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»

Выполнил:
студент гр. 153504
Подвальников А.С.

Проверил:
Гриценко Н.Ю.

Минск 2023

СОДЕРЖАНИЕ

1 ЦЕЛИ РАБОТЫ

Изучить средства синхронизации и взаимного исключения операционной системы Windows. Изучить средств синхронизации и взаимного исключения. Реализовать многозадачное приложение для моделирования гонки машин, где средства синхронизации используются для синхронизации движения автомобилей.

2 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Средства синхронизации и взаимного исключения в операционной системе Windows предоставляют механизмы для управления параллельным выполнением процессов и потоков, обеспечивая правильное взаимодействие и координацию между ними. Эти средства позволяют избежать гонок данных и других проблем, связанных с параллельным выполнением. Важно учитывать, что неправильное использование средств синхронизации и взаимного исключения может привести к блокировкам и деградации производительности системы. Для работы с средствами синхронизации и взаимного исключения в Windows разработан ряд функций и объектов. Из наиболее распространенных средств синхронизации следующие: Мьютексы (используются для ограничения доступа к ресурсам только одним потоком или процессом в определенный момент времени. Они широко применяются для синхронизации между процессами.), Семафоры (позволяют контролировать количество потоков, которые могут получить доступ к ресурсу. Это полезное средство для ограничения параллельного доступа к ограниченному количеству ресурсов.), Критические секции (представляют собой участок кода, к которому может получить доступ только один поток в определенный момент времени. Они часто используются для синхронизации внутри одного процесса.), События (позволяют одному потоку сигнализировать другим потокам о наступлении определенного события. Это может быть полезно для ожидания выполнения определенных условий.), Критические ресурсы (может описывать любой ресурс, к которому нужно обеспечить эксклюзивный доступ. Используется с помощью различных механизмов синхронизации).

3 РЕЗУЛЬТАТЫ ВЫПОЛНЕНИЯ ЛАБОРАТОРНОЙ РАБОТЫ

В ходе выполнения лабораторной работы было реализовано многозадачное приложение для моделирования гонки машин, где средства синхронизации используются для синхронизации движения автомобилей. Результат работы программы показан на рисунке 3.1.

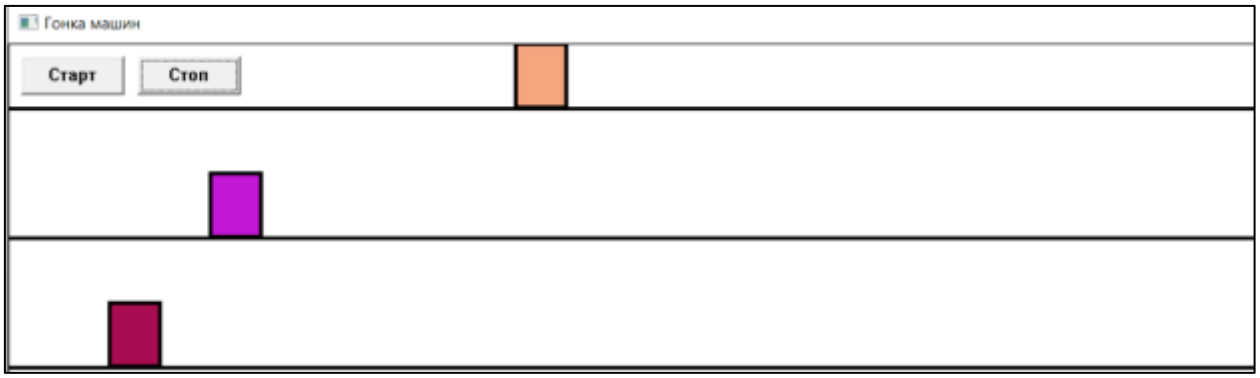

Рисунок 3.1 – Результат работы программы

ВЫВОДЫ

В ходе выполнения данной лабораторной работы были изучены средства синхронизации и взаимного исключения операционной системы Windows. Изучены средства синхронизации и взаимного исключения. Реализовано многозадачное приложение для моделирования гонки машин, где средства синхронизации используются для синхронизации движения автомобилей.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Щупак Ю. Win32 API. Разработка приложений для Windows. – СПб: Питер, 2008. – 592 с.: ип.

[2] [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/en-us/windows/win32/api/winreg/ – Дата доступа 20.10.2023

[3] [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/en-us/windows/win32/sysinfo/enumerating-registry-subkeys – Дата доступа 20.10.2023

[4] [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/en-us/windows/win32/sysinfo/registry – Дата доступа 20.10.2023

[5] [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/en-us/windows/win32/sysinfo/about-the-registry – Дата доступа 20.10.2023

[6] [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/en-us/windows/win32/sysinfo/registry-functions – Дата доступа 21.10.2023

ПРИЛОЖЕНИЕ А
(обязательное)
Листинг кода

Main.cpp

```
#include <windows.h>
#include <stdio.h>
#include <time.h>

struct Car {
    int position;
    int speed;
    COLORREF color;
};

const int NUM_CARS = 3;
Car cars[NUM_CARS];
bool isRaceStarted = false;
int carWidth = 40;
int carHeight = 50;

HWND hwnd;
HDC hdc;
HDC memDC;
HBITMAP memBitmap;
HBITMAP hOldBitmap;
int screenWidth, screenHeight;
PAINTSTRUCT ps;

HANDLE startRaceEvent; // Событие начала гонки
HANDLE stopRaceEvent;  // Событие остановки гонки

void GetScreenSize() {
    screenWidth = GetSystemMetrics(SM_CXSCREEN);
    screenHeight = GetSystemMetrics(SM_CYSCREEN);
}

void StartRace() {
    isRaceStarted = true;
    SetEvent(startRaceEvent); // Устанавливаем событие начала гонки
    for (int i = 0; i < NUM_CARS; ++i) {
        cars[i].position = 0;
        cars[i].speed = rand() % 5 + 1;
        cars[i].color = RGB(rand() % 256, rand() % 256, rand() % 256);
    }
}

void StopRace() {
    isRaceStarted = false;
```

```
    SetEvent(stopRaceEvent); // Устанавливаем событие остановки гонки
}

DWORD WINAPI CarThread(LPVOID param) {
    Car* car = (Car*)param;
    while (1) {
        // Ожидаем событие начала гонки
        WaitForSingleObject(startRaceEvent, INFINITE);

        while (isRaceStarted) {
            car->position += car->speed;
            if (car->position > screenWidth) {
                car->position = -carWidth;
            }
            Sleep(100);
        }

        // Ожидаем событие остановки гонки
        WaitForSingleObject(stopRaceEvent, INFINITE);
    }
    return 0;
}

void UpdateCarPosition() {
    for (int i = 0; i < NUM_CARS; ++i) {
        cars[i].position += cars[i].speed;
        if (cars[i].position > screenWidth) {
            cars[i].position = -carWidth;
        }
    }
}

void RedrawWindow(HWND hwnd) {
    InvalidateRect(hwnd, NULL, TRUE);
    UpdateWindow(hwnd);
}

LRESULT CALLBACK WindowProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM
lParam) {
    switch (uMsg) {
        case WM_PAINT: {
            hdc = BeginPaint(hwnd, &ps);

            // Очищаем экран (закрашиваем фон)
            HBRUSH hBackgroundBrush = CreateSolidBrush(RGB(255, 255,
255)); // Белый фон
            SelectObject(hdc, hBackgroundBrush);
            Rectangle(hdc, 0, 0, screenWidth, 300);
            DeleteObject(hBackgroundBrush);

            for (int i = 0; i < NUM_CARS; ++i) {
```

```
                // Рисуем трассу под машиной
                HPEN hPen = CreatePen(PS_SOLID, 3, RGB(0, 0, 0));
                SelectObject(hdc, hPen);
                MoveToEx(hdc, 0, i * 100 + carHeight, NULL);
                LineTo(hdc, screenWidth, i * 100 + carHeight);
                DeleteObject(hPen);

                HBRUSH hBrush = CreateSolidBrush(cars[i].color);
                SelectObject(hdc, hBrush);
                Rectangle(hdc, cars[i].position, i * 100, cars[i].position
+ carWidth, i * 100 + carHeight);
                DeleteObject(hBrush);
            }
            EndPaint(hwnd, &ps);
            break;
        }
        case WM_COMMAND:
            if (LOWORD(wParam) == 1) {
                StartRace();
                SetTimer(hwnd, 1, 100, NULL);  // Устанавливаем таймер с
интервалом 100 мс (0.1 сек)
            } else if (LOWORD(wParam) == 2) {
                StopRace();
                KillTimer(hwnd, 1);  // Убираем таймер
            }
            break;
        case WM_TIMER:
            if (wParam == 1) {
                UpdateCarPosition();
                RedrawWindow(hwnd);
            }
            break;
    }
    return DefWindowProc(hwnd, uMsg, wParam, lParam);
}

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

    srand(time(NULL));

    // Инициализируем события
    startRaceEvent = CreateEvent(NULL, FALSE, FALSE, NULL);
    stopRaceEvent = CreateEvent(NULL, FALSE, FALSE, NULL);

    HANDLE hThreads[NUM_CARS];
    for (int i = 0; i < NUM_CARS; ++i) {
        hThreads[i] = CreateThread(NULL, 0, CarThread, &cars[i], 0, NULL);
        if (hThreads[i] == NULL) {
            MessageBox(NULL, "Failed to create thread!", "Error",
MB_ICONERROR);
```

```
            return 1;
        }
    }

    WNDCLASS wc = {0};
    wc.lpfnWndProc = WindowProc;
    wc.hInstance = hInstance;
    wc.lpszClassName = "RaceWindowClass";
    RegisterClass(&wc);

    hwnd = CreateWindow(
        "RaceWindowClass",
        "Гонка машин",
        WS_OVERLAPPEDWINDOW,
        100,
        100,
        screenWidth,
        300,
        NULL,
        NULL,
        hInstance,
        NULL
    );

    CreateWindow("button", "Старт", WS_CHILD | WS_VISIBLE, 10, 10, 80, 30,
hwnd, (HMENU)1, hInstance, NULL);
    CreateWindow("button", "Стоп", WS_CHILD | WS_VISIBLE, 100, 10, 80, 30,
hwnd, (HMENU)2, hInstance, NULL);

    hdc = GetDC(hwnd);
    memDC = CreateCompatibleDC(hdc);
    memBitmap = CreateCompatibleBitmap(hdc, screenWidth, 300);
    hOldBitmap = (HBITMAP)SelectObject(memDC, memBitmap);

    ShowWindow(hwnd, nCmdShow);
    UpdateWindow(hwnd);

    MSG msg = {0};
    while (GetMessage(&msg, NULL, 0, 0)) {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }

    for (int i = 0; i < NUM_CARS; ++i) {
        WaitForSingleObject(hThreads[i], INFINITE);
        CloseHandle(hThreads[i]);
    }

    SelectObject(memDC, hOldBitmap);
    DeleteObject(memBitmap);
    DeleteDC(memDC);
```

```
    ReleaseDC(hwnd, hdc);

    return 0;
}
```