Университет ИТМО

Лабораторная работа №6

Дисциплина: Системное программное обеспечение

Кафедра: Информатики и Прикладной Математики

Студентка: Тихонова Екатерина

Группа: 2125

Санкт-Петербург

2015 год

#include "stdafx.h"

#include "windows.h"

#include "process.h"

#define MAX\_ARRAY 5

using namespace std;

CRITICAL\_SECTION critsect;

int array[MAX\_ARRAY];

void EmptyArray(void \*);

void PrintArray(void \*);

void FullArray(void \*);

void main()

{

InitializeCriticalSection(&critsect);

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

printf("error begin thread \n");

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

printf("error begin thread \n");

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

printf("error begin thread \n");

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

printf("error begin thread \n");

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

printf("error begin thread \n");

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

printf("error begin thread \n");

Sleep(10000);

}

void EmptyArray(void \*)

{

printf("emptyArray\n");

EnterCriticalSection(&critsect);

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

Sleep(1000);

LeaveCriticalSection(&critsect);

\_endthread();

}

void PrintArray(void \*)

{

printf("PrintArray\n");

EnterCriticalSection(&critsect);

for (int x = 0; x < (MAX\_ARRAY + 1); x++) printf("%d \n", array[x]);

Sleep(1000);

LeaveCriticalSection(&critsect);

\_endthread();

}

void FullArray(void \*)

{

printf("FullArray\n");

EnterCriticalSection(&critsect);

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

Sleep(1000);

LeaveCriticalSection(&critsect);

\_endthread();

}

#include "stdafx.h"

#include <windows.h>

#include <stdio.h>

#define THREADCOUNT 4

HANDLE hWriteEvent;

HANDLE hThreads[THREADCOUNT];

DWORD WINAPI MultiplyNumber(LPVOID);

int number = 1;

int counter = 0;

void CreateEventAndThreads(void)

{

DWORD dwThreadID;

hWriteEvent = CreateEvent(

NULL,

TRUE,

TRUE,

TEXT("Multiplication")

);

if (hWriteEvent == NULL)

{

printf("CreateEvent failed (%d)\n", GetLastError());

return;

}

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

{

hThreads[i] = CreateThread(

NULL,

0,

MultiplyNumber,

NULL,

0,

&dwThreadID);

if (hThreads[i] == NULL)

{

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

return;

}

}

}

void CloseEvents()

{

CloseHandle(hWriteEvent);

}

DWORD WINAPI MultiplyNumber(LPVOID lpParam)

{

UNREFERENCED\_PARAMETER(lpParam);

DWORD dwWaitResult;

printf("Thread with id %d wait event...\n", GetCurrentThreadId());

dwWaitResult = WaitForSingleObject(

hWriteEvent,

INFINITE);

counter++;

switch (dwWaitResult)

{

case WAIT\_OBJECT\_0:

printf("Thread with id= %d and number %d Multiplies the number of %d to %d\n",

GetCurrentThreadId(), counter, number, counter);

number = number\*counter;

printf("result: %d\n", number);

break;

default:

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

return 0;

}

printf("Thread %d exiting\n", GetCurrentThreadId());

return 1;

}

int main(void)

{

DWORD dwWaitResult;

CreateEventAndThreads();

printf("begin work:\n");

dwWaitResult = WaitForMultipleObjects(

THREADCOUNT,

hThreads,

TRUE,

INFINITE);

switch (dwWaitResult)

{

case WAIT\_OBJECT\_0:

printf("all thread ended...\n");

printf("result : %d\n", number);

break;

default:

printf("WaitForMultipleObjects failed (%d)\n", GetLastError());

return 1;

}

CloseEvents();

return 0;

}

#include "stdafx.h"

#include <windows.h>

#include <stdio.h>

#define THREADCOUNT 2

#define MAX\_VALUE 6;

#define MIN\_VALUE 1;

HANDLE ghMutex;

HANDLE hThreads[THREADCOUNT];

DWORD WINAPI RollTheDice(LPVOID);

int diceNumber = 0;

int CreateMuteAndThreads()

{

DWORD ThreadID;

ghMutex = CreateMutex(

NULL,

FALSE,

NULL);

if (ghMutex == NULL)

{

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

return 1;

}

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

{

hThreads[i] = CreateThread(

NULL,

0,

(LPTHREAD\_START\_ROUTINE)RollTheDice,

NULL,

0,

&ThreadID);

if (hThreads[i] == NULL)

{

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

return 1;

}

}

}

int main(void)

{

CreateMuteAndThreads();

WaitForMultipleObjects(THREADCOUNT, hThreads, TRUE, INFINITE);

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

CloseHandle(hThreads[i]);

CloseHandle(ghMutex);

return 0;

}

DWORD WINAPI RollTheDice(LPVOID lpParam)

{

UNREFERENCED\_PARAMETER(lpParam);

DWORD dwCount = 0, dwWaitResult;

while (dwCount < 10)

{

dwWaitResult = WaitForSingleObject(

ghMutex,

INFINITE);

switch (dwWaitResult)

{

case WAIT\_OBJECT\_0:

\_\_try {

printf("Thread %d next...step %d\n",

GetCurrentThreadId(), dwCount);

dwCount++;

}

\_\_finally {

if (!ReleaseMutex(ghMutex))

{

// Handle error.

}

}

break;

case WAIT\_ABANDONED:

return FALSE;

}

}

return TRUE;

}

#include "stdafx.h"

#include <windows.h>

#include <stdio.h>

#define MAX\_SEM\_COUNT 10

#define THREADCOUNT 12

HANDLE ghSemaphore;

DWORD WINAPI ThreadProc(LPVOID);

int main(void)

{

HANDLE aThread[THREADCOUNT];

DWORD ThreadID;

int i;

ghSemaphore = CreateSemaphore(

NULL,

MAX\_SEM\_COUNT,

MAX\_SEM\_COUNT,

NULL);

if (ghSemaphore == NULL)

{

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

return 1;

}

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

{

aThread[i] = CreateThread(

NULL,

0,

(LPTHREAD\_START\_ROUTINE)ThreadProc,

NULL,

0,

&ThreadID);

if (aThread[i] == NULL)

{

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

return 1;

}

}

WaitForMultipleObjects(THREADCOUNT, aThread, TRUE, INFINITE);

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

CloseHandle(aThread[i]);

CloseHandle(ghSemaphore);

return 0;

}

DWORD WINAPI ThreadProc(LPVOID lpParam)

{

UNREFERENCED\_PARAMETER(lpParam);

DWORD dwWaitResult;

BOOL bContinue = TRUE;

while (bContinue)

{

dwWaitResult = WaitForSingleObject(

ghSemaphore,

0L);

switch (dwWaitResult)

{

case WAIT\_OBJECT\_0:

// TODO: Perform task

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

bContinue = FALSE;

Sleep(5);

if (!ReleaseSemaphore(

ghSemaphore,

1,

NULL))

{

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

}

break;

case WAIT\_TIMEOUT:

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

break;

}

}

return TRUE;

}