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

Сивый А.А.

Проверил:

Гриценко Н.Ю.

Минск 2023

**СОДЕРЖАНИЕ**

[1 Цели работы 3](#_gjdgxs)

[2 Краткие теоретические сведения 4](#_30j0zll)

[3 Полученные результаты 5](#_1fob9te)

[Вывод 6](#_4q3p755gzys6)

[Приложение А (обязательное) Листинг кода 7](#_4q3p755gzys6)

**1 ЦЕЛИ РАБОТЫ**

1 Изучить основные понятия и принципы синхронизации и взаимного исключения, такие как критические секции, мьютексы, семафоры, события и таймеры.

2 Научиться создавать открывать, закрывать и удалять объекты синхронизации с помощью функций Win32 API.

3 Научить использовать объекты синхронизации для координации доступа к общим ресурсам и ожидания завершения потоков.

4 Разработать программный продукт на выбранную тему.

**2 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

Многопоточность – это способность программы выполнять несколько задач одновременно с использованием нескольких потоков исполнения. Многопоточность позволяет увеличить производительность и отзывчивость приложений, а также эффективно использовать ресурсы процессора. Однако многопоточность также вводит сложность в разработку и отладку программ, так как потоки могут конкурировать за доступ к общим ресурсам, таким как память, файлы или устройства ввода-вывода. Для того, чтобы избежать проблем, связанных с состоянием гонки, взаимоблокировками или нарушением целостности данных, необходимо синхронизировать выполнение нескольких потоков и обеспечить взаимное исключение при работе с общими ресурсами. WinAPI предоставляет различные средства синхронизации и взаимного исключения для потоков, такие как события, мьютексы, семафоры, таймеры, критические секции и другие. С помощью этих объектов потоки могут ожидать наступления определенных условий, блокировать или освобождать доступ к ресурсам, а также координировать свою работу в правильном порядке.

Функции для работы с объектами синхронизации в WinAPI позволяют создавать, открывать, закрывать, ожидать и управлять различными типами объектов синхронизации, такими как события, мьютексы, семафоры и таймеры. Вот некоторые из этих функций:

CreateEvent – создает или открывает именованный или безымянный объект события.

SetEvent – устанавливает состояние объекта события в сигнальное.

ResetEvent – устанавливает состояние объекта события в несигнальное.

WaitForSingleObject – ожидает, пока объект синхронизации перейдет в сигнальное состояние или истечет указанный интервал времени.

WaitForMultipleObjects – ожидает, пока один или все объекты синхронизации в массиве перейдут в сигнальное состояние или истечет указанный интервал времени.

CreateMutex – создает или открывает именованный или безымянный объект мьютекса.

ReleaseMutex – освобождает владение объектом мьютекса.

CreateSemaphore – увеличивает счетчик объекта семафора на указанное значение.

**3 ПОЛУЧЕННЫЕ РЕЗУЛЬТАТЫ**

На рисунке 3.1 можно наблюдать приглашение для пользователя начать выполнение программы.

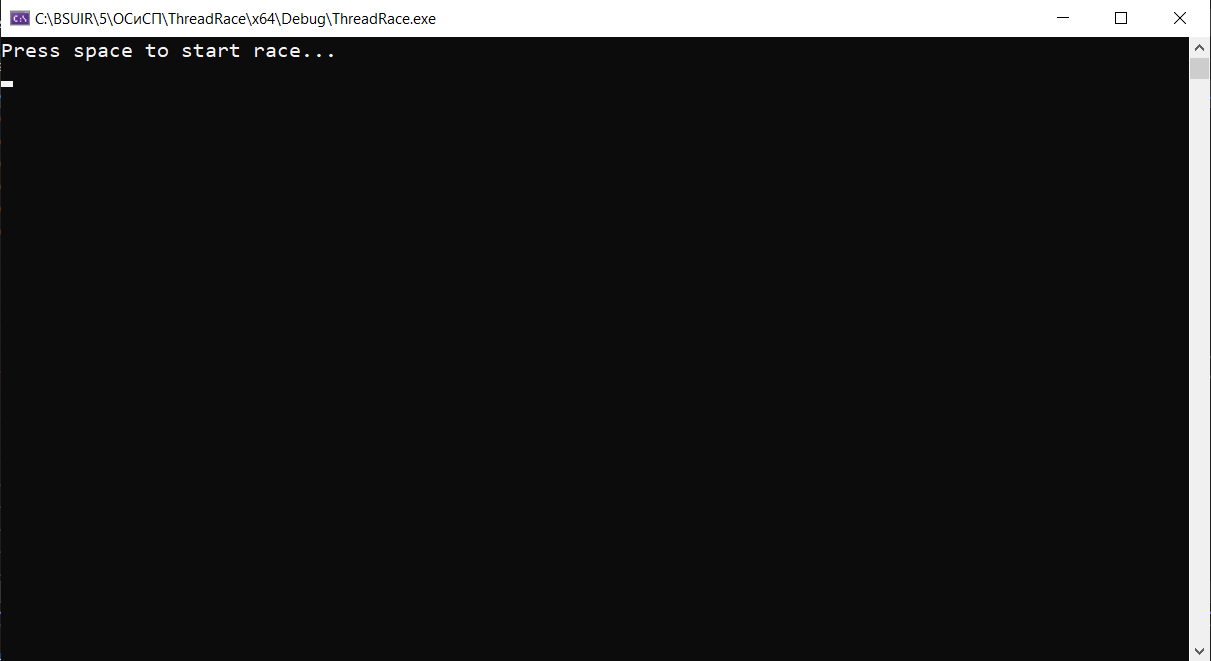


Рисунок 3.1 – Начальный экран программы.

На рисунке 3.2 продемонстрирован ход выполнения программы.

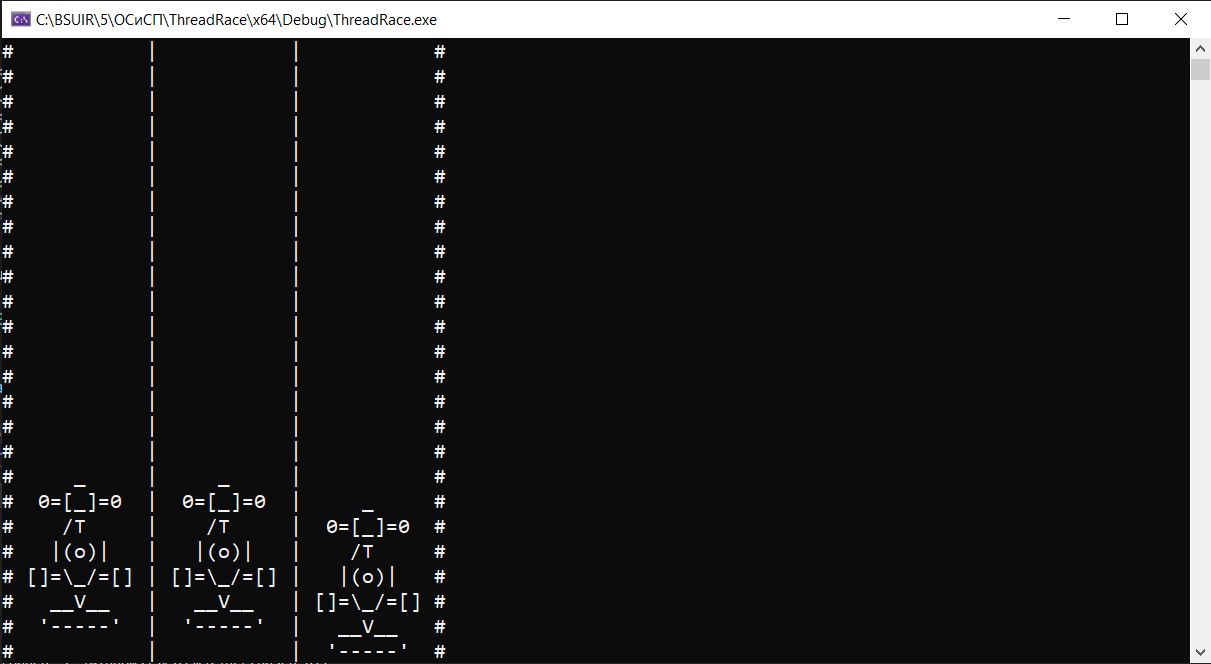


Рисунок 3.2 – Выполнение программы.

На рисунке 3.3 представлен результат выполнения программы.

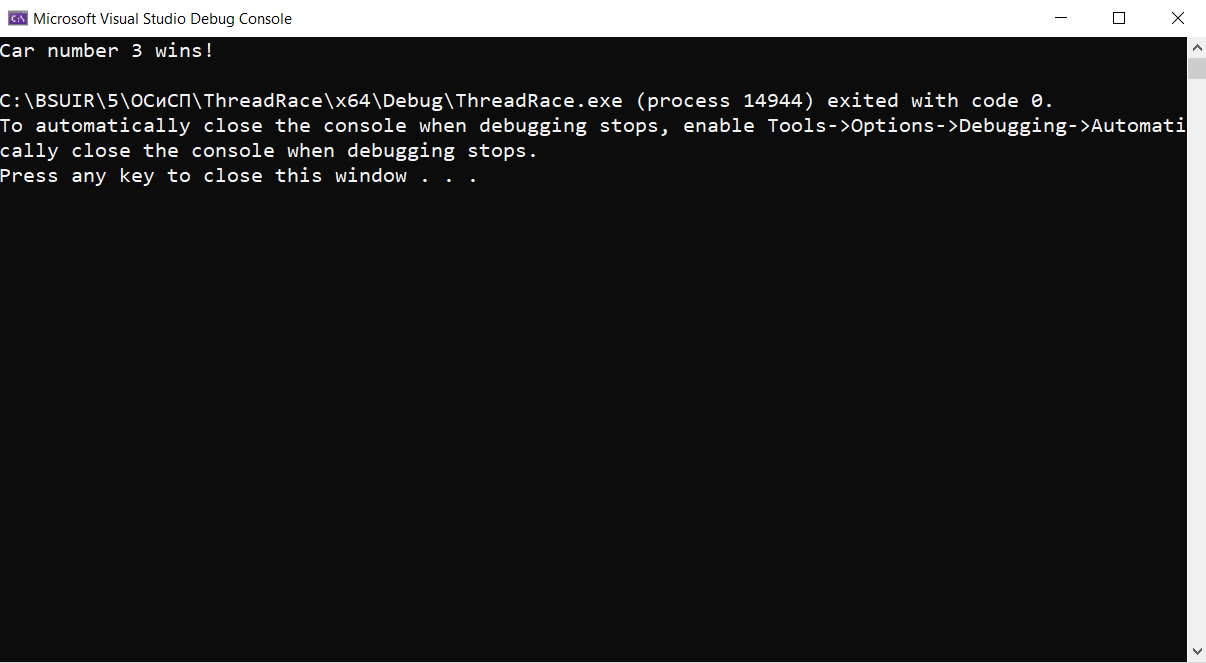


Рисунок 3.3 – Результат выполнения программы.**ВЫВОД**

Средства синхронизации и взаимного исключения являются важными элементами многопоточного программирования, которые позволяют обеспечить корректность, эффективность и безопасность работы приложений, использующих общие ресурсы или выполняющих параллельные вычисления. Windows предоставляет разнообразный набор объектов синхронизации, таких как мьютексы, семафоры, критические секции, события и таймеры, которые можно использовать для решения различных задач синхронизации и взаимного исключения. В результате выполнения лабораторной работы было разработано многозадачное приложение для моделирования гонки машин, где средства синхронизации используются для синхронизации движения автомобилей.

**ПРИЛОЖЕНИЕ А**

**(обязательное)**

**Листинг кода**

#include <iostream>

#include <Windows.h>

#include <stdlib.h>

#include <time.h>

#include <map>

#include <utility> // pair

#include <string>

#include <conio.h>

using namespace std;

bool exitRace = false;

HANDLE mutex;

std::map<int, std::pair<int, int>> cars{

{0, std::pair<int, int>(2, 18)},

{1, std::pair<int, int>(14, 18)},

{2, std::pair<int, int>(26, 18)}

};

string car[7] = { " \_ ",

" 0=[\_]=0 ",

" /T\ ",

" |(o)| ",

"[]=\\\_/=[]",

" \_\_V\_\_ ",

" '-----' "

};

void gotoxy(int x, int y) // to print in any place

{

COORD coord;

coord.X = x;

coord.Y = y;

SetConsoleCursorPosition(GetStdHandle(STD\_OUTPUT\_HANDLE), coord);

}

void hideCursor() { // hide cursor

CONSOLE\_CURSOR\_INFO cursor;

cursor.dwSize = 100;

cursor.bVisible = false;

SetConsoleCursorInfo(GetStdHandle(STD\_OUTPUT\_HANDLE), &cursor);

}

DWORD WINAPI translate(void\* data) {

int carIndex = \*((int\*)data);

while (!exitRace)

{

WaitForSingleObject(mutex, INFINITE);

// Отрисовка трассы

for (int i = 0; i < 25; i++) {

gotoxy(0, i);

cout << "# #";

gotoxy(12, i);

cout << "| |";

}

for (int i = 0; i < 3; i++)

{

for (int j = 0; j < 7; j++)

{

gotoxy(cars[i].first, j + cars[i].second);

cout << car[j];

}

}

if (cars[carIndex].second < 25 && cars[carIndex].second > 0)

cars[carIndex].second--;

if (cars[carIndex].second == 0)

{

system("cls");

exitRace = true;

cout << "Car number " << carIndex + 1 << " wins!" << endl;

}

ReleaseMutex(mutex);

Sleep(100);

}

return 0;

}

int main() {

cout << "Press space to start race..." << endl;

while (true)

{

if (\_kbhit())

{

char c = \_getch();

if (c == 32)

break;

else

return 1;

}

}

cout << "1..." << endl;

Sleep(1000);

cout << "2..." << endl;

Sleep(1000);

cout << "3..." << endl;

cout << "Go go go!" << endl;

Sleep(500);

hideCursor();

HANDLE hThread1, hThread2, hThread3;

mutex = CreateMutex(NULL, FALSE, NULL);

int car1 = 0, car2 = 1, car3 = 2;

hThread1 = CreateThread(NULL, 0, translate, &car1, 0, NULL);

hThread2 = CreateThread(NULL, 0, translate, &car2, 0, NULL);

hThread3 = CreateThread(NULL, 0, translate, &car3, 0, NULL);

SetThreadPriority(hThread2, THREAD\_PRIORITY\_TIME\_CRITICAL);

SetThreadPriority(hThread3, THREAD\_PRIORITY\_LOWEST);

WaitForSingleObject(hThread1, INFINITE);

WaitForSingleObject(hThread2, INFINITE);

WaitForSingleObject(hThread3, INFINITE);

CloseHandle(mutex);

return 0;

}