

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний аерокосмічний університет ім. М. Є. Жуковського
«Харківський авіаційний інститут»

Факультет радіоелектроніки, комп'ютерних систем та інфокомунікацій

Кафедра комп'ютерних систем, мереж і кібербезпеки

Лабораторна робота

з Системного програмування
(назва дисципліни)

на тему: «Вивчення вбудованих об'єктів синхронізації в ОС Windows»

Виконала: студентка 3-го курсу групи №525ст2
напряму підготовки (спеціальності)
123-«Комп'ютерна інженерія»

(шифр і назва напряму підготовки (спеціальності))

Коваленко Я.О.

(прізвище й ініціали студента)

Прийняв: асистент каф.503

Мозговий М.В.

(посада, науковий ступінь, прізвище й ініціали)

Національна шкала: _____

Кількість балів: _____

Оцінка: ECTS _____

Цель работы:

Изучение встроенных объектов синхронизации в ОС Windows. 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.

Ход работы:

Код программы:

```
#include <stdio.h>
#include <Windows.h>

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

    if (WaitForSingleObject(mutex, 0) == WAIT_OBJECT_0)
    {
        printf("Program started! Press any key to stop!");
        getchar();
        ReleaseMutex(mutex);
    }
    else
    {
        printf("Can't start program!");
        getchar();
    }

    CloseHandle(mutex);
    return 0;
}

#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <Windows.h>
#include <time.h>

HANDLE semaphore;

int random_int(int min, int max);
DWORD WINAPI thread_function(LPVOID param);

int main()
{
    int max_handles;
    int max_threads;
    srand(time(NULL));

    printf("Input max number of handles\n");
    printf(">>");
    scanf("%i", &max_handles);

    printf("Input max number of threads\n");
    printf(">>");
```

```

scanf("%i", &max_threads);

HANDLE* threads = new HANDLE[max_threads];
semaphore = CreateSemaphoreA(NULL, max_handles, max_handles, "MySemaphore");
if (semaphore == NULL)
    return 1;

HANDLE file = CreateFileA("result.txt", GENERIC_WRITE, FILE_SHARE_WRITE, NULL,
CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL);
if (file == INVALID_HANDLE_VALUE)
{
    CloseHandle(file);
    return 1;
}
CloseHandle(file);

for (int i = 0; i < max_threads; i++)
{
    int* params = new int[2];
    params[0] = i;
    params[1] = random_int(1, 5);

    threads[i] = CreateThread(NULL, 0, thread_function, (LPVOID)params, NULL,
NULL);
}
WaitForMultipleObjects(max_threads, threads, TRUE, INFINITE);

return 0;
}

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 %i waiting for semaphore\n", thread_number);
    }

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

    Sleep(params[1] * 1000);

    HANDLE file = CreateFileA("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 %i made this in %f seconds\n0", thread_number, time_elapsed);
WriteFile(file, str, strlen(str), NULL, NULL);
CloseHandle(file);

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

return 0;
}

int random_int(int min, int max)
{
    return min + rand() % (max + 1 - min);
}
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <Windows.h>
#include <time.h>

#define ARRAY_MAX 50000000

CRITICAL_SECTION section;
int* array;

DWORD WINAPI thread_function_min(LPVOID use_critical_section);
DWORD WINAPI thread_function_max(LPVOID use_critical_section);
DWORD WINAPI thread_function_avg(LPVOID use_critical_section);
void generate_array(int* array);
int random_int(int min, int max);

int main()
{
    HANDLE* threads;
    clock_t start;
    float elapsed_time;

    srand(time(NULL));
    InitializeCriticalSection(&section);

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

    start = clock();

```

```

threads = new HANDLE[3];
threads[0] = CreateThread(NULL, 0, thread_function_min, (LPVOID)TRUE, NULL, NULL);
threads[1] = CreateThread(NULL, 0, thread_function_avg, (LPVOID)TRUE, NULL, NULL);
threads[2] = CreateThread(NULL, 0, thread_function_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("With critical section it took %f seconds\n\n", elapsed_time);

start = clock();
threads = new HANDLE[3];
threads[0] = CreateThread(NULL, 0, thread_function_min, (LPVOID)FALSE, NULL, NULL);
threads[1] = CreateThread(NULL, 0, thread_function_avg, (LPVOID)FALSE, NULL, NULL);
threads[2] = CreateThread(NULL, 0, thread_function_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("Without critical section it took %f seconds\n\n", elapsed_time);

DeleteCriticalSection(&section);
}

DWORD WINAPI thread_function_min(LPVOID use_critical_section)
{
    if ((bool)use_critical_section)
    {
        while (!TryEnterCriticalSection(&section))
        {
            //Nothing. Waiting until critical section free.
        }
    }

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

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

    return 0;
}

```

```

DWORD WINAPI thread_function_max(LPVOID use_critical_section)
{
    if ((bool)use_critical_section)
    {
        while (!TryEnterCriticalSection(&section))
        {
            //Nothing. Waiting until critical section free.
        }
    }

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

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

    return 0;
}

DWORD WINAPI thread_function_avg(LPVOID use_critical_section)
{
    if ((bool)use_critical_section)
    {
        while (!TryEnterCriticalSection(&section))
        {
            //Nothing. Waiting until critical section free.
        }
    }

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

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

    return 0;
}

```

```

void generate_array(int* array)
{
for (int i = 0; i < ARRAY_MAX; i++)
{
    array[i] = random_int(0, 500);
}
}

```

```

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

```

Результат работы:

```

Program started! Press any key to stop!

```

```

Program started! Press any key to stop!

E:\SP\lab4\lab4.1\x64\Debug\lab4.1.exe (process 4652) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .

```

```

>>5
Input max number of threads
>>5
Thread 1 decrement semaphore. Going to sleep
Thread 0 decrement semaphore. Going to sleep
Thread 3 decrement semaphore. Going to sleep
Thread 2 decrement semaphore. Going to sleep
Thread 4 decrement semaphore. Going to sleep
Thread 1 released semaphore.
Thread 4 released semaphore.
Thread 2 released semaphore.
Thread 3 released semaphore.
Thread 0 released semaphore.

E:\SP\lab4\lab4.2\Debug\lab4.2.exe (process 3684) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .

```



```
Min: 0
Avg: 171.798691
Max: 500
With critical section it took 2.172000 seconds

Max: 500
Min: 0
Avg: 171.798691
Without critical section it took 0.260000 seconds

E:\SP\lab4\lab4.3\Debug\lab4.3.exe (process 820) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Opt
ions->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

Выводы:

В результате выполнения данной лабораторной работы были изучены встроенные объекты синхронизации в ОС Windows; системных вызовов Win32 API для реализации алгоритмов межпоточной и межпроцессной синхронизации.