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 программування»

**«Изучение системных вызовов Win32 API работы объектов синхронизации и алгоритмов межпоточной**

**и межпроцессной синхронизации в ОС Windows»**

ХАІ.503.535а.20В. 123,1805038

Виконав студент гр. 535а

(№ групи)

Мартиненко Антон Олександрович

(П.І.Б.)

(підпис, дата)

Перевірив асистент

(науковий ступінь, вчене звання, посада)

Лейченко К.М.

(підпис, дата) (П.І.Б.)

2021

**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 критических операторам (обращение к массиву) должен быть реализован в виде взаимоисключения одновременного обращения к источнику данных (массиву).

Задача: программа должна иметь 2 режима работы: с взаимоисключением 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 rinning! |

**Скриншоты работы программы №1**

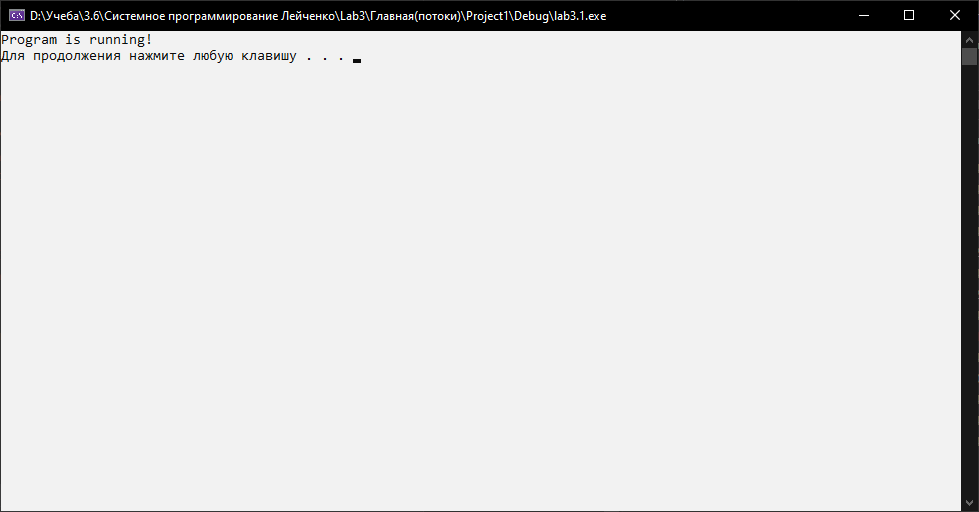
****

Рис. 1 – Скриншот работы программы

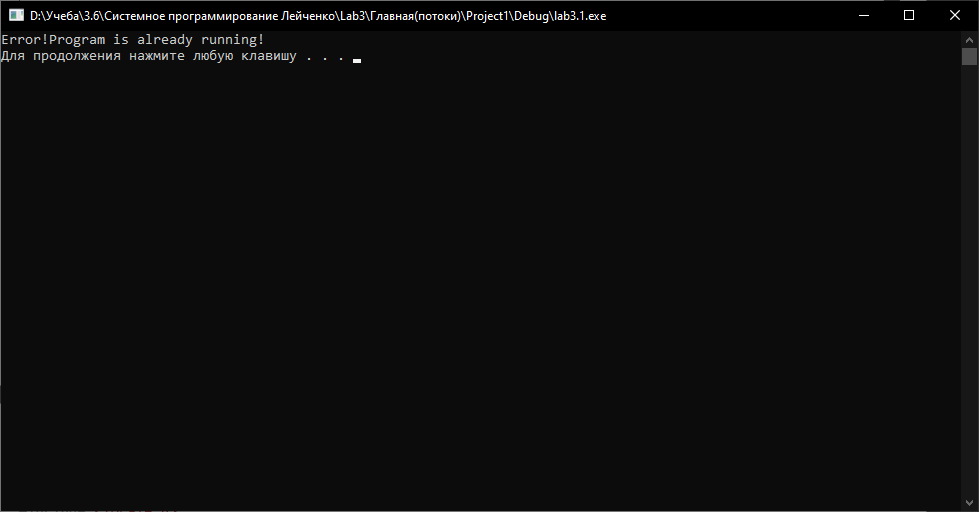
****

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**

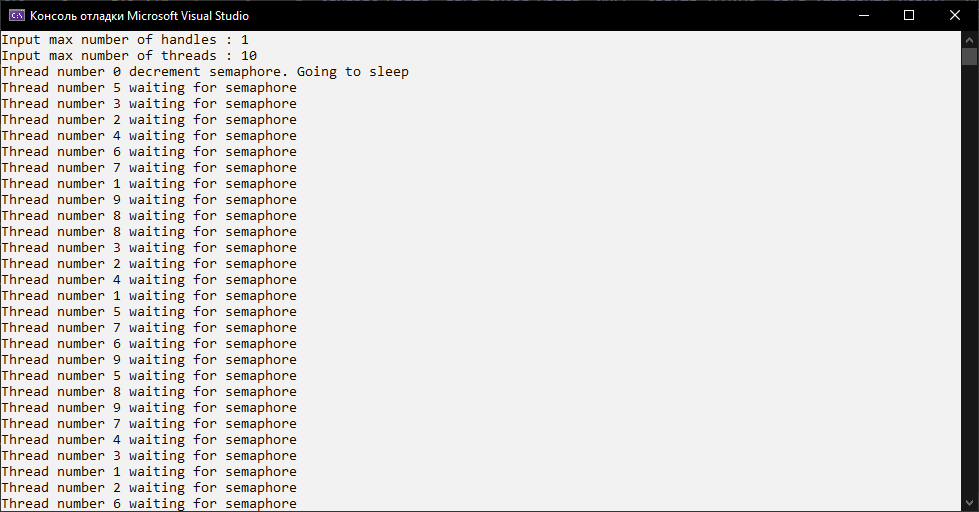
****

Рис. 3 – Скриншот работы программы

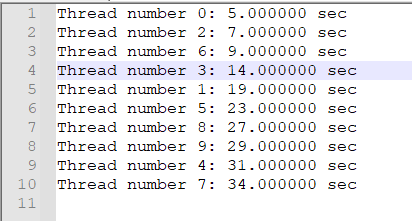
****

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 взаимоисключением | Array[200000] | Вывод времени в консоль |
| Проверка работоспособности без взаимоисключения | Array[200000] | Вывод времени в консоль |

**Скриншоты работы программы №3**

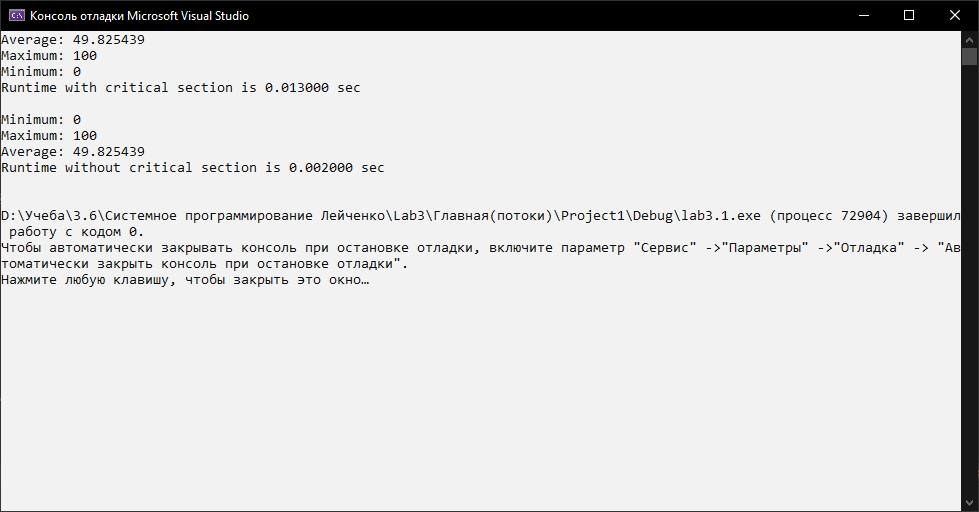
****

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.