

Задание 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.

Код программы

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

int main()
{
    // name of named mutex
    TCHAR sMutexName[] { L"OneInstanceMutex" };

    // try to open mutex
    HANDLE hOneInstMutex = OpenMutex(MUTEX_ALL_ACCESS, FALSE, sMutexName);

    // if cant open mutex, than create it
    if (!hOneInstMutex)
    {
        hOneInstMutex = CreateMutex(NULL, TRUE, sMutexName);
        wprintf(L"Is Running");
        while (1) { }
    }
    else // if opening succeded than app intance already run
    {
        wprintf(L"Only 1 app intance available \n");
        Sleep(5000);
    }
}
```

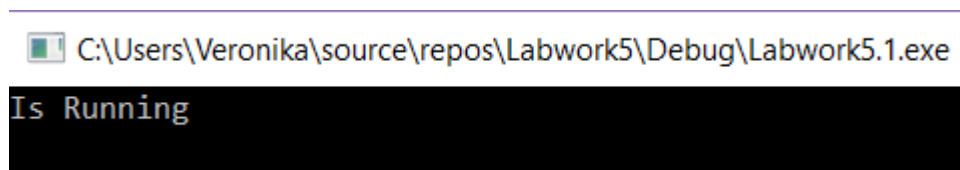


Рисунок 1 – Запуск первого экземпляра

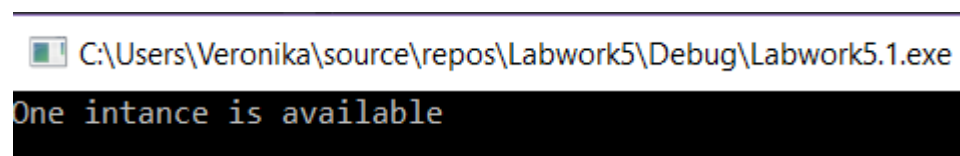


Рисунок 2 – Рисунок 1 – Запуск 2 экземпляра

Задание 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.

Код программы

```
#include <stdio.h>
#include <iostream>
#include "windows.h"
#include <ctime>
#include "tchar.h"

#define MIN_THREAD_SLEEPING 1
#define MAX_THREAD_SLEEPING 3
#define FILE_HANDLES_COUNT 3
#define BUFF_FOR_WRITING_TO_LOG_FILE_SIZE 32

// struct for contain params for new threads
typedef struct {
    DWORD id;
    DWORD timeSleep;
} NewThreaParams, *LPNewThreadParams;

// global variables
HANDLE hFilesAccessSemaphore;
CONST TCHAR ctsFileForSaving[] = L"Log.txt";

using namespace std;

// func for creating new thread
DWORD WINAPI NewThread(LPVOID param)
{
    // save params in stack
    DWORD dwThreadUserId = ((LPNewThreadParams)param)->id;
    DWORD dwThreadUserSleepingTime = ((LPNewThreadParams)param)->timeSleep;

    HANDLE hFileForSaving; // handle to file for saving info
    DWORD dwWaitingRes = ERROR_SUCCESS;
    clock_t startTime = clock(), endTime;

    // wait for resource (handle to file)
    WaitForSingleObject(hFilesAccessSemaphore, INFINITE);

    printf_s("Thread %d get resource\n", dwThreadUserId);

    printf_s("Thread %d will sleep for %d second\n", dwThreadUserId,
dwThreadUserSleepingTime);
    Sleep(dwThreadUserSleepingTime * 1000);

    // create handle to file
```

```

if ((hFileForSaving = CreateFile(ctsFileForSaving,
    GENERIC_WRITE,
    FILE_SHARE_WRITE,
    NULL,
    OPEN_ALWAYS,
    FILE_ATTRIBUTE_NORMAL,
    NULL)) == INVALID_HANDLE_VALUE)
{
    cout << "Error from thread " << dwThreadUserId << ", cant open file\n";
    ReleaseSemaphore(hFilesAccessSemaphore, 1, NULL);
    ExitThread(1);
}

// set file pointer to end
if ((dwWaitingRes = SetFilePointer(hFileForSaving, 0, NULL, FILE_END)) ==
INVALID_SET_FILE_POINTER)
{
    cout << "Error from thread " << dwThreadUserId << ", cant set a file pointer in the
file\n";
    ReleaseSemaphore(hFilesAccessSemaphore, 1, NULL);
    ExitThread(2);
}
endTime = clock();

// buff for writing to file
TCHAR buff[BUFF_FOR_WRITING_TO_LOG_FILE_SIZE]{ '\0' };
DWORD dwWrittenBytes;

// make string for saving in file
_sntprintf_s(buff, BUFF_FOR_WRITING_TO_LOG_FILE_SIZE, L"Thread %d lasted %d seconds\n",
dwThreadUserId, ((endTime - startTime) / 1000));

// write to file
if (!WriteFile(hFileForSaving, buff, _tcslen(buff) * sizeof(TCHAR), &dwWrittenBytes,
NULL))
{
    cout << "Error from thread " << dwThreadUserId << ", cant write into the file\n";
    ReleaseSemaphore(hFilesAccessSemaphore, 1, NULL);
    ExitThread(3);
}
// clos handle to file
CloseHandle(hFileForSaving);
printf_s("Thread %d complete, bytes writen to file:%d\n", dwThreadUserId,
dwWrittenBytes);

// allow other thread use resource (handle to file)
ReleaseSemaphore(hFilesAccessSemaphore, 1, NULL);
return 0;
}

void StartThreads(DWORD dwCountThreads, HANDLE* Threads, DWORD dwRandRange) {
    for (int i = 0; i < dwCountThreads; i++)
    {
        srand(clock());
        NewThreaParams param = { 0 }; // struct for containing params for new thread
        param.id = i; // set user id
        param.timeSleep = (rand() % dwRandRange) + MIN_THREAD_SLEEPING; // rand time to
sleeping
        Threads[i] = CreateThread(NULL, NULL, NewThread, (LPVOID)&param, NULL, NULL);
        if ((!Threads[i]) || (Threads[i] == INVALID_HANDLE_VALUE))
        {
            cout << "Error, cant start thread number: " << i << endl;
            exit(0);
        }
        Sleep(100); // for srand, for more differance in time between each rand()
    }
}

```

```

}
int main()
{
    HANDLE* Threads;

    // get count threads
    DWORD dwCountThreads;
    cout << "File handles count =>" << FILE_HANDLES_COUNT << endl;
    cout << "Input count of threads:";
    scanf_s("%d", &dwCountThreads);

    // alloc memory for new threads handles
    Threads = (HANDLE*)malloc(sizeof(HANDLE) * dwCountThreads);

    // create counter semaphore for control count handles to file
    hFilesAccessSemaphore = CreateSemaphore(NULL, FILE_HANDLES_COUNT, FILE_HANDLES_COUNT,
L"FilesSemaphore");
    if (!hFilesAccessSemaphore || hFilesAccessSemaphore == INVALID_HANDLE_VALUE)
    {
        cout << " Cant create semaphore\n";
        return 1;
    }

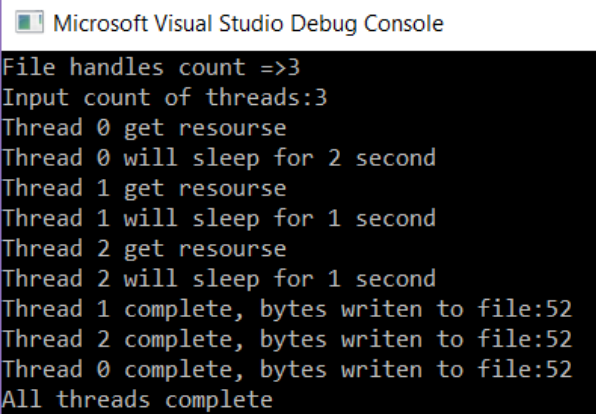
    DWORD dwRandRange = MAX_THREAD_SLEEPING - MIN_THREAD_SLEEPING + 1;
    StartThreads(dwCountThreads, Threads, dwRandRange);

    WaitForMultipleObjects(dwCountThreads, Threads, TRUE, INFINITE); // wait until all
thread completed

    // close all handles
    CloseHandle(hFilesAccessSemaphore);
    for (int i = 0; i < dwCountThreads; i++)
        CloseHandle(Threads[i]);
    free(Threads);

    CloseHandle(hFilesAccessSemaphore);
    cout << "All threads complete\n";
    getchar();
    return 0;
}

```

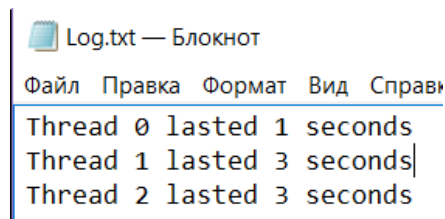


```

Microsoft Visual Studio Debug Console
File handles count =>3
Input count of threads:3
Thread 0 get resource
Thread 0 will sleep for 2 second
Thread 1 get resource
Thread 1 will sleep for 1 second
Thread 2 get resource
Thread 2 will sleep for 1 second
Thread 1 complete, bytes writen to file:52
Thread 2 complete, bytes writen to file:52
Thread 0 complete, bytes writen to file:52
All threads complete

```

Рисунок 3 – Выполнение программы



```
Log.txt — Блокнот
Файл  Правка  Формат  Вид  Справка
Thread 0 lasted 1 seconds
Thread 1 lasted 3 seconds
Thread 2 lasted 3 seconds
```

Рисунок 4 – Выполнение программы

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 взаимoisключения одновременного обращения к источнику данных (массиву).

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

Код программы

```
#include <stdio.h>
#include "iostream"
#include "windows.h"
#include <ctime>

#define COUNT_EL_ARRAY 10000000
#define ARRAY_NUM_MIN 0
#define ARRAY_NUM_MAX 100000

#define SUM_OF_ELEMENTS 0
#define MIN_MAX_VALUE 1
#define AVERAGE_VALUE 2

DWORD WINAPI WithCriticalSection(LPVOID param);
DWORD WINAPI WithoutCriticalSection(LPVOID param);
void GenerateArrayNums(int* arr); // generate numbers for array

CRITICAL_SECTION CriticalSection; // critical section
HANDLE* threadHandles; // handles of create threads
CONST DWORD COUNT_THREADS = 3; // count threads
int arrayForCalculating[COUNT_EL_ARRAY]{ 0 };

using namespace std;

LONG64 AverageValue(int* mass)
{
    LONG64 result = 0;
```

```

    for (int i = 0; i < COUNT_EL_ARRAY; i++)
    {
        result += mass[i];
    }
    result /= COUNT_EL_ARRAY;
    cout << "Avarage:" << result << endl;
    return result;
}

void MaxMinValue(int* mass)
{
    int max = mass[0];
    int min = mass[0];
    for (int r = 1; r < COUNT_EL_ARRAY; r++)
    {
        if (max < mass[r]) max = mass[r];
        if (min > mass[r]) min = mass[r];
    }
    cout << "\nMin: " << min << endl;
    cout << "Max: " << max << endl;
}

int SumOfElemets(int* mass)
{
    int sum = mass[0];
    for (int i = 1; i < COUNT_EL_ARRAY; i++)
    {
        sum += mass[i];
    }
    cout << "Sum of elements:" << sum << endl;;
    return sum;
}

int main(int argc, char* argv[])
{
    clock_t start, end;

    InitializeCriticalSection(&CriticalSection);

    threadHandles = (HANDLE*)malloc(sizeof(HANDLE) * COUNT_THREADS);

    GenereteArrayNums(arrayForCalculating); // generate array nums

    // start with using critical section
    cout << "Start function with critical section\n";
    start = clock();
    for (int i = 0; i < COUNT_THREADS; i++) // create new threads
    {
        threadHandles[i] = CreateThread(NULL, NULL, WithCriticalSection, (LPVOID)i, NULL, NULL);
    }
    WaitForMultipleObjects(COUNT_THREADS, threadHandles, TRUE, INFINITE);
    end = clock();
    cout << "Function time in critical section : " << end - start << " ms.\n";
    //cout <<
    "\n*****\n" << endl;
    // start without using critical section
    cout << "\nStart function without critical section\n";
    start = clock();
    for (int i = 0; i < COUNT_THREADS; i++) // create new threads
    {
        threadHandles[i] = CreateThread(NULL, NULL, WithoutCriticalSection, LPVOID(i),
        NULL, NULL);
    }
    // wait until all threads complete
    WaitForMultipleObjects(COUNT_THREADS, threadHandles, TRUE, INFINITE);
}

```

```

end = clock();
cout << "Function time without critical section : " << end - start << " ms.\n";

for (int i = 0; i < COUNT_THREADS; i++)
    CloseHandle(threadHandles[i]);
DeleteCriticalSection(&CriticalSection);
return 0;
}

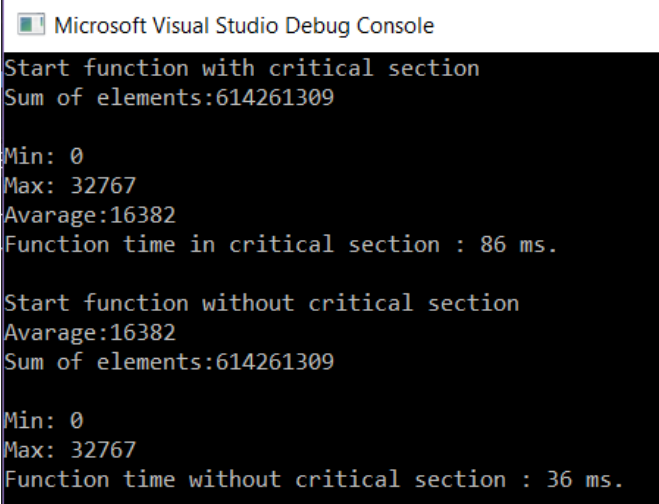
void GenerateArrayNums(int* arr)
{
    DWORD randRange = ARRAY_NUM_MAX - ARRAY_NUM_MIN;
    // array generation
    for (DWORD i = 0; i < COUNT_EL_ARRAY; i++)
    {
        arr[i] = (rand() % randRange) + ARRAY_NUM_MIN;
    }
}

// with using critical section
DWORD WINAPI WithCriticalSection(LPVOID param)
{
    switch ((int)param)
    {
        {
            case SUM_OF_ELEMENTS:
            {
                EnterCriticalSection(&CriticalSection);
                SumOfElemets(arrayForCalculating);
                LeaveCriticalSection(&CriticalSection);
            } break;
            case MIN_MAX_VALUE:
            {
                EnterCriticalSection(&CriticalSection);
                MaxMinValue(arrayForCalculating);
                LeaveCriticalSection(&CriticalSection);
            } break;
            case AVERAGE_VALUE:
            {
                EnterCriticalSection(&CriticalSection);
                AverageValue(arrayForCalculating);
                LeaveCriticalSection(&CriticalSection);
            } break;
            default:
                break;
        }
    }
    return 0;
}

// without using critical section
DWORD WINAPI WithoutCriticalSection(LPVOID param)
{
    switch ((int)param)
    {
        {
            case SUM_OF_ELEMENTS:
            {
                SumOfElemets(arrayForCalculating);
            } break;
            case MIN_MAX_VALUE:
            {
                MaxMinValue(arrayForCalculating);
            } break;
            case AVERAGE_VALUE:
            {
                AverageValue(arrayForCalculating);
            } break;
            default:
                break;
        }
    }
}

```

```
        break;  
    }  
    return 0;  
}
```



```
Microsoft Visual Studio Debug Console  
Start function with critical section  
Sum of elements:614261309  
  
Min: 0  
Max: 32767  
Avarage:16382  
Function time in critical section : 86 ms.  
  
Start function without critical section  
Avarage:16382  
Sum of elements:614261309  
  
Min: 0  
Max: 32767  
Function time without critical section : 36 ms.
```

Рисунок 5 – Выполнение программы