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

Мамченко К.А.

Проверил:

Гриценко Н.Ю.

Минск 2023

**СОДЕРЖАНИЕ**

[1 Цели работы 3](#_gjdgxs)

[2 Краткие теоретические сведения 4](#_30j0zll)

[3 Полученные результаты](#_1fob9te) 5

[Вывод](#_4q3p755gzys6) 6

[Список использованных источников](#_4q3p755gzys6) 7

[Приложение А (обязательное) Листинг кода](#_4q3p755gzys6) 8

**1 ЦЕЛИ РАБОТЫ**

1 Изучить средства синхронизации и взаимного исключения (Windows).

2 Построить многозадачное приложение для моделирования гонки машин, где средства синхронизации используются для синхронизации движения автомобилей.

**2 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

Синхронизация и взаимное исключение являются ключевыми концепциями в многопоточном программировании и используются для координации доступа к общим ресурсам, таким как общие переменные или общие структуры данных. В вашем случае, общим ресурсом является массив *g\_carPositions*, в котором хранятся позиции машин.

Для обеспечения синхронизации и взаимного исключения в приложении можно использовать мьютексы (мьютексы Windows API). Мьютекс ­ это объект синхронизации, который может быть захвачен одним потоком в определенный момент времени, а все остальные потоки должны ждать его освобождения, прежде чем получить доступ к общему ресурсу. В коде используется функция *WaitForSingleObject* для ожидания доступа к мьютексу и функция *ReleaseMutex* для его освобождения.

Когда поток хочет получить доступ к общему ресурсу (например, обновить позицию машины), он должен сначала захватить мьютекс с помощью *WaitForSingleObject*. Это гарантирует, что только один поток может одновременно обновлять позицию машины. После окончания работы с общим ресурсом поток должен освободить мьютекс с помощью *ReleaseMutex*, чтобы другие потоки могли получить доступ к нему.

Таким образом, использование мьютексов обеспечивает взаимное исключение, предотвращая одновременный доступ нескольких потоков к общему ресурсу и предотвращая состояния гонки, которые могут приводить к непредсказуемому поведению и ошибкам в программе.

Однако важно быть осторожным при использовании мьютексов, чтобы избежать потенциальных проблем, таких как взаимная блокировка (*deadlock*), когда два или более потока блокируют друг друга, ожидая освобождения мьютексов, которые они сами захватили. Чтобы избежать взаимной блокировки, важно правильно организовать порядок захвата и освобождения мьютексов в разных потоках.

В целом, использование мьютексов и правильная синхронизация помогают создать безопасное и надежное многопоточное приложение, где обновление позиций машин будет происходить с соблюдением взаимного исключения и предотвращением состояний гонки.

**3 ПОЛУЧЕННЫЕ РЕЗУЛЬТАТЫ**

На рисунке 3.1 продемонстрирована результат работы программы, где в качестве машин представлены прогресс быры.

Изображение выглядит как снимок экрана, текст, Красочность

Автоматически созданное описание

Рисунок 3.1 ­ Результат работы программы

.

**ВЫВОД**

В результате было разработано для моделирования гонки машин, где средства синхронизации используются для синхронизации движения автомобилей.

Для этого был изучен теоретический материал и средства Win32 Api, используемые для этих целей.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

[1] Использование объектов мьютекса, документация Microsoft [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/ru-ru/windows/win32/sync/using-mutex-objects>.

**ПРИЛОЖЕНИЕ А**

**(обязательное)**

**Листинг кода**

#include <windows.h>

#include <vector>

#include <thread>

const int NUM\_CARS = 5;

const int CAR\_WIDTH = 50;

const int CAR\_HEIGHT = 30;

const int TRACK\_WIDTH = 800;

const int TRACK\_HEIGHT = 400;

// Глобальные переменные

HWND g\_hWnd;

HDC g\_hDC;

std::vector<int> g\_carPositions;

HANDLE g\_hMutex;

COLORREF GetCarColor(int index) {

if (index == 0) {

return RGB(255, 0, 0);

} else if (index == 1) {

return RGB(0, 255, 0);

} else if (index == 2) {

return RGB(0, 0, 255);

} else if (index == 3) {

return RGB(100, 0, 100);

} else if (index == 4) {

return RGB(0, 110, 0);

}

}

// Функция рисования автомобилей

void DrawCars()

{

int gap = 0;

WaitForSingleObject(g\_hMutex, INFINITE); // Ждем доступа к общему ресурсу

for (int i = 0; i < NUM\_CARS; i++)

{

RECT rect;

rect.left = g\_carPositions[i];

rect.right = rect.left + CAR\_WIDTH;

rect.top = gap; // Устанавливаем вертикальную позицию

rect.bottom = CAR\_HEIGHT + gap; // Устанавливаем вертикальную позицию

HBRUSH hBrush = CreateSolidBrush(GetCarColor(i));

FillRect(g\_hDC, &rect, hBrush);

DeleteObject(hBrush);

gap += 50;

}

ReleaseMutex(g\_hMutex); // Освобождаем доступ к общему ресурсу

}

// Функция обновления позиций автомобилей

void UpdateCarPositions(int carIndex)

{

while (true)

{

WaitForSingleObject(g\_hMutex, INFINITE); // Ждем доступа к общему ресурсу

g\_carPositions[carIndex] += rand() % 5 + 1; // Генерация случайного шага

if (g\_carPositions[carIndex] >= TRACK\_WIDTH) // Если автомобиль достиг конца трека, вернуть его в начало

g\_carPositions[carIndex] = 0;

ReleaseMutex(g\_hMutex); // Освобождаем доступ к общему ресурсу

// Обновить окно

InvalidateRect(g\_hWnd, NULL, TRUE);

UpdateWindow(g\_hWnd);

Sleep(100); // Задержка между обновлениями позиций

}

}

// Функция обработки сообщений окна

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

{

switch (uMsg)

{

case WM\_PAINT:

{

PAINTSTRUCT ps;

g\_hDC = BeginPaint(hWnd, &ps);

DrawCars();

EndPaint(hWnd, &ps);

}

return 0;

case WM\_DESTROY:

PostQuitMessage(0);

return 0;

}

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

}

// Функция создания окна

HWND CreateAppWindow(HINSTANCE hInstance)

{

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

WNDCLASS wc = {};

wc.lpfnWndProc = WindowProc;

wc.hInstance = hInstance;

wc.lpszClassName = CLASS\_NAME;

RegisterClass(&wc);

return CreateWindowEx(

0, // дополнительные стили окна

CLASS\_NAME, // имя класса окна

L"Гонка машин", // заголовок окна

WS\_OVERLAPPEDWINDOW, // стиль окна

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

TRACK\_WIDTH, TRACK\_HEIGHT, // размеры окна

NULL, // дескриптор родительского окна

NULL, // дескриптор меню

hInstance, // дескриптор экземпляра приложения

NULL // указатель на данные создания окна

);

}

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

{

g\_hMutex = CreateMutex(NULL, FALSE, NULL); // Создаем мьютекс

g\_carPositions.resize(NUM\_CARS, 0); // Инициализируем позиции автомобилей

// Создаем потоки для каждой машины

std::vector<std::thread> threads;

for (int i = 0; i < NUM\_CARS; i++)

{

threads.push\_back(std::thread(UpdateCarPositions, i));

}

// Создаем окно приложения

g\_hWnd = CreateAppWindow(hInstance);

if (g\_hWnd == NULL)

return 0;

// Отображаем окно

ShowWindow(g\_hWnd, nCmdShow);

UpdateWindow(g\_hWnd);

// Запускаем цикл обработки сообщений

MSG msg = {};

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

{

TranslateMessage(&msg);

DispatchMessage(&msg);

}

// Ожидаем завершения потоков

for (auto& thread : threads)

{

thread.join();

}

CloseHandle(g\_hMutex); // Закрываем мьютекс

return static\_cast<int>(msg.wParam);

}