = {

sizeof(WNDCLASSEX),

CS\_HREDRAW | CS\_VREDRAW,

WndProc,

0L,

0L,

GetModuleHandle(NULL),

NULL, NULL, (HBRUSH)(COLOR\_WINDOW + 1),

NULL, L"AirplaneApp", NULL

};

if (!RegisterClassEx(&wcex))

{

MessageBox(NULL, L"Call to RegisterClassEx failed!", L"Error", MB\_ICONERROR);

return 1;

}

hMainWnd = CreateWindow(

L"AirplaneApp",

L"Airplane App",

WS\_OVERLAPPEDWINDOW, CW\_USEDEFAULT, CW\_USEDEFAULT,

800, 600, NULL, NULL, hInstance, NULL);

// Creating a label for a airplaneSpeed

CreateWindow(L"STATIC", L"0 km/h", WS\_CHILD | WS\_VISIBLE, 10, 20, 80, 20, hMainWnd, (HMENU)ID\_SPEED\_LABEL, hInstance, NULL);

if (!hMainWnd)

{

MessageBox(NULL, L"Call to CreateWindow failed!", L"Error", MB\_ICONERROR);

return 1;

}

ShowWindow(hMainWnd, nCmdShow);

UpdateWindow(hMainWnd);

// Setting the hook

SetKeyboardHook();

// Setting the timer

SetTimer(hMainWnd, ID\_MAIN\_TIMER, 20, NULL);

MSG msg;

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

TranslateMessage(&msg);

DispatchMessage(&msg);

}

// Delete the hook

UnhookKeyboardHook();

return (int)msg.wParam;

}

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

switch (message){

case WM\_PAINT:

{

PAINTSTRUCT ps;

HDC hdc = BeginPaint(hWnd, &ps);

RECT rc;

GetClientRect(hWnd, &rc);

// Setting the background color to light blue (RGB(153, 204, 255))

HBRUSH hLightBlueBrush = CreateSolidBrush(RGB(153, 204, 255));

FillRect(hdc, &rc, hLightBlueBrush);

DeleteObject(hLightBlueBrush);

// Rendering the ground in light green colot (RGB(102, 255, 102))

HBRUSH hLightGreenBrush = CreateSolidBrush(RGB(102, 255, 102));

RECT groundRect = { rc.left, rc.bottom - 50, rc.right, rc.bottom };

FillRect(hdc, &groundRect, hLightGreenBrush);

DeleteObject(hLightGreenBrush);

// Main Background the same as the system

//FillRect(hdc, &rc, (HBRUSH)(COLOR\_WINDOW + 1));

DrawAirplane(hdc, airplaneX, airplaneY, AIRPLANE\_WIDTH, AIRPLANE\_HEIGHT);

EndPaint(hWnd, &ps);

}

break;

case WM\_KEYDOWN:

switch (wParam) {

case VK\_SPACE:

SwitchLandingGear();

InvalidateRect(hWnd, NULL, TRUE);

break;

case VK\_LEFT:

if (airplaneSpeed > 0)

airplaneSpeed--;

// Stop airplane movement

else StopAirplaneMovement();

break;

case VK\_RIGHT:

if (airplaneSpeed < MAX\_SPEED)

airplaneSpeed++;

// Start airplane movement

StartAirplaneMovement();

break;

case VK\_UP:

UpdateAirplanePosition(0, -airplaneSpeed);

InvalidateRect(hWnd, NULL, TRUE);

break;

case VK\_DOWN:

UpdateAirplanePosition(0, airplaneSpeed);

InvalidateRect(hWnd, NULL, TRUE);

break;

}

// Updating text in ID\_SPEED\_LABEL

{

HWND hSpeedLabel = GetDlgItem(hWnd, ID\_SPEED\_LABEL);

if (hSpeedLabel != NULL) {

WCHAR speedText[16];

swprintf\_s(speedText, L"%d km/h", airplaneSpeed \* 80);

SetWindowTextW(hSpeedLabel, speedText);

}

}

break;

case WM\_TIMER:

if (wParam == ID\_MAIN\_TIMER && isAirplaneMoving) {

UpdateAirplanePosition(airplaneSpeed, 0);

InvalidateRect(hWnd, NULL, TRUE);

}

break;

case WM\_DESTROY:

KillTimer(hWnd, ID\_MAIN\_TIMER);

PostQuitMessage(0);

break;

default:

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

}

return 0;

}

void DrawAirplane(HDC hdc, int x, int y, int width, int height) {

// Coordinates for body

int left = x;

int right = x + width;

int top = y;

int bottom = y + height;

// Drawing an airplane body

{

HBRUSH hBrush = CreateSolidBrush(RGB(255, 165, 0));

HBRUSH hOldBrush = (HBRUSH)SelectObject(hdc, hBrush);

Rectangle(hdc, left, top, right, bottom);

SelectObject(hdc, hOldBrush);

DeleteObject(hBrush);

}

// Drawing an airplane cabin

{

HBRUSH hBrush = CreateSolidBrush(RGB(0, 191, 255));

HBRUSH hOldBrush = (HBRUSH)SelectObject(hdc, hBrush);

// Drawing a semicircle (right side)

int radius = height / 2;

int centerX = x + width;

int centerY = y + (height / 2);

//Arc(hdc, right - radius, top, right + radius, bottom, right, bottom, right, top);

Chord(hdc, right - radius, top, right + radius, bottom, right, bottom, right, top);

//Pie(hdc, right - radius, top, right + radius, bottom, right, bottom, right, top);

SelectObject(hdc, hOldBrush);

DeleteObject(hBrush);

}

// Drawing an airplane tail

{

HBRUSH hBrush = CreateSolidBrush(RGB(255, 165, 0));

HBRUSH hOldBrush = (HBRUSH)SelectObject(hdc, hBrush);

int offsetT1 = 20;

int offsetT2 = offsetT1 \* 1.7;

// Creating an array of tail points

POINT points[] = {

{left, bottom},

{left - offsetT1, bottom - offsetT1},

{left - offsetT2, top - offsetT2},

{left, top},

{left, bottom}

};

// Creating a contour for painting

HRGN hRegion = CreatePolygonRgn(points, sizeof(points) / sizeof(points[0]), WINDING);

// Fill in the region with color

FillRgn(hdc, hRegion, hBrush);

Polyline(hdc, points, sizeof(points) / sizeof(points[0]));

// Delete the created region and brush

DeleteObject(hRegion);

SelectObject(hdc, hOldBrush);

DeleteObject(hBrush);

}

// Drawing an airplane wing

{

HBRUSH hBrush = CreateSolidBrush(RGB(255, 165, 0));

HBRUSH hOldBrush = (HBRUSH)SelectObject(hdc, hBrush);

int offsetH = width / 4;

int offsetV = height / 2;

// Creating an array of wing points

POINT points[] = {

{right - offsetH, top + offsetV},

{left + offsetH \* 0.6, bottom + offsetV \* 1.3},

{left + offsetH \* 1.4, top + offsetV},

};

// Creating a contour for painting

HRGN hRegion = CreatePolygonRgn(points, sizeof(points) / sizeof(points[0]), WINDING);

// Fill in the region with color

FillRgn(hdc, hRegion, hBrush);

Polyline(hdc, points, sizeof(points) / sizeof(points[0]));

// Delete the created region and brush

DeleteObject(hRegion);

SelectObject(hdc, hOldBrush);

DeleteObject(hBrush);

}

// Drawing an airplane windows

{

HBRUSH hBrush = CreateSolidBrush(RGB(0, 191, 255));

HBRUSH hOldBrush = (HBRUSH)SelectObject(hdc, hBrush);

int winCount = 4;

int offsetH = width / winCount;

int offsetV = height / 2;

int offset = offsetH / 4;

for (size\_t i = 0; i < winCount; i++)

{

Ellipse(hdc, left + i \* offsetH + offset, top + offset, left + (i + 1) \* offsetH - offset, top + offsetV);

}

SelectObject(hdc, hOldBrush);

DeleteObject(hBrush);

}

if (isLandingGear) {

int offsetX = width / 10;

int offsetY = width / 10;

Ellipse(hdc, left + offsetX, bottom, left + offsetX \* 2, bottom + offsetY);

Ellipse(hdc, right - offsetX \* 2, bottom, right - offsetX, bottom + offsetY);

}

}

void SwitchLandingGear() {

isLandingGear = !isLandingGear;

}

void UpdateAirplanePosition(int deltaX, int deltaY) {

airplaneX += deltaX;

airplaneY += deltaY;

RECT clientRect;

GetClientRect(hMainWnd, &clientRect);

// Horizontal

if (airplaneX + 100 <= 0) {

airplaneX = clientRect.right;

}

else if (airplaneX >= clientRect.right) {

airplaneX = -100;

}

// Vertical

if (airplaneY <= 0) {

airplaneY = clientRect.top;

}

else if (airplaneY >= clientRect.bottom - (50 + AIRPLANE\_HEIGHT)) {

airplaneY = clientRect.bottom - (50 + AIRPLANE\_HEIGHT);

// Collision with the ground

if(!isLandingGear && !isCrashed)

{

StopAirplaneMovement();

isCrashed = true;

MessageBox(hMainWnd, L"You lost control!\nBOOM!", L"Catastrophe", MB\_ICONERROR);

SendMessage(hMainWnd, WM\_DESTROY, 0, 0);

}

}

}

void StartAirplaneMovement() {

isAirplaneMoving = true;

}

void StopAirplaneMovement() {

isAirplaneMoving = false;

}

// Function for processing the left hook

LRESULT CALLBACK KeyboardHookProc(int nCode, WPARAM wParam, LPARAM lParam) {

if (nCode >= 0) {

if (wParam == WM\_KEYDOWN) {

KBDLLHOOKSTRUCT\* pKeyInfo = (KBDLLHOOKSTRUCT\*)lParam;

if (pKeyInfo->vkCode == VK\_SPACE) {

// Depending on the state of the landing gear, we output a message to Debug

// using ! because hook before WM

if (!isLandingGear) {

OutputDebugString(L"Landing Gear Deployed\n");

}

else {

OutputDebugString(L"Landing Gear Retracted\n");

}

//// Преобразуйте переменную time в более удобочитаемый формат

//int timeInSeconds = pKeyInfo->time / 1000; // Преобразуем миллисекунды в секунды

//int days = timeInSeconds / (60 \* 60 \* 24);

//int hours = (timeInSeconds / (60 \* 60)) % 24;

//int minutes = (timeInSeconds / 60) % 60;

//int seconds = timeInSeconds % 60;

//std::wstring time = L"dd hh/mm/ss: "

// + std::to\_wstring(days) + L" "

// + std::to\_wstring(hours) + L"/"

// + std::to\_wstring(minutes) + L"/"

// + std::to\_wstring(seconds) + L"\n";

//// Выводите переменную wstrTime в отладочную консоль

//OutputDebugString(time.c\_str());

//// Преобразуйте переменную time в широкий формат (wstring)

//std::wstring wstrTime = std::to\_wstring(pKeyInfo->time);

//// Выводите переменную wstrTime в отладочную консоль

//OutputDebugString(wstrTime.c\_str());

}

}

}

return CallNextHookEx(landingGear\_hKeyboardHook, nCode, wParam, lParam);

}

// Function for setting the left hook

void SetKeyboardHook() {

landingGear\_hKeyboardHook = SetWindowsHookEx(WH\_KEYBOARD\_LL, KeyboardHookProc, NULL, 0);

if (landingGear\_hKeyboardHook == NULL) {

MessageBox(NULL, L"Failed to set keyboard hook", L"Error", MB\_ICONERROR);

}

}

// Function for removing the left hook

void UnhookKeyboardHook() {

if (landingGear\_hKeyboardHook != NULL) {

UnhookWindowsHookEx(landingGear\_hKeyboardHook);

landingGear\_hKeyboardHook = NULL;

}

}

Листинг 2 ­– Файл AirplaneDef.h

#pragma once

// Identifiers of interface elements

#define ID\_MAIN\_TIMER 1

#define ID\_SPEED\_LABEL 2

// Constants

const int MAX\_SPEED = 15;

const int AIRPLANE\_WIDTH = 150;

const int AIRPLANE\_HEIGHT = 50;