

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний аерокосмічний університет ім. М.Є. Жуковського
“Харківський авіаційний інститут”

Кафедра комп’ютерних систем, мереж і кібербезпеки

Лабораторна робота № 3

з дисципліни «Системне програмування»

**«Изучение системных вызовов Win32 API работы
объектов синхронизации и алгоритмов межпоточной
и межпроцессной синхронизации в ОС Windows»**

XAI.503.535a.20B. 123,1805038

Виконав студент гр. 535a
(№ групи)

Мартиненко Антон Олександрович
(П.І.Б.)

(підпис, дата)

Перевірив асистент

(науковий ступінь, вчене звання, посада)

Лейченко К.М.
(підпис, дата) (П.І.Б.)

Цель работы:

1. Изучение встроенных объектов синхронизации в ОС Windows.
2. Изучение системных вызовов Win32 API для реализации алгоритмов межпоточной и межпроцессной синхронизации.

Постановка задачи:

Программа 1:

Требуется разработать программу, которая контролирует наличие только одного экземпляра самой программы в памяти. Т.е. при попытке запустить программу при уже наличии одного запущенного экземпляра, программа выдает ошибку о невозможности старта. Сама программа просто должна вывести в консоль фразу "Is Running" в случае успешного запуска.

Программа 2:

Программа должна контролировать кол-во одновременно открытых указателей на файлы между всеми запущенными потоками. Приложение при старте создает заданное кол-во потоков, где каждый поток при старте переходит в спящий режим на период времени от 1 до 3 сек, потом пытается открыть файл для записи и записать в него время выполнения данной операции. После чего подождать от 1 до 3 сек. И закрыть файл. Программа в процессе работы не может открыть больше, чем заданное кол-во файловых указателей. В случае, когда уже новый поток не может превысить кол-во одновременно открытых файлов он ожидает пока хотя бы один файл не будет закрыт.

Программа 3:

Необходимо написать программу, которая реализует 3х поточную работу (любой алгоритм: например, 1 поток считает сумму чисел в массиве, 2ой поток считает среднее значение в массиве, 3ий поток считает макс. и мин значение в массиве). Сам алгоритм вычисления с обращением к критическим операторам (обращение к массиву) должен быть реализован в виде взаимного исключения одновременного обращения к источнику данных (массиву).

Задача: программа должна иметь 2 режима работы: с взаимным исключением и без. В каждом режиме должен производиться замер времени работы. Для получения более ощутимых интервалов работать с массивом от 50 тыс. элементов.

Выполнение работы**Код программы №1****Lab3.1.cpp**

```
#include "pch.h"

using namespace std;

int main()
{
    HANDLE mutex = CreateMutexA(NULL, FALSE, "yMutex");

    if (WaitForSingleObject(mutex, 0) == WAIT_OBJECT_0)
    {
        cout << "Program is running!" << endl;
        system("pause");
        ReleaseMutex(mutex);
    }
    else
```

```

{
    cout << "Error!Program is already running!" << endl;
    system("pause");
}

CloseHandle(mutex);
return 0;
}

```

Тестирование программы №1

Название теста	Входные данные	Выходные данные
Первичный запуск программы	Запуск программы	Вывод в консоль: Program is running
Вторичный запуск программы	Запуск программы	Вывод в консоль: Error! Program is already running!

Скриншоты работы программы №1

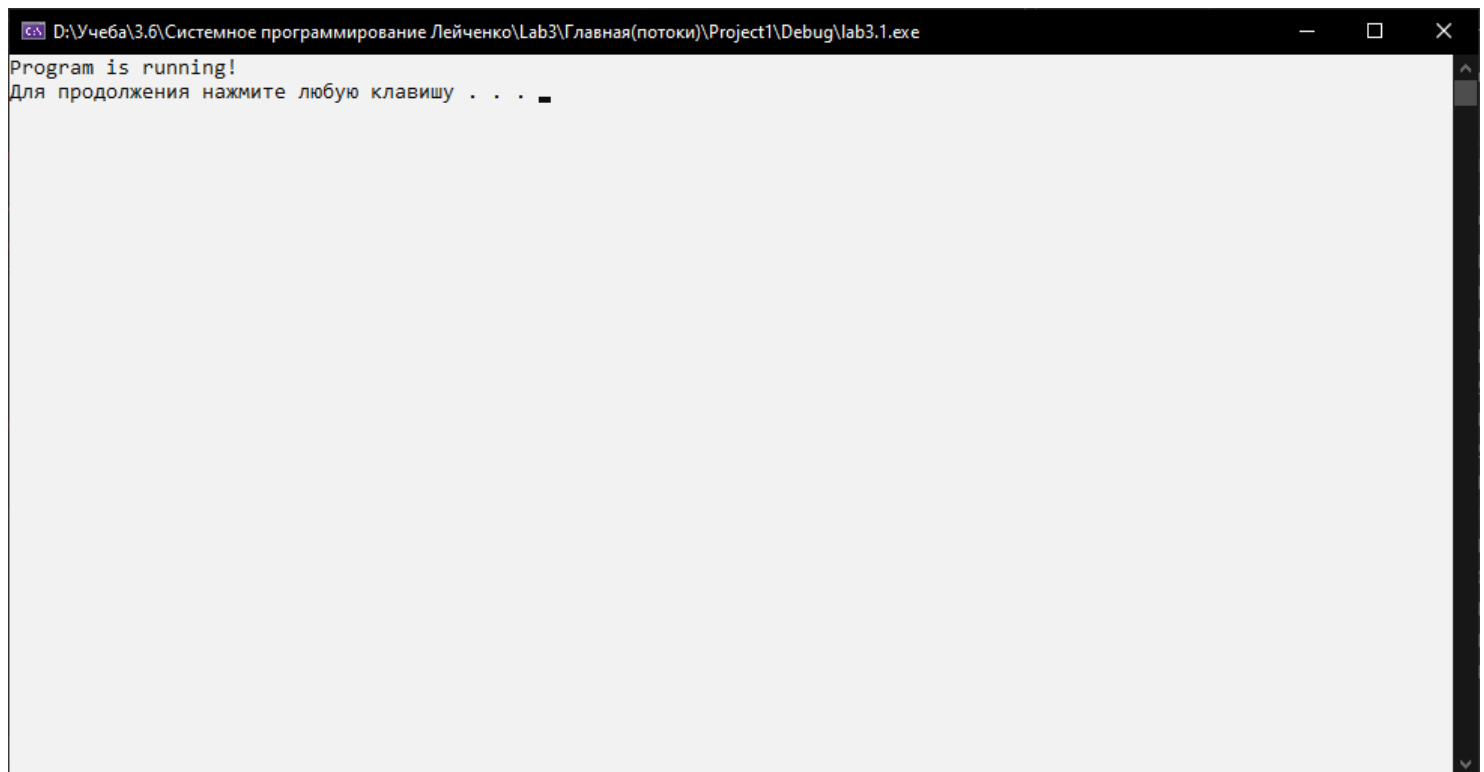


Рис. 1 – Скриншот работы программы

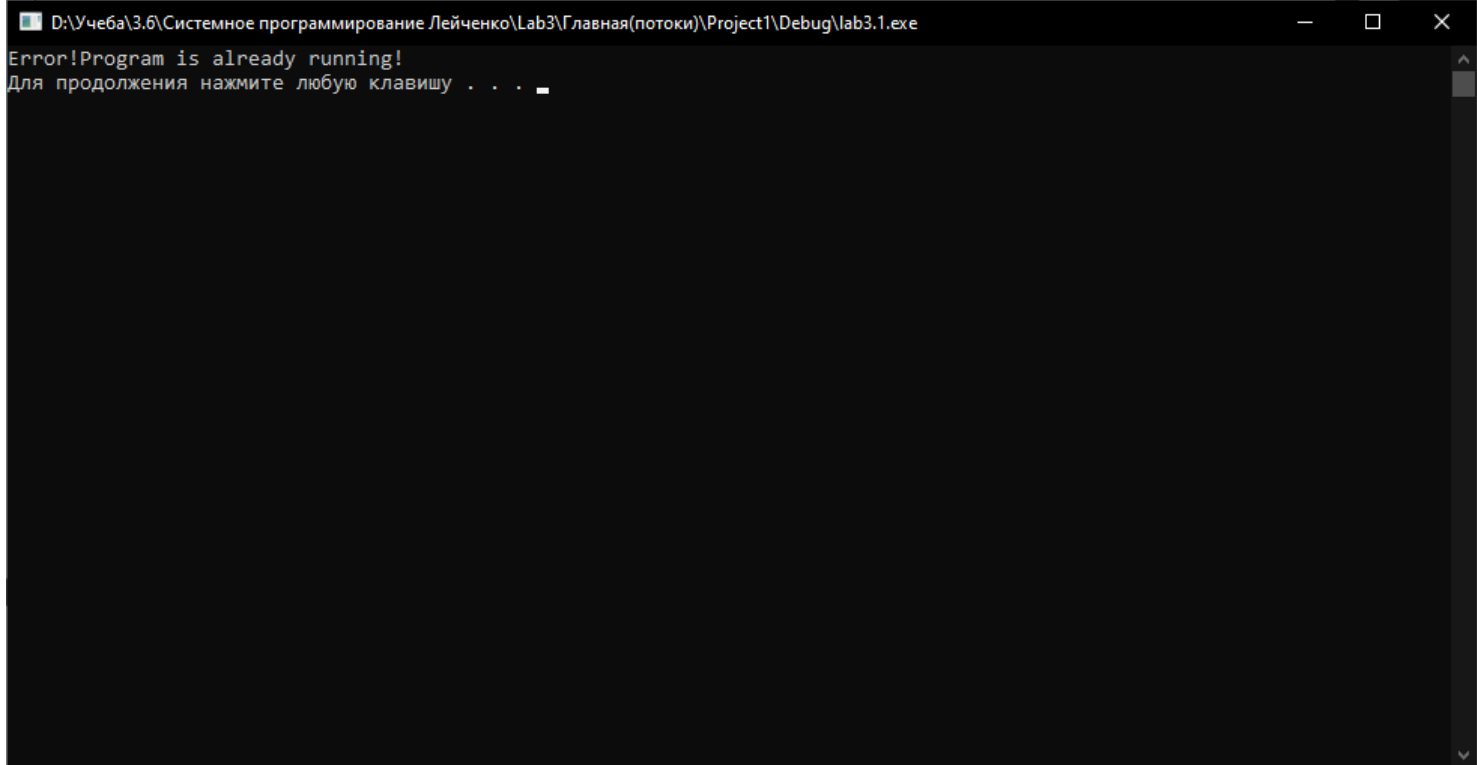


Рис. 2 – Скриншот работы программы

Код программы №2

Task3.2.cpp

```
#include "pch.h"

using namespace std;

HANDLE semaphore;

DWORD WINAPI thread_function(LPVOID param)
{
    int* params = (int*)param;
    clock_t start = clock();
    int thread_number = params[0];
    LPSTR str = new CHAR[128];

    DWORD result = WaitForSingleObject(semaphore, 500);
    while (result != WAIT_OBJECT_0)
    {
        result = WaitForSingleObject(semaphore, 1000);
        printf("Thread number %i waiting for semaphore\n", thread_number);
    }

    printf("Thread number %i decrement semaphore. Going to sleep\n", thread_number);

    Sleep(params[1] * 1000);

    HANDLE file = CreateFileA("work_result.txt", GENERIC_WRITE, FILE_SHARE_WRITE, NULL, OPEN_ALWAYS,
    FILE_ATTRIBUTE_NORMAL, NULL);
    if (file == INVALID_HANDLE_VALUE)
    {
        ReleaseSemaphore(semaphore, 1, NULL);
        return 0;
    }
    SetFilePointer(file, 0, NULL, FILE_END);

    clock_t finish = clock();
    float time_elapsed = (finish - start) / CLK_TCK;

    sprintf(str, "Thread number %i: %f sec\n\0", thread_number, time_elapsed);
    WriteFile(file, str, strlen(str), NULL, NULL);
}
```

```

    CloseHandle(file);

    printf("Thread number %i released semaphore.\n", thread_number);
    ReleaseSemaphore(semaphore, 1, NULL);
    return 0;
}

int getRandValue(int min, int max)
{
    return min + rand() % (max + 1 - min);
}

int main()
{
    int max_handles;
    int max_threads;
    srand(time(NULL));
    printf("Input max number of handles : ");
    scanf("%i", &max_handles);

    printf("Input max number of threads : ");
    scanf("%i", &max_threads);

    //Создание семафора
    HANDLE* threads = new HANDLE[max_threads];
    semaphore = CreateSemaphoreA(NULL, max_handles, max_handles, "MySemaphore");
    if (semaphore == NULL) {
        printf("Error of creating the semaphore");
        system("pause");
        return 0;
    }

    //Создание файла
    HANDLE file = CreateFileA("work_result.txt", GENERIC_WRITE, FILE_SHARE_WRITE, NULL, CREATE_ALWAYS,
    FILE_ATTRIBUTE_NORMAL, NULL);
    if (file == INVALID_HANDLE_VALUE)
    {
        CloseHandle(file);
        return 0;
    }
    CloseHandle(file);

    for (int i = 0; i < max_threads; i++)
    {
        int* params = new int[2];
        params[0] = i; //Количество потоков
        params[1] = getRandValue(1, 5); //Время сна.

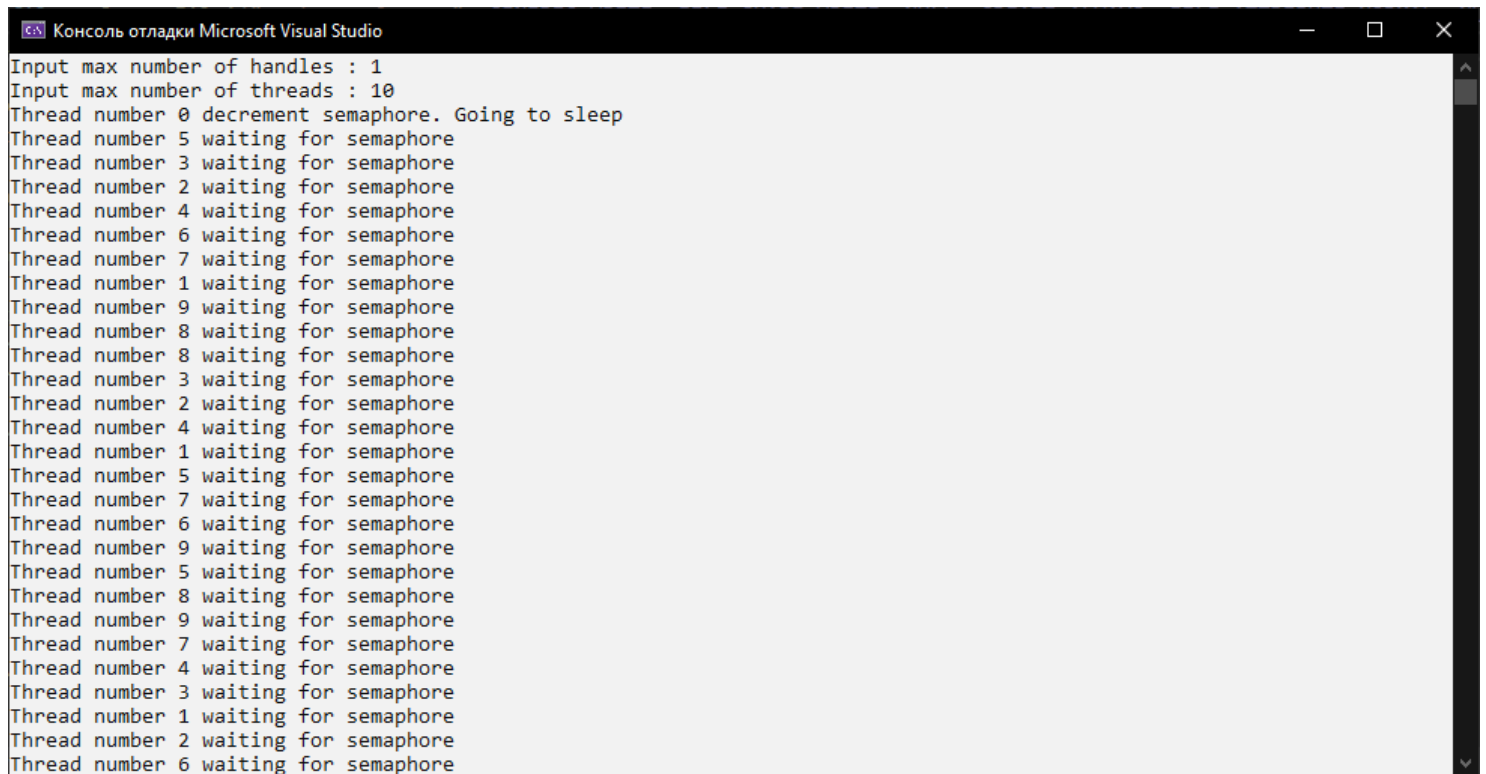
        threads[i] = CreateThread(
            NULL, // дескриптор защиты
            0, // начальный размер стека
            thread_function, // функция потока
            (LPVOID)params, // параметры потока
            NULL, // опции создания
            NULL); // идентификатор потока
    }
    WaitForMultipleObjects(max_threads, threads, TRUE, INFINITE);
    return 0;
}

```

Тестирование программы №2

Название теста	Входные данные	Выходные данные
Проверка работоспособности	Количество указателей: 1 Количество потоков: 10	Вывод информации в файл
Вызов ошибки	Количество указателей: 0 Количество потоков: 10	Вывод в консоль: Error of creating the semaphore

Скриншоты работы программы №2

The image shows a screenshot of the 'Консоль отладки Microsoft Visual Studio' (Microsoft Visual Studio Debug Console) window. The window has a title bar with the Visual Studio logo and standard minimize, maximize, and close buttons. The console area is light gray and contains the following text:

```
Input max number of handles : 1
Input max number of threads : 10
Thread number 0 decrement semaphore. Going to sleep
Thread number 5 waiting for semaphore
Thread number 3 waiting for semaphore
Thread number 2 waiting for semaphore
Thread number 4 waiting for semaphore
Thread number 6 waiting for semaphore
Thread number 7 waiting for semaphore
Thread number 1 waiting for semaphore
Thread number 9 waiting for semaphore
Thread number 8 waiting for semaphore
Thread number 8 waiting for semaphore
Thread number 3 waiting for semaphore
Thread number 2 waiting for semaphore
Thread number 4 waiting for semaphore
Thread number 1 waiting for semaphore
Thread number 5 waiting for semaphore
Thread number 7 waiting for semaphore
Thread number 6 waiting for semaphore
Thread number 9 waiting for semaphore
Thread number 5 waiting for semaphore
Thread number 8 waiting for semaphore
Thread number 9 waiting for semaphore
Thread number 7 waiting for semaphore
Thread number 4 waiting for semaphore
Thread number 3 waiting for semaphore
Thread number 1 waiting for semaphore
Thread number 2 waiting for semaphore
Thread number 6 waiting for semaphore
```

Рис. 3 – Скриншот работы программы

1	Thread number 0: 5.000000 sec
2	Thread number 2: 7.000000 sec
3	Thread number 6: 9.000000 sec
4	Thread number 3: 14.000000 sec
5	Thread number 1: 19.000000 sec
6	Thread number 5: 23.000000 sec
7	Thread number 8: 27.000000 sec
8	Thread number 9: 29.000000 sec
9	Thread number 4: 31.000000 sec
10	Thread number 7: 34.000000 sec
11	

Рис. 4 – Скриншот работы программы

Код программы №3

Lab3.3.cpp

```
#include "pch.h"
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <Windows.h>
#include <time.h>
#include <locale.h>
#define ARRAY_MAX 200000

CRITICAL_SECTION section;
int* array;

DWORD WINAPI thread_get_min(LPVOID use_critical_section);
DWORD WINAPI thread_get_max(LPVOID use_critical_section);
DWORD WINAPI thread_get_avg(LPVOID use_critical_section);

int getRandValue(int min, int max)
{
    return min + rand() % (max + 1 - min);
}

void arrayFilling(int* array)
{
    for (int i = 0; i < ARRAY_MAX; i++)
    {
        array[i] = getRandValue(0, 100);
    }
}

int main()
{
    setlocale(LC_CTYPE, "rus");
    HANDLE* threads;
    clock_t start;
    float elapsed_time;
    srand(time(NULL));
    InitializeCriticalSection(&section);

    array = new int[ARRAY_MAX];
    arrayFilling(array);

    start = clock();
    threads = new HANDLE[3];
    threads[0] = CreateThread(NULL, 0, thread_get_min, (LPVOID)TRUE, NULL, NULL);
    threads[1] = CreateThread(NULL, 0, thread_get_avg, (LPVOID)TRUE, NULL, NULL);
    threads[2] = CreateThread(NULL, 0, thread_get_max, (LPVOID)TRUE, NULL, NULL);
    WaitForMultipleObjects(3, threads, TRUE, INFINITE);

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

```

    {
        CloseHandle(threads[i]);
    }
    elapsed_time = ((float)(clock() - start)) / CLK_TCK;
    printf("Runtime with critical section is %f sec\n\n", elapsed_time);

    start = clock();
    threads = new HANDLE[3];
    threads[0] = CreateThread(NULL, 0, thread_get_min, (LPVOID)FALSE, NULL, NULL);
    threads[1] = CreateThread(NULL, 0, thread_get_avg, (LPVOID)FALSE, NULL, NULL);
    threads[2] = CreateThread(NULL, 0, thread_get_max, (LPVOID)FALSE, NULL, NULL);
    WaitForMultipleObjects(3, threads, TRUE, INFINITE);
    for (int i = 0; i < 3; i++)
    {
        CloseHandle(threads[i]);
    }
    elapsed_time = ((float)(clock() - start)) / CLK_TCK;
    printf("Runtime without critical section is %f sec\n\n", elapsed_time);

    DeleteCriticalSection(&section);
}

DWORD WINAPI thread_get_min(LPVOID use_critical_section)
{
    if ((bool)use_critical_section)
    {
        while (!TryEnterCriticalSection(&section))
        {
        }
    }

    int min = array[0];
    for (int i = 0; i < ARRAY_MAX; i++)
    {
        if (min > array[i])
            min = array[i];
    }
    printf("Minimum: %i\n", min);

    if ((bool)use_critical_section)
    {
        LeaveCriticalSection(&section);
    }

    return 0;
}

DWORD WINAPI thread_get_max(LPVOID use_critical_section)
{
    if ((bool)use_critical_section)
    {
        while (!TryEnterCriticalSection(&section))
        {
        }
    }

    int max = array[0];
    for (int i = 0; i < ARRAY_MAX; i++)
    {
        if (max < array[i])
            max = array[i];
    }
    printf("Maximum: %i\n", max);

    if ((bool)use_critical_section)
    {
        LeaveCriticalSection(&section);
    }
}

```



```

    return 0;
}

DWORD WINAPI thread_get_avg(LPVOID use_critical_section)
{
    if ((bool)use_critical_section)
    {
        while (!TryEnterCriticalSection(&section))
        {
        }
    }

    float avg = 0;
    for (int i = 0; i < ARRAY_MAX; i++)
    {
        avg += array[i];
    }
    printf("Average: %f\n", avg / ARRAY_MAX);

    if ((bool)use_critical_section)
    {
        LeaveCriticalSection(&section);
    }

    return 0;
}

```

Тестирование программы №3

Название теста	Входные данные	Выходные данные
Проверка работоспособности с взаимoisключением	Array[200000]	Вывод времени в консоль
Проверка работоспособности без взаимoisключения	Array[200000]	Вывод времени в консоль

Скриншоты работы программы №3

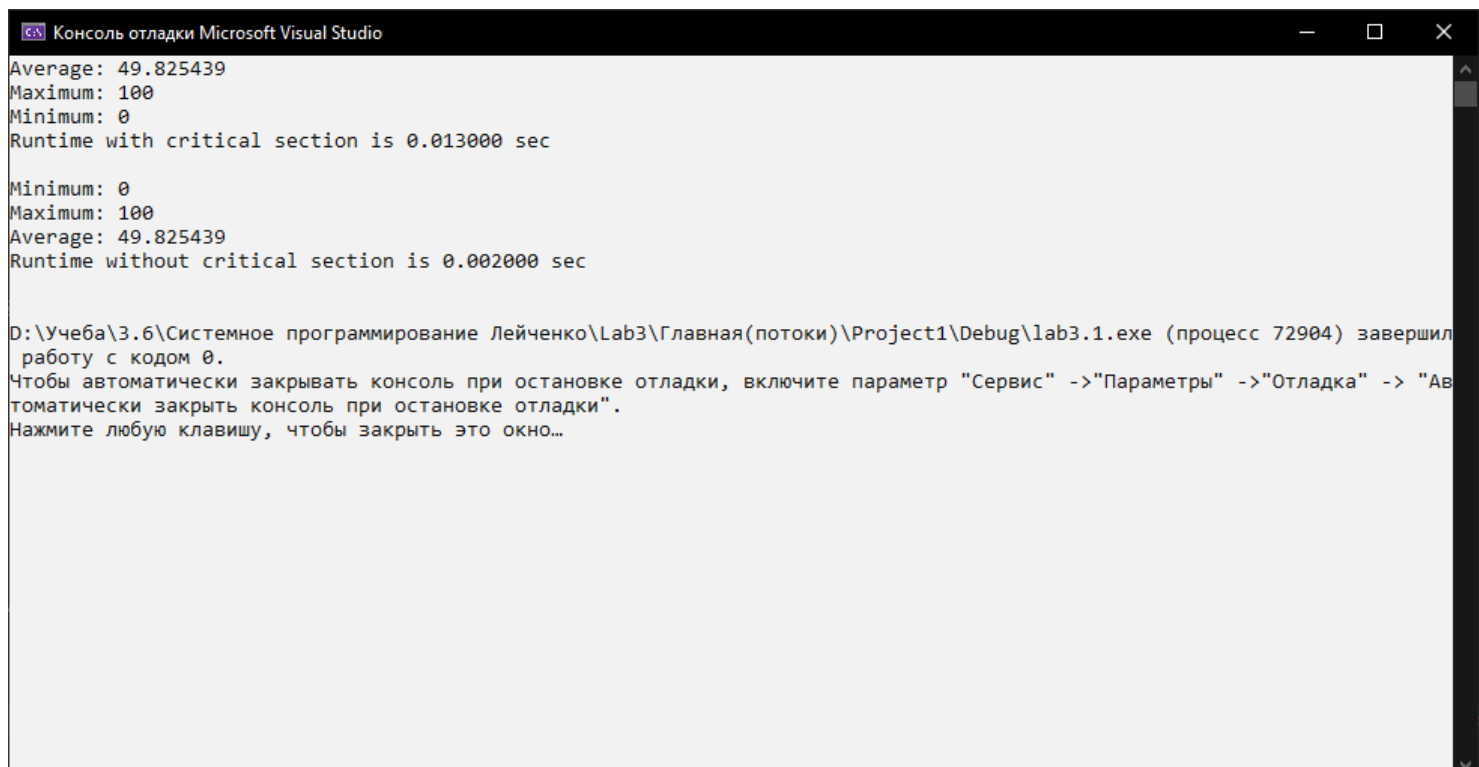


Рис. 5 – Скриншот работы программы

Выводы: в ходе выполнения данной лабораторной работы мы изучили встроенные объекты синхронизации в ОС Windows, а так же системные вызовы Win32 API для реализации алгоритмов межпоточной и межпроцессной синхронизации.