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 №6

на тему

**СРЕДСТВА СИНХРОНИЗАЦИИ И ВЗАИМНОГО ИСКЛЮЧЕНИЯ (WINDOWS).**

**ИЗУЧЕНИЕ И ИСПОЛЬЗОВАНИЕ СРЕДСТВ СИНХРОНИЗАЦИИ И ВЗАИМНОГО ИСКЛЮЧЕНИЯ**

Студент Е. С. Кахновский

Преподаватель Н. Ю. Гриценко

Минск 2023

СОДЕРЖАНИЕ

[1 Цель работы 3](#_Toc146752068)

[2 Теоретические сведения 4](#_Toc146752069)

[3 Результат выполнения 5](#_Toc146752070)

[Заключение 7](#_Toc146752071)

[Список использованных источников 8](#_Toc146752072)

[Приложение А (обязательное) Листинг кода 9](#_Toc146752073)

1. **ЦЕЛЬ РАБОТЫ**

Целью данной лабораторной работы является изучение   
средств синхронизации и взаимного исключения (*Windows*). Использование данных средств будет продемонстрировано на оконном приложении «*Airplane*», позволяющее управлять моделью самолета, изображенного с помощью функций *GDI*.

1. **ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

Средства синхронизации и взаимного исключения в *Windows* играют важную роль при работе с многозадачными приложениями и многопоточными программами. Они предназначены для управления доступом к общим ресурсам и предотвращения гонок данных. Вот краткие теоретические сведения о некоторых из них [1]:

1. Мьютексы (*Mutexes*): Мьютексы используются для ограничения доступа к общим ресурсам одновременно только одним потоком. Они поддерживают две операции: захват (*lock*) и освобождение (*unlock*). При попытке захвата уже занятого мьютекса, поток будет заблокирован до освобождения мьютекса другим потоком.
2. Семафоры (*Semaphores*): Семафоры представляют счетчик, который используется для ограничения количества потоков, имеющих доступ к ресурсам. Они могут поддерживать более одного потока внутри критической секции. Семафоры могут быть исходно установлены на определенное значение и уменьшаться при входе в критическую секцию и увеличиваться при выходе.
3. Критические секции (*Critical* *Sections*): Критические секции представляют собой блоки кода, в которых только один поток может выполняться одновременно. Они создаются с помощью функций, таких как *InitializeCriticalSection*, *EnterCriticalSection* и *LeaveCriticalSection*. Критические секции обычно более эффективны, чем мьютексы и семафоры, но они могут использоваться только в пределах одного процесса.
4. События (*Events*): События позволяют потокам ожидать определенных условий или событий и продолжать выполнение, когда эти условия выполняются. События могут быть автоматически сброшенными (как только один поток ожидает, они автоматически сбрасываются) или ручными (остаются установленными до явного сброса). События часто используются для сигнализации между потоками.
5. Интерлокированные операции (*Interlocked* *Operations*): *WIN* *API* предоставляет функции, такие как *InterlockedIncrement*, *InterlockedDecrement*, и другие, для безопасного выполнения операций над разделяемыми переменными. Эти операции гарантируют атомарное выполнение и предотвращают гонки данных.
6. **РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ**

Основой является приложение «*Airplane*» (рисунок 1), позволяющее управлять самолетом по нажатию клавиш на клавиатуре.

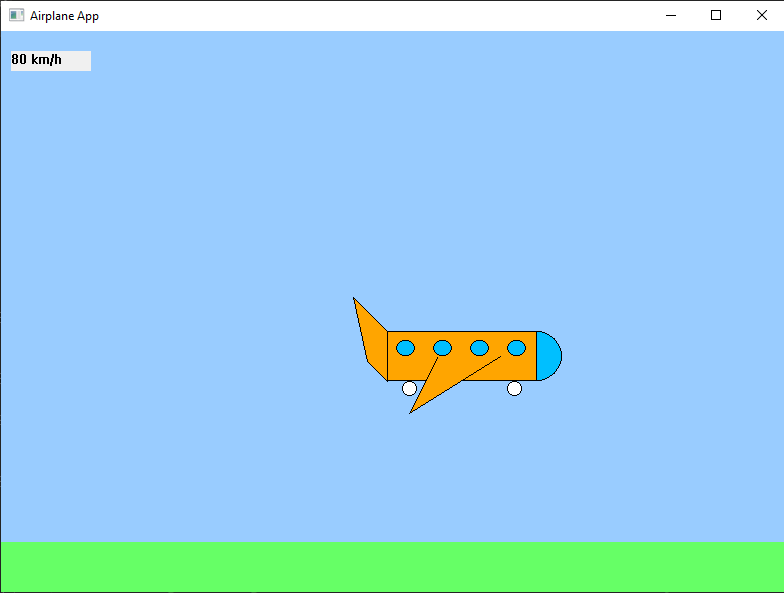


Рисунок 1 – Окно приложения

В ходе выполнения задания был реализован функционал сохранения координат и скорости самолета, после выхода из приложения, в реестр *Windows* (рисунок 2), а также сохранение логов в текстовый файл *flight\_recorder.txt* с помощью *mapping* (рисунок 3).

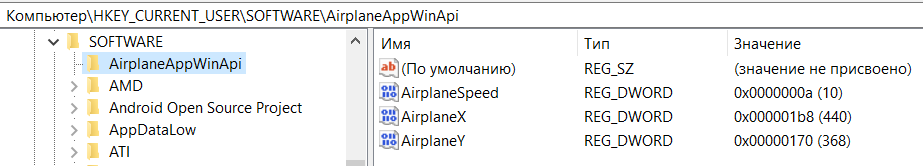


Рисунок 2– Отчет записи в реестр

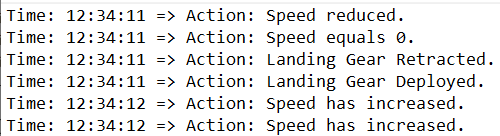


Рисунок 3– Отчет записи в *flight\_recorder.txt*

Для обеспечения синхронизации потоков записи в реестр и текстовый файл, были использованы средства взаимного исключения мьютексы (*mutex*) (рисунок 4).

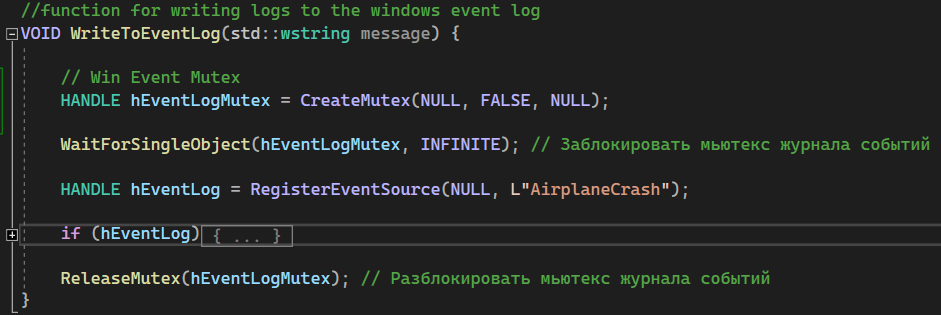


Рисунок 4– Использование мьютекса журнала событий

# ЗАКЛЮЧЕНИЕ

В результате лабораторной работы были изучены  
средства синхронизации и взаимного исключения (*Windows*). Использование данных средств было продемонстрировано на оконном приложении «*Airplane*».

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Основы программирования для Win32 API [Электронный ресурс]. – Режим доступа: <https://dims.karelia.ru/win32/>.
2. Сопоставление файлов [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/ru-ru/windows/win32/memory/file-mapping>.
3. Ведение журнала событий [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/eventlog/event-logging/.

# ПРИЛОЖЕНИЕ А (обязательное) Листинг кода

Листинг 1 – Файл *Airplane.cpp*

#include <windows.h>

#include <corecrt\_wstdio.h>

#include <string>

#include <fstream>

#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;

// Memory mapped file

HANDLE hFile = NULL;

HANDLE hMapFile = NULL;

LPVOID pMappedData = NULL;

SIZE\_T mappedDataSize = 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();

// File & Thread func

// Function for recording keystrokes to a file in separate thread

DWORD WINAPI RecordKeyPressThread(LPVOID lpParam);

VOID CallRecordKeyPressInThread(CONST CHAR\* actionString);

VOID InitializeMappingFile();

VOID UninitializeMappingFile();

// Hook func

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

VOID SetKeyboardHook();

VOID UnhookKeyboardHook();

// Registry func

VOID SaveCoordinatesToRegistry();

VOID LoadCoordinatesFromRegistry();

// EventLog func

VOID WriteToEventLog(std::wstring message);

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

InitializeMappingFile();

// При запуске приложения, загрузите координаты из реестра

LoadCoordinatesFromRegistry();

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);

}

// Перед закрытием приложения, сохраните текущие координаты в реестре

SaveCoordinatesToRegistry();

// Delete the hook

UnhookKeyboardHook();

UninitializeMappingFile();

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;

airplaneSpeed = 0;

CallRecordKeyPressInThread("Airplane crashed. WARNING!");

WriteToEventLog(L"Airplane crashed. WARNING!");

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;

switch (pKeyInfo->vkCode) {

case 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");

CallRecordKeyPressInThread("Landing Gear Deployed.");

}

else {

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

CallRecordKeyPressInThread("Landing Gear Retracted.");

}

break;

case VK\_LEFT:

if (airplaneSpeed > 0)

CallRecordKeyPressInThread("Speed reduced.");

else CallRecordKeyPressInThread("Speed equals 0.");

break;

case VK\_RIGHT:

if (airplaneSpeed < MAX\_SPEED)

CallRecordKeyPressInThread("Speed has increased.");

else CallRecordKeyPressInThread("Speed equals MAX\_SPEED.");

break;

case VK\_UP:

CallRecordKeyPressInThread("Airplane climb.");

break;

case VK\_DOWN:

CallRecordKeyPressInThread("Airplane landing.");

break;

}

}

}

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;

}

}

DWORD WINAPI RecordKeyPressThread(LPVOID lpParam) {

std::string actionString = (CHAR\*)(lpParam);

if (pMappedData != NULL) {

// Getting current time

SYSTEMTIME currentTime;

GetLocalTime(&currentTime);

// Forming a strings with information about the time and the key

std::string time = "Time: " + std::to\_string(currentTime.wHour) + ":" +

std::to\_string(currentTime.wMinute) + ":" + std::to\_string(currentTime.wSecond);

std::string keyInfo = time + " => Action: " + actionString + "\n";

// Copying the information to a memory-mapped file

SIZE\_T dataSize = keyInfo.size() \* sizeof(CHAR);

OutputDebugString(L" Size of log string ");

OutputDebugString(std::to\_wstring(dataSize).c\_str());

OutputDebugString(L" bytes\n");

if (mappedDataSize + dataSize >= FILESIZE)

{

UninitializeMappingFile();

InitializeMappingFile();

mappedDataSize = 0;

}

memcpy((CHAR\*)pMappedData + mappedDataSize, keyInfo.c\_str(), dataSize);

mappedDataSize += dataSize;

}

return 0;

}

// Func for call RecordKeyPress in separate thread

VOID CallRecordKeyPressInThread(CONST CHAR\* actionString) {

HANDLE hRecordThread = CreateThread(NULL, 0, RecordKeyPressThread, (VOID\*)actionString, 0, NULL);

if (hRecordThread == NULL) {

MessageBox(NULL, L"CreateThread failed!", L"Error", MB\_ICONERROR);

return;

}

else {

// Waiting thread

WaitForSingleObject(hRecordThread, INFINITE);

// Close thread handle

CloseHandle(hRecordThread);

}

}

VOID InitializeMappingFile() {

// CreateFile

hFile = CreateFile(

fileName, // Filename

GENERIC\_READ | GENERIC\_WRITE, // Access mode (read and write)

0, // No sharing

NULL, // Default protection

CREATE\_ALWAYS, // Create a new file or overwrite an existing

FILE\_ATTRIBUTE\_NORMAL, // File attributes (normal)

NULL // Template for creating files

);

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

MessageBox(NULL, L"CreateFile failed!", L"Error", MB\_ICONERROR);

return;

}

// CreateFileMapping

hMapFile = CreateFileMapping(

hFile, // File descriptor

NULL, // Default protection

PAGE\_READWRITE, // Display access mode (read and write)

0, // Display file size (0 means the whole file)

FILESIZE, // Display file size (the highest byte)

NULL // Display file name

);

if (hMapFile == NULL) {

MessageBox(NULL, L"CreateFileMapping failed!", L"Error", MB\_ICONERROR);

CloseHandle(hFile);

return;

}

// Display the mapped file in memory

pMappedData = MapViewOfFile(hMapFile, FILE\_MAP\_ALL\_ACCESS, 0, 0, FILESIZE);

if (pMappedData == NULL) {

MessageBox(NULL, L"MapViewOfFile failed!", L"Error", MB\_ICONERROR);

CloseHandle(hMapFile);

CloseHandle(hFile);

return;

}

}

VOID UninitializeMappingFile() {

if (pMappedData != NULL) {

UnmapViewOfFile(pMappedData);

pMappedData = NULL;

}

if (hMapFile != NULL) {

CloseHandle(hMapFile);

hMapFile = NULL;

}

if (hFile != INVALID\_HANDLE\_VALUE) {

CloseHandle(hFile);

hFile = INVALID\_HANDLE\_VALUE;

}

}

// Function for writing coordinates to the registry

VOID SaveCoordinatesToRegistry() {

HKEY hKey;

if (RegCreateKey(HKEY\_CURRENT\_USER, subKey, &hKey) == ERROR\_SUCCESS) {

RegSetValueEx(hKey, valueNameX, 0, REG\_DWORD, (const BYTE\*)&airplaneX, sizeof(DWORD));

RegSetValueEx(hKey, valueNameY, 0, REG\_DWORD, (const BYTE\*)&airplaneY, sizeof(DWORD));

RegSetValueEx(hKey, valueNameSpeed, 0, REG\_DWORD, (const BYTE\*)&airplaneSpeed, sizeof(DWORD));

RegCloseKey(hKey);

}

}

// Function for reading coordinates from the registry

VOID LoadCoordinatesFromRegistry() {

HKEY hKey;

if (RegOpenKey(HKEY\_CURRENT\_USER, subKey, &hKey) == ERROR\_SUCCESS) {

DWORD x, y, speed;

DWORD dataSize = sizeof(DWORD);

if (RegQueryValueEx(hKey, valueNameX, 0, NULL, (LPBYTE)&x, &dataSize) == ERROR\_SUCCESS &&

RegQueryValueEx(hKey, valueNameY, 0, NULL, (LPBYTE)&y, &dataSize) == ERROR\_SUCCESS &&

RegQueryValueEx(hKey, valueNameSpeed, 0, NULL, (LPBYTE)&speed, &dataSize) == ERROR\_SUCCESS)

{

airplaneX = x;

airplaneY = y;

airplaneSpeed = speed;

}

RegCloseKey(hKey);

}

}

//function for writing logs to the windows event log

VOID WriteToEventLog(std::wstring message) {

HANDLE hEventLog = RegisterEventSource(NULL, L"AirplaneСrash");

if (hEventLog) {

// Getting current time

SYSTEMTIME currentTime;

GetLocalTime(&currentTime);

// Forming a strings with information about the time and the key

std::wstring time = L"Time: " + std::to\_wstring(currentTime.wHour) + L":" +

std::to\_wstring(currentTime.wMinute) + L":" + std::to\_wstring(currentTime.wSecond);

std::wstring logMessage = time + L" => Action: " + message;

const wchar\_t\* messageStrings[1];

messageStrings[0] = logMessage.c\_str();

ReportEvent(hEventLog, EVENTLOG\_INFORMATION\_TYPE, 0, 0, NULL, 1, 0, messageStrings, NULL);

DeregisterEventSource(hEventLog);

}

}

Листинг 2 – Файл *AirplaneDef.h*

#pragma once

// Identifiers of interface elements

#define ID\_MAIN\_TIMER 1

#define ID\_SPEED\_LABEL 2

// Airplane Constants

CONST INT MAX\_SPEED = 15;

CONST INT AIRPLANE\_WIDTH = 150;

CONST INT AIRPLANE\_HEIGHT = 50;

// FileMapping Constants

CONST WCHAR\* fileName = L"flight\_recorder.txt";

CONST INT FILESIZE = 1048576; // 32000 click = 1 mb

// Registry Constants

CONST LPCWSTR subKey = L"Software\\AirplaneAppWinApi";

CONST LPCWSTR valueNameX = L"AirplaneX";

CONST LPCWSTR valueNameY = L"AirplaneY";

CONST LPCWSTR valueNameSpeed = L"AirplaneSpeed";