/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Lab2. C++. Win32.

\*\* Mutex & Semaphores.

\*\* Matviychyk Bohdan. IO-91.

\*\* MA = a \* MB \* MC+b \* MO

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "stdafx.h"

#include <iostream>

#include <conio.h>

#include <windows.h>

using std::cout;

using std::endl;

const int n = 4000;

const int p = 4;

const int h = n/p;

int MB[n][n];

int MA[n][n];

int MO[n][n];

// ОР:

int a;

int b;

int MC[n][n];

// Семафоры:

HANDLE Sim1 = CreateSemaphore(NULL,0,3,NULL);

HANDLE Sim2 = CreateSemaphore(NULL,0,3,NULL);

HANDLE Sim3 = CreateSemaphore(NULL,0,3,NULL);

HANDLE Sim4 = CreateSemaphore(NULL,0,3,NULL)

// События:

HANDLE EvEnd2 = CreateEvent(NULL,TRUE,FALSE,NULL);

HANDLE EvEnd3 = CreateEvent(NULL,TRUE,FALSE,NULL);

HANDLE EvEnd4 = CreateEvent(NULL,TRUE,FALSE,NULL);

HANDLE EndEvents[3] = {EvEnd2, EvEnd3, EvEnd4};

// Мютексы:

HANDLE Mute = CreateMutex(NULL,FALSE,NULL);

// CS

CRITICAL\_SECTION CS;

// \* \* \* \* \* \* \* Ввод матрицы.

void MatrixInput(int M[n][n])

{

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

{

for (int j = 0; j < n; j++)

{

M[i][j] = 1;

}

}

}

// \* \* \* \* \* \* \* Вывод матрицы.

void MatrixOutput(int M[n][n])

{

if (n <= 10)

{

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

{

for (int j = 0; j < n; j++)

{

cout<<M[i][j]<<" ";

}

cout<<endl;

}

cout<<endl;

}

}

// \* \* \* \* \* \* \* Умножение матриц.

void MatrixMnozh(int one, int two, int M1[n][n], int M2[n][n], int MM[n][n])

{

for (int i = one; i < two; i++)

{

for (int j = 0; j < n; j++)

{

MM[j][i] = 0;

for (int k = 0; k < n; k++)

{

MM[j][i] += M1[k][j] \* M2[i][k];

}

}

}

}

void MatrixPuls(int one, int two, int M1[n][n], int M2[n][n], int MM[n][n])

{

for (int i = one; i < two; i++)

{

for (int j = 0; j < n; j++)

{

MM[i][j] += M1[i][j] \* M2[i][j];

}

}

}

// \* \* \* \* \*

void NumbMetrixMnozh(int one, int two, int NUM, int V1[n][n], int VE[n][n])

{

for (int i = one; i < two; i++)

{

for (int j=0;j<n;j++){

VE[i][j] = NUM \* V1[i][j];}

}

}

void ob (int one, int two,int al, int bl, int Mb[n][n], int Mc[n][n], int Mo[n][n],int MA[n][n]){

int Mz[n][n];

int Mx[n][n];

int My[n][n];

NumbMetrixMnozh(one,two,al,Mb,Mz);

MatrixMnozh(one,two,Mz,MC,Mx);

NumbMetrixMnozh(one,two,bl,Mo,My);

MatrixPuls(one,two,Mx,My,MA);

}

// \* \* \* \* \* \* \* Task1

void Task1()

{

cout<<"Task1 started"<<endl;

a=1;

ReleaseSemaphore(Sim1,3,NULL);//signal to t2,t3,t4

WaitForSingleObject(Sim2,INFINITE);

WaitForSingleObject(Sim3,INFINITE);

WaitForSingleObject(Sim4,INFINITE);

WaitForSingleObject(Mute,INFINITE);

int a1 = a;

int b1 = b;

ReleaseMutex(Mute);

int MC1[n][n];

EnterCriticalSection(&CS);

memcpy(MC1, MC, sizeof(MC));

LeaveCriticalSection(&CS);

ob(0,h,a1,b1,MC1,MB,MO,MA);

WaitForMultipleObjects(3, EndEvents, TRUE, INFINITE);//receiving a signal from the t2,t3,t4

ResetEvent(EvEnd2);

ResetEvent(EvEnd3);

ResetEvent(EvEnd4);

MatrixOutput(MA);

cout<<"Task1 finished."<<endl;

}

// \* \* \* \* \* \* \* Task2

void Task2()

{

cout<<"Task2 started"<<endl;

b=1;

MatrixInput(MB);

ReleaseSemaphore(Sim2,3,NULL);

WaitForSingleObject(Sim1,INFINITE);//receiving a signal from the t1

WaitForSingleObject(Sim3,INFINITE);//receiving a signal from the t3

WaitForSingleObject(Sim4,INFINITE);//receiving a signal from the t4

WaitForSingleObject(Mute,INFINITE);//CS

int a2 = a;

int b2=b;

ReleaseMutex(Mute);//end CS

int MC2[n][n];

EnterCriticalSection(&CS);

memcpy(MC2, MC, sizeof(MC));

LeaveCriticalSection(&CS);

ob(h,2\*h,a2,b2,MC2,MB,MO,MA);

SetEvent(EvEnd2);//signal to t1

cout<<"Task2 finished."<<endl;

}

// \* \* \* \* \* \* \* Task3

void Task3()

{

cout<<"Task3 started"<<endl;

MatrixInput(MC);

ReleaseSemaphore(Sim3,3,NULL);//signal to t1,t2,t4

WaitForSingleObject(Sim1,INFINITE);//receiving a signal from the t1

WaitForSingleObject(Sim2,INFINITE);

WaitForSingleObject(Sim4,INFINITE);//receiving a signal from the t4

WaitForSingleObject(Mute,INFINITE);//CS

int a3 = a;

int b3 = b;

ReleaseMutex(Mute);//end CS

int MC3[n][n];

EnterCriticalSection(&CS);

memcpy(MC3, MC, sizeof(MC));

LeaveCriticalSection(&CS);

ob(2\*h,3\*h,a3,b3,MC3,MB,MO,MA);

SetEvent(EvEnd3);//signal to t1

cout<<"Task3 finished."<<endl;

}

// \* \* \* \* \* \* \* Task4

void Task4()

{

cout<<"Task4 started"<<endl;

MatrixInput(MO);

ReleaseSemaphore(Sim4,3,NULL);//signal to t1,t2,t3

WaitForSingleObject(Sim1,INFINITE);//receiving a signal from the t1

WaitForSingleObject(Sim2,INFINITE);

WaitForSingleObject(Sim3,INFINITE);//receiving a signal from the t3

WaitForSingleObject(Mute,INFINITE);//CS

int a4 = a;

int b4 = b;

ReleaseMutex(Mute);//end CS

int MC4[n][n];

EnterCriticalSection(&CS);

memcpy(MC4, MC, sizeof(MC));

LeaveCriticalSection(&CS);

ob(3\*h,4\*h,a4,b4,MC4,MB,MO,MA);

SetEvent(EvEnd4);//signal to t1

cout<<"Task4 finished."<<endl;

}

// \* \* \* \* \* \* \*

void main()

{

cout<<"Lab2 started"<<endl;

//#pragma comment(linker, "/STACK:500000000")

DWORD t1id, t2id, t3id,t4id;

InitializeCriticalSection(&CS);

HANDLE TT1 = CreateThread(NULL, 125000000, (LPTHREAD\_START\_ROUTINE)Task1, NULL, CREATE\_SUSPENDED, &t1id);

HANDLE TT2 = CreateThread(NULL, 125000000, (LPTHREAD\_START\_ROUTINE)Task2, NULL, CREATE\_SUSPENDED, &t2id);

HANDLE TT3 = CreateThread(NULL, 125000000, (LPTHREAD\_START\_ROUTINE)Task3, NULL, CREATE\_SUSPENDED, &t3id);

HANDLE TT4 = CreateThread(NULL, 125000000, (LPTHREAD\_START\_ROUTINE)Task4, NULL, CREATE\_SUSPENDED, &t4id);

ResumeThread(TT1);

ResumeThread(TT2);

ResumeThread(TT3);

ResumeThread(TT4);

WaitForSingleObject(TT1, INFINITE);

WaitForSingleObject(TT2, INFINITE);

WaitForSingleObject(TT3, INFINITE);

WaitForSingleObject(TT4, INFINITE);

cout<<"Lab2 finished."<<endl;

getchar();

}