**Санкт-Петербургский национальный исследовательский университет**

**информационных технологий, механики и оптики**

**Кафедра информатики и прикладной математикаи**

Лабораторная работа 5

Объекты синхронизации

Выполнил: Гхази Даниэль

Группа P3218

Преподаватель: Зыков А.Г.

2017 г.

**Задание**

Исследование на конкретном примере следующих объектов синхронизации:

1. критические секции

2. мьютексы

3. семафоры

4. события

Задачу для синхронизации выбрать на свое усмотрение.

Задачи для каждого метода синхронизации должны быть различными. Задачи должны наглядно демонстрировать выбранный метод синхронизации и учитывать его особенности. Студент, сдающий работу должен аргументированно обосновать задачу, выбранную для синхронизации и метод синхронизации.

**Код программы**

* Критическая секция

#include <windows.h>

#include <iostream>

#include <string>

#include <process.h>

#define MAX\_ARRAY 13

CRITICAL\_SECTION critsect;

int array[MAX\_ARRAY];

void EmptyArray(void \*);

void PrintArray(void \*);

void FillArray(void \*);

using namespace std;

int main()

{

InitializeCriticalSection(&critsect);

if (\_beginthread(FillArray,1024,NULL)==-1)

cout << "Error begin thread " << endl;

if (\_beginthread(PrintArray,1024,NULL)==-1)

cout << "Error begin thread " << endl;

if (\_beginthread(EmptyArray,1024,NULL)==-1)

cout << "Error begin thread " << endl;

if (\_beginthread(PrintArray,1024,NULL)==-1)

cout << "Error begin thread " << endl;

if (\_beginthread(FillArray,1024,NULL)==-1)

cout << "Error begin thread " << endl;

if (\_beginthread(PrintArray,1024,NULL)==-1)

cout << "Error begin thread " << endl;

Sleep(5000);

system("Pause");

return 0;

}

void EmptyArray(void \*)

{

EnterCriticalSection(&critsect);

cout << "EmptyArray" << endl;

for (int x=0;x<(MAX\_ARRAY+1); x++) array[x]=0;

Sleep(1000);

LeaveCriticalSection(&critsect);

\_endthread();

}

void PrintArray(void \*)

{

EnterCriticalSection(&critsect);

cout << "PrintArray" << endl;

for (int x=0;x<(MAX\_ARRAY+1); x++) cout << array[x] << " ";

cout << endl;

Sleep(1000);

LeaveCriticalSection(&critsect);

\_endthread();

}

void FillArray(void \*)

{

EnterCriticalSection(&critsect);

cout << "FillArray" << endl;

for (int x=0;x<(MAX\_ARRAY+1); x++) array[x]=x;

Sleep(1000);

LeaveCriticalSection(&critsect);

\_endthread();

}

* Событие

#include <stdio.h>

#include <iostream>

#include <fstream>

#include <windows.h>

#include <string>

#include <process.h>

using namespace std;

HANDLE event;

void Test1(void \*);

void Test2(void \*);

void Test3(void \*);

int main()

{

event=CreateEvent(NULL,true,false,"FirstStep");

if (\_beginthread(Test1,1024,NULL)==-1)

cout << "Error begin thread " << endl;

if (\_beginthread(Test2,1024,NULL)==-1)

cout << "Error begin thread " << endl;

if (\_beginthread(Test3,1024,NULL)==-1)

cout << "Error begin thread " << endl;

if (event!=NULL){

Sleep(10);

SetEvent(event);

cout << "Set event " << endl;

Sleep(1000);

ResetEvent(event);

cout << "Reset event " << endl;

CloseHandle(event);

} else {

cout << "error create event" << endl;

}

getchar();

return 0;

}

void Test1(void \*)

{

DWORD dwWaitResult=3;

while(dwWaitResult!=WAIT\_OBJECT\_0) {

dwWaitResult = WaitForSingleObject(event,1);

cout << "Test 1 TIMEOUT" << endl;

}

cout << "Event Test 1 " << endl;

\_endthread();

}

void Test2(void \*)

{

DWORD dwWaitResult=3;

while(dwWaitResult!=WAIT\_OBJECT\_0){

dwWaitResult = WaitForSingleObject(event,1);

cout << "Test 2 TIMEOUT" << endl;

}

cout << "Event Test 2 " << endl;

\_endthread();

}

void Test3(void \*)

{

DWORD dwWaitResult=3;

while(dwWaitResult!=WAIT\_OBJECT\_0){

dwWaitResult = WaitForSingleObject(event,1);

cout << "Test 3 TIMEOUT" << endl;

}

cout << "Event Test 3 " << endl;

\_endthread();

}

* Мьютекс

#include <stdio.h>

#include <iostream>

#include <fstream>

#include <windows.h>

#include <string>

#define FILE\_1 "file1"

#define FILE\_2 "file2"

#define FILE\_3 "file3"

#define THREADS\_NUMBER 3

using namespace std;

HANDLE hMutex;

DWORD WINAPI Output\_without\_mutex(CONST LPVOID path)

{

char buf[50];

ifstream ifs((char\*)path);

while (!ifs.eof())

{

ifs.getline(buf, 256);

cout << buf<< endl;

}

ifs.close();

}

DWORD WINAPI Output\_with\_mutex(CONST LPVOID path)

{

WaitForSingleObject(hMutex, INFINITE);

char buf[50];

ifstream ifs((char\*)path);

while (!ifs.eof())

{

ifs.getline(buf, 256);

cout << buf<< endl;

}

ifs.close();

ReleaseMutex(hMutex);

cout << "Mutex Released"<< endl;

}

void Output(int i)

{

LPTHREAD\_START\_ROUTINE lpstart=NULL;

if (i==1)

{

lpstart=&Output\_without\_mutex;

}else

{

lpstart = &Output\_with\_mutex;

hMutex = CreateMutex(NULL, false, NULL);

cout << "Mutex Opened\n"<< endl;

}

char \*Paths[3]={"file1.txt", "file2.txt", "file3.txt"};

HANDLE hThreads[THREADS\_NUMBER];

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

{

hThreads[i]=CreateThread(NULL, 0, lpstart, Paths[i],0, NULL);

}

WaitForMultipleObjects(THREADS\_NUMBER, hThreads, TRUE, 1000\*15);

}

int main()

{

int i;

cout << "1) Output without mutex\n2) Output with mutex\n";

cin >> i;

switch(i)

{

case 1:

{

Output(i);

break;

}

case 2:

{

Output(i);

break;

}

}

cin >> i;

return 0;

}

* Cемафор

#include <windows.h>

#include <stdio.h>

#define MAX\_SEM\_COUNT 3

#define THREADCOUNT 15

CRITICAL\_SECTION critsect;

HANDLE ghSemaphore;

HANDLE hMutex;

DWORD WINAPI ThreadProc( LPVOID );

int main( void )

{

//InitializeCriticalSection(&critsect);

HANDLE aThread[THREADCOUNT];

DWORD ThreadID;

int i;

// Create a semaphore with initial and max counts of MAX\_SEM\_COUNT

ghSemaphore = CreateSemaphore(

NULL, // default security attributes

MAX\_SEM\_COUNT, // initial count

MAX\_SEM\_COUNT, // maximum count

NULL); // unnamed semaphore

if (ghSemaphore == NULL)

{

printf("CreateSemaphore error: %d\n", GetLastError());

return 1;

}

// Create worker threads

hMutex = CreateMutex(NULL,false, NULL);

for( i=0; i < THREADCOUNT; i++ )

{

aThread[i] = CreateThread(

NULL, // default security attributes

0, // default stack size

(LPTHREAD\_START\_ROUTINE) ThreadProc,

NULL, // no thread function arguments

0, // default creation flags

&ThreadID); // receive thread identifier

if( aThread[i] == NULL )

{

printf("CreateThread error: %d\n", GetLastError());

getchar();

return 1;

}

}

// Wait for all threads to terminate

WaitForMultipleObjects(THREADCOUNT, aThread, TRUE, INFINITE);

// Close thread and semaphore handles

for( i=0; i < THREADCOUNT; i++ )

CloseHandle(aThread[i]);

CloseHandle(ghSemaphore);

system("pause");

return 0;

}

DWORD WINAPI ThreadProc( LPVOID lpParam )

{

//WaitForSingleObject(hMutex, INFINITE);

// lpParam not used in this example

UNREFERENCED\_PARAMETER(lpParam);

DWORD dwWaitResult;

BOOL bContinue=TRUE;

while(bContinue)

{

// Try to enter the semaphore gate.

dwWaitResult = WaitForSingleObject(

ghSemaphore, // handle to semaphore

0L); // zero-second time-out interval

switch (dwWaitResult)

{

// The semaphore object was signaled.

case WAIT\_OBJECT\_0:

// TODO: Perform task

WaitForSingleObject(hMutex, INFINITE);

printf("Thread %d: wait succeeded\n", GetCurrentThreadId());

ReleaseMutex(hMutex);

bContinue=FALSE;

// Simulate thread spending time on task

Sleep(5);

// Release the semaphore when task is finished

//EnterCriticalSection(&critsect);

if (!ReleaseSemaphore(

ghSemaphore, // handle to semaphore

1, // increase count by one

NULL) ) // not interested in previous count

{

printf("ReleaseSemaphore error: %d\n", GetLastError());

}

// else

// {

// printf("Semaphore released in by thread %d\n", GetCurrentThreadId());

// }

//LeaveCriticalSection(&critsect);

break;

// The semaphore was nonsignaled, so a time-out occurred.

case WAIT\_TIMEOUT:

WaitForSingleObject(hMutex, INFINITE);

printf("Thread %d: wait timed out\n", GetCurrentThreadId());

ReleaseMutex(hMutex);

break;

}

}

// ReleaseMutex(hMutex);

return TRUE;

}

**Вывод**

В ходе выполнения лабораторной работы были применены на практике знания об объектах синхронизации, области применения каждого из них и предоставляемых преимуществах. Безусловно, приведенная версия программы не может найти практического применения в том состоянии, в котором находится на момент сдачи лабораторной работы, однако впоследствии может быть значительно расширена при необходимости, на данный же момент она является хорошим наглядным примером использования объектов синхронизации, демонстрируя те случаи, в которых без них просто не обойтись.