Міністерство освіти і науки, молоді та спорту України

Національний аерокосмічний університет ім. М.Є. Жуковського

Кафедра 503

Лабораторна робота № 1

з дисципліни

«Системне програмування»

Тема: «Изучение встроенных объектов синхронизации в ОС Windows.

Изучение системных вызовов Win32 API для реализации алгоритмов межпоточной и межпроцессной синхронизации..»

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | Виконав: | ст. гр. 535Б |
|  |  |  | Жовнір В.Е |
|  |  | Перевірив: | асистент каф. 503 |
|  |  |  | Мозговий М.В. |

#### 

#### Харків 2020

### 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.

1. **Текст программы**

#include "stdafx.h"

int main()

{

HANDLE mutex = CreateMutex(NULL, FALSE, TEXT("UNIQUE\_MUTEX"));

if (WaitForSingleObject(mutex, 0) == WAIT\_OBJECT\_0) {

\_tprintf(TEXT("Running..."));

}

else {

\_tprintf(TEXT("Program is already running. Fatal Error."));

}

getchar();

return 0;

}

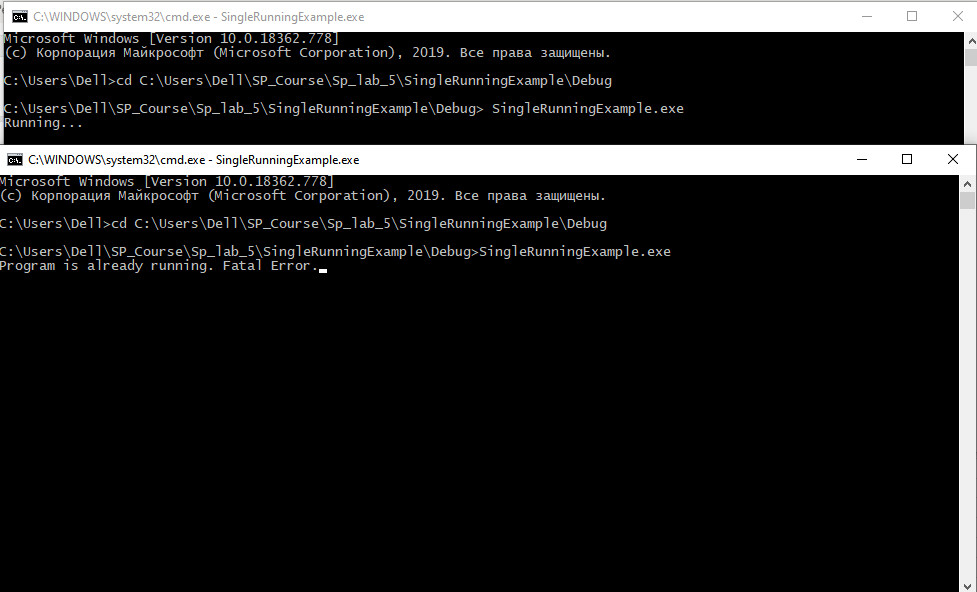
****

Рисунок 1 – результат работы программы для задания 1

**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.

1. **Текст программы**

// LimitedFileAccess.cpp : Defines the entry point for the console application.

//

#include "stdafx.h"

#define SLEEPING\_TIME\_MIN 1000

#define SLEEPING\_TIME\_MAX 3000

#define WAITING\_TIME\_MIN 1000

#define WAITING\_TIME\_MAX 3000

#define FILE\_NAME TEXT("FileToCapture.txt")

#define MAX\_BUF 50

#define TIME\_OUT 1000

HANDLE sem;

int GetRand(int min, int max) {

return ((int)rand()) % (max - min) + min;

}

HANDLE OpenRequiredFile() {

WaitForSingleObject(sem, INFINITE);

HANDLE hFile = CreateFile(

FILE\_NAME, GENERIC\_WRITE, FILE\_SHARE\_WRITE,

NULL, OPEN\_EXISTING, FILE\_ATTRIBUTE\_NORMAL, 0);

SetFilePointer(hFile, 0, NULL, FILE\_END);

return hFile;

}

void CloseFile(HANDLE hFile) {

CloseHandle(hFile);

DWORD count;

ReleaseSemaphore(sem, 1, (LPLONG)&count);

}

void ThreadStart() {

DWORD currId = GetCurrentThreadId(), realC;

srand((unsigned int)currId);

//elapse time

DWORD start = GetTickCount();

//sleep

DWORD waitTime = GetRand(SLEEPING\_TIME\_MIN, SLEEPING\_TIME\_MAX);

Sleep(waitTime);

//open and write

HANDLE hFile = OpenRequiredFile();

LPSTR buf = new CHAR[MAX\_BUF];

memset(buf, 0, MAX\_BUF);

realC = sprintf(buf, "ID: %d, waiting time: %d", currId, GetTickCount() - start);

buf[realC] = '\n';

printf("%s", buf);

WriteFile(hFile, (LPVOID)buf, realC+1, &realC, NULL);

//wait

waitTime = GetRand(WAITING\_TIME\_MIN, WAITING\_TIME\_MAX);

Sleep(waitTime);

//close

delete[] buf;

CloseFile(hFile);

}

int \_tmain(INT argc, TCHAR\*\* argv)

{

if (argc != 3) {

\_tprintf(TEXT("Wrong number of args"));

}

INT nAvailablePointers = \_ttoi(argv[1]);

INT nThreads = \_ttoi(argv[2]);

//make semaphore

sem = CreateSemaphore(

NULL, nAvailablePointers, nAvailablePointers, NULL);

//launch the threads

HANDLE\* threads = new HANDLE[nThreads];

for (size\_t i = 0; i < nThreads; i++)

{

DWORD id;

threads[i] = CreateThread(

NULL, 0, (LPTHREAD\_START\_ROUTINE)ThreadStart, NULL, 0, &id);

}

for (size\_t i = 0; i < nThreads; i++)

WaitForSingleObject(threads[i], INFINITE);

delete[] threads;

return 0;

}

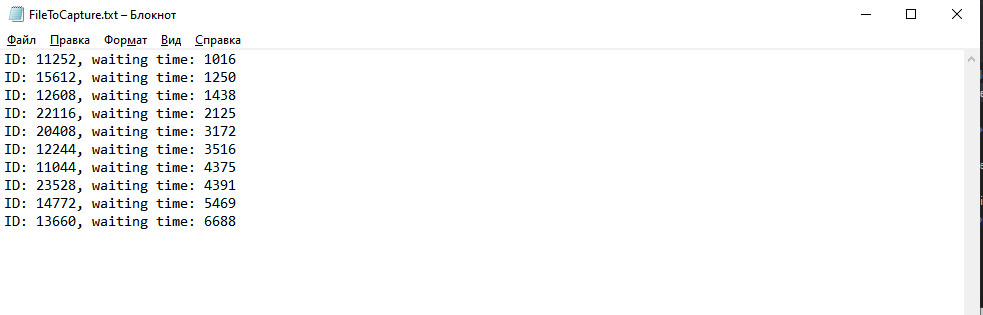
****

Рисунок 2 – результат работы программы для задания 2

**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.

#include "Header.h"

using namespace std;

int main(int argc, char\* argv[])

{

clock\_t start, end;

InitializeCriticalSection(&gCriticalSection);

// alloc memory for new threads handles

threadHandles = (HANDLE\*)malloc(sizeof(HANDLE) \* COUNT\_THREADS);

GenereteArrayNums(arrayForCalculating); // generate array nums

// start with using critical section --------------------------------------------------------------- first mode

cout << " \*\*\* Start calculating with critical section \*\*\*\n";

start = clock();

for (int i = 0; i < COUNT\_THREADS; i++) // create new threads

{

threadHandles[i] = CreateThread(NULL,

NULL,

StartNewThreadWithCriticalSection,

(LPVOID)i,

NULL,

NULL);

}

// wait until all threads complete

WaitForMultipleObjects(COUNT\_THREADS, threadHandles, TRUE, INFINITE);

end = clock();

cout << "Calculating with critical section continued for " << end - start << " ms.\n";

cout << "--------------------------------------------------------------------\n";

// start without using critical section ------------------------------------------------------------- second mode

cout << " \*\*\* Start calculating without critical section \*\*\*\n";

start = clock();

for (int i = 0; i < COUNT\_THREADS; i++) // create new threads

{

threadHandles[i] = CreateThread(NULL,

NULL,

StartNewThreadWithoutCriticalSection,

LPVOID(i),

NULL,

NULL);

}

// wait until all threads complete

WaitForMultipleObjects(COUNT\_THREADS, threadHandles, TRUE, INFINITE);

end = clock();

cout << "Calculating without critical section continued for " << end - start << " ms.\n";

for (int i = 0; i < COUNT\_THREADS; i++)

CloseHandle(threadHandles[i]);

DeleteCriticalSection(&gCriticalSection);

return 0;

}

// calculating with using critical section

DWORD WINAPI StartNewThreadWithCriticalSection(LPVOID param)

{

switch ((int)param)

{

case SUM\_OPERATION:

{

EnterCriticalSection(&gCriticalSection);

ArrNumsSum(arrayForCalculating);

LeaveCriticalSection(&gCriticalSection);

} break;

case BUBBLE\_SORT\_OPERATION:

{

EnterCriticalSection(&gCriticalSection);

BubbleSort(arrayForCalculating);

LeaveCriticalSection(&gCriticalSection);

} break;

case AVARAGE\_OPERATION:

{

EnterCriticalSection(&gCriticalSection);

CalcAvarageValue(arrayForCalculating);

LeaveCriticalSection(&gCriticalSection);

} break;

default:

break;

}

return 0;

}

// calculating without using critical section

DWORD WINAPI StartNewThreadWithoutCriticalSection(LPVOID param)

{

switch ((int)param)

{

case SUM\_OPERATION:

{

ArrNumsSum(arrayForCalculating);

} break;

case BUBBLE\_SORT\_OPERATION:

{

BubbleSort(arrayForCalculating);

} break;

case AVARAGE\_OPERATION:

{

CalcAvarageValue(arrayForCalculating);

} break;

default:

break;

}

return 0;

}

// generate array in setted range

void GenereteArrayNums(int\* arr)

{

DWORD randRange = ARRAY\_NUM\_MAX\_BOUNDARY - ARRAY\_NUM\_MIN\_BOUNDARY;

// array generation

for (DWORD i = 0; i < ARRAY\_SIZE; i++)

{

arr[i] = (rand() % randRange) + ARRAY\_NUM\_MIN\_BOUNDARY;

}

}

// Calculate avarage value of array

LONG64 CalcAvarageValue(int\* arr)

{

LONG64 res = 0;

for (int i = 0; i < ARRAY\_SIZE; i++)

{

res += arr[i];

}

res /= ARRAY\_SIZE;

cout << "Avarage:" << res << endl;

return res;

}

// array sort using Bubble sort algorithm

void BubbleSort(int\* arr)

{

for (int i = 0; i < ARRAY\_SIZE; i++)

{

for (int j = 0; i < ARRAY\_SIZE - i - 1; i++)

{

if (arr[j] > arr[j + 1]) swap(arr[j], arr[j]); // fake swap elements for preserving the initial state

// of the array for fair comparison execution with

// and without critical section

}

}

cout << "First:" << arr[0] << ";last:" << arr[ARRAY\_SIZE - 1] << endl;

return;

}

// calc array nums sum

int ArrNumsSum(int\* arr)

{

int sum = arr[0];

for (int i = 1; i < ARRAY\_SIZE; i++)

{

sum += arr[i];

}

cout << "Sum:" << sum << endl;;

return sum;

}

**Текст программы файла Header.h**

#include <stdio.h>

#include "iostream"

#include "windows.h"

#include <ctime>

#define ARRAY\_SIZE 10000000 // array size which will be created for calculating, 10kk

#define ARRAY\_NUM\_MIN\_BOUNDARY 0 // low bound for rand array nums

#define ARRAY\_NUM\_MAX\_BOUNDARY 100000 // high bound for rand arrray nums

#define SUM\_OPERATION 0

#define BUBBLE\_SORT\_OPERATION 1

#define AVARAGE\_OPERATION 2

////////////////// functions prototypes

DWORD WINAPI StartNewThreadWithCriticalSection(LPVOID param);

DWORD WINAPI StartNewThreadWithoutCriticalSection(LPVOID param);

void GenereteArrayNums(int\* arr); // generate numbers for array

int ArrNumsSum(int\* arr); // calculate array numbers sum

void BubbleSort(int\* arr);

LONG64 CalcAvarageValue(int\* arr);

/////////////////// global variables

CRITICAL\_SECTION gCriticalSection; // critical section

HANDLE\* threadHandles; // handles of creat threads

CONST DWORD COUNT\_THREADS = 3; // count threads

int arrayForCalculating[ARRAY\_SIZE]{ 0 }; // array for calculating

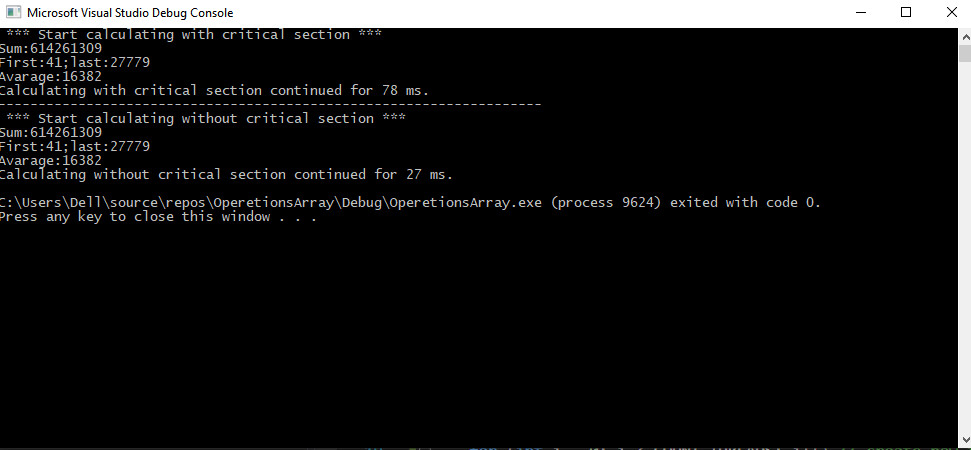
****

Рисунок 3 – результат работы программы для задания 3