Міністерство освіти і науки України

Національний технічний університет України

«Київський політехнічний інститут ім. Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Лабораторна робота №2

з дисципліни **«**Програмування паралельних комп’ютерних систем**»**

C++, Win32

Виконав:

студент 3 курсу

ФІОТ гр. ІО-53

Возний Олександр

Залікова - 5304

Перевірив:

Корочкін О. В.

Київ – 2018 р.

**Технічне завдання:**

Розробити програму для розв’язання ПКС із СП (структура на рис. 1) математичної задачі: a = min(d\*Z + e\*T\*(MO\*MK)).

Мова реалізації: C++, бібліотека Win32.

Засоби взаємодії: семафори, м’ютекси, події, критичні секції.



Рис. 1 Структурна схема ПКС

**Виконання роботи:**

**Етап 1. Розробка паралельного алгоритму**

1. B­h = d\*Zh + e\*T\*(MO\*MKh)

СР: d, e, T, MO

1. mini = min(Bh), i = 1->4
2. min = min(min, mini)

СР: min

**Етап 2. Розроблення алгоритмів роботи кожного процесу**

Т1

1. Введення даних Z, e, MK
2. Сигнал T2, T3, T4 про завершення введення S ­2,3,4-1­
3. Очікувати сигнали T3, T4 про введення W­3,4-1
4. Копія d1=d
5. Копія e1=e КД
6. Копія T1=T
7. Копія MO1=MO
8. Розрахунок B­h = d\*Zh + e\*T\*(MO\*MKh)
9. Сигнал T2, T3, T4 про обчислення S­­2,3,4-2
10. Очікувати сигнали T2, T3, T4 про обчислення W­2,3,4-2
11. Обчислення: min1 = min(Bh )
12. Обчислення: min = min(min, min1) КД
13. Сигнал Т3 про завершення обчислення S3-3

Т2

1. Очікувати сигнали Т1,T3,T4 про введення W1,3,4-1
2. Копія d2=d
3. Копія e2=e
4. Копія T2=T КД
5. Копія MO2=MO
6. Розрахунок B­h = d\*Zh + e\*T\*(MO\*MKh)
7. Сигнал T1, T3, T4 про обчислення S­­1,3,4-1
8. Очікувати сигнали T1, T3, T4 про обчислення W­1,3,4-2
9. Обчислення: min2 = min(Bh )
10. Обчислення: min = min(min, min2) КД
11. Сигнал Т3 про завершення обчислення S3-2

Т3

1. Введення даних d, MO
2. Сигнал T1, T2, T4 про завершення введення S ­1,2,4-1­
3. Очікувати сигнали T1, T4 про введення W­1,4-1
4. Копія d3=d
5. Копія e3=e КД
6. Копія T3=T
7. Копія MO3=MO
8. Розрахунок B­h = d\*Zh + e\*T\*(MO\*MKh)
9. Сигнал T1, T2, T4 про обчислення S­­1,2,4-2
10. Очікувати сигнали T1, T2, T4 про обчислення W­1,2,4-2
11. Обчислення: min3 = min(Bh )
12. Обчислення: min = min(min, min3) КД
13. Очікувати сигнал T1, T2, T4 про завершення обчислень W1,2,4-3
14. Виведення результату a

Т4

1. Введення даних T
2. Сигнал T1, T2, T3 про завершення введення S ­1,2,3-1­
3. Очікувати сигнали T1, T3 про введення W­1,3-1
4. Копія d4=d
5. Копія e4=e КД
6. Копія T4=T
7. Копія MO4=MO
8. Розрахунок B­h = d\*Zh + e\*T\*(MO\*MKh)
9. Сигнал T1, T2, T3 про обчислення S­­1,2,3-2
10. Очікувати сигнали T1, T2, T3 про обчислення W­1,2,3-2
11. Обчислення: min4 = min(Bh )
12. Обчислення: min = min(min, min4) КД
13. Сигнал Т3 про завершення обчислення S3-3

**Етап 4. Розроблення програми**

// PROLab4.cpp : Defines the entry point for the console application.

//

#include "stdafx.h"

#include "Vector.h"

#include <iostream>

#include <algorithm>

#include <functional>

#include <vector>

#include <windows.h>

//using namespace std;

//

// Lab #2 (winapi) Parallel and distributed computing

//

// IO-53 Voznyi Oleksandr, #5304

//

// a = min(d\*Z + e\*T\*(MO\*MK))

//

#define MAX\_THREADS 4

DWORD dwThreadIdArray[MAX\_THREADS];

HANDLE hThreadArray[MAX\_THREADS];

HANDLE hEventArray[3]; //input

HANDLE hEventArray2[3]; //calculations final part

HANDLE hSemaphoreArray[4]; //callculations part1

HANDLE mutex;

HANDLE semaphore;

Vector\* Z;

Matrix\* MK;

Matrix\* MO;

Vector\* T;

CRITICAL\_SECTION CriticalSection;

int d;

int e;

int a;

int N = 5;

int P = 4;

int H = N / P;

int min = 99999;

//Vector\* B = new Vector(N);

void f1(void)

{

std::cout << "Thread 1 started \n";

MK = new Matrix(N);

e = 1;

Z = new Vector(N);

Z->setElementAt(0, -2);

SetEvent(hEventArray[0]);

WaitForMultipleObjects(3, hEventArray, true, INFINITE);

//wait for input

WaitForSingleObject(semaphore, INFINITE);

//copy part1

int d1 = d;

int e1 = e;

Vector\* T1 = T;

ReleaseSemaphore(semaphore, 1, NULL);

EnterCriticalSection(&CriticalSection);

//copy part2

Matrix\* MO1 = MO;

LeaveCriticalSection(&CriticalSection);

//executions

Matrix\* MT = new Matrix(N);

MT = MO1->Mult(MK, 0, H);

Vector\*R = new Vector(N);

R = MT->Mult(T1, 0, H);

R->Mult(e1, 0, H);

Z->Mult(d1, 0, H);

Vector\* B = new Vector(N);

B = R->Add(Z, 0, H);

//B->Add(R, Z, H, 2 \* H);

ReleaseSemaphore(hSemaphoreArray[0], 4, NULL);

WaitForMultipleObjects(4, hSemaphoreArray, true, INFINITE);

int min1 = 99999;

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

{

if (B->elementAt(i) < min1)

{

min1 = B->elementAt(i);

}

}

WaitForSingleObject(mutex, INFINITE);

//stuff with min here

if (min1 < min)

{

min = min1;

}

ReleaseMutex(mutex);

SetEvent(hEventArray2[0]);

//?? WaitForMultipleObjects(4, hEventArray2, true, INFINITE);

std::cout << "Thread 1 finished\n";

}

void f2(void)

{

std::cout << "Thread 2 started \n";

WaitForMultipleObjects(3, hEventArray, true, INFINITE);

//wait for input

WaitForSingleObject(semaphore, INFINITE);

//copy part1

int d2 = d;

int e2 = e;

Vector\* T2 = T;

ReleaseSemaphore(semaphore, 1, NULL);

EnterCriticalSection(&CriticalSection);

//copy part2

Matrix\* MO2 = MO;

LeaveCriticalSection(&CriticalSection);

//executions

Matrix\* MT = new Matrix(N);

MT = MO2->Mult(MK, H, 2 \* H);

Vector\*R = new Vector(N);

R = MT->Mult(T2, H, 2 \* H);

R->Mult(e2, H, 2 \* H);

Z->Mult(d2, H, 2 \* H);

Vector\* B = new Vector(N);

B = R->Add(Z, H, 2 \* H);

//B->Add(R, Z, H, 2 \* H);

ReleaseSemaphore(hSemaphoreArray[1], 4, NULL);

WaitForMultipleObjects(4, hSemaphoreArray, true, INFINITE);

int min2 = 99999;

for (int i = H; i < 2 \* H; i++)

{

if (B->elementAt(i) < min2)

{

min2 = B->elementAt(i);

}

}

WaitForSingleObject(mutex, INFINITE);

//stuff with min here

if (min2 < min)

{

min = min2;

}

ReleaseMutex(mutex);

SetEvent(hEventArray2[1]);

//?? WaitForMultipleObjects(4, hEventArray2, true, INFINITE);

std::cout << "Thread 2 finished\n";

}

void f3(void)

{

std::cout << "Thread 3 started \n";

d = 1;

MO = new Matrix(N);

SetEvent(hEventArray[1]);

WaitForMultipleObjects(3, hEventArray, true, INFINITE);

//wait for input

WaitForSingleObject(semaphore, INFINITE);

//copy part1

int d3 = d;

int e3 = e;

Vector\* T3 = T;

ReleaseSemaphore(semaphore, 1, NULL);

EnterCriticalSection(&CriticalSection);

//copy part2

Matrix\* MO3 = MO;

LeaveCriticalSection(&CriticalSection);

//executions

Matrix\* MT = new Matrix(N);

MT = MO3->Mult(MK, 2 \* H, 3 \* H);

Vector\*R = new Vector(N);

R = MT->Mult(T3, 2 \* H, 3 \* H);

R->Mult(e3, 2 \* H, 3 \* H);

Z->Mult(d3, 2 \* H, 3 \* H);

Vector\* B = new Vector(N);

B = R->Add(Z, 2 \* H, 3 \* H);

//B->Add(R, Z, H, 2 \* H);

ReleaseSemaphore(hSemaphoreArray[2], 4, NULL);

WaitForMultipleObjects(4, hSemaphoreArray, true, INFINITE);

int min3 = 99999;

for (int i = 2 \* H; i < 3 \* H; i++)

{

if (B->elementAt(i) < min3)

{

min3 = B->elementAt(i);

}

}

WaitForSingleObject(mutex, INFINITE);

//stuff with min here

if (min3 < min)

{

min = min3;

}

ReleaseMutex(mutex);

//SetEvent(hEventArray2[1]);

WaitForMultipleObjects(3, hEventArray2, true, INFINITE);

int a = min;

std::cout << "\n";

if (N < 10)

{

std::cout << "a = " << a;

std::cout << "\n";

}

std::cout << "Thread 3 finished\n";

}

void f4(void)

{

std::cout << "Thread 4 started \n";

T = new Vector(N);

SetEvent(hEventArray[2]);

WaitForMultipleObjects(3, hEventArray, true, INFINITE);

//wait for input

WaitForSingleObject(semaphore, INFINITE);

//copy part1

int d4 = d;

int e4 = e;

Vector\* T4 = T;

ReleaseSemaphore(semaphore, 1, NULL);

EnterCriticalSection(&CriticalSection);

//copy part2

Matrix\* MO4 = MO;

LeaveCriticalSection(&CriticalSection);

//executions

Matrix\* MT = new Matrix(N);

MT = MO4->Mult(MK, 3 \* H, 4 \* H);

Vector\*R = new Vector(N);

R = MT->Mult(T4, 3 \* H, 4 \* H);

R->Mult(e4, 3 \* H, 4 \* H);

Z->Mult(d4, 3 \* H, 4 \* H);

Vector\* B = new Vector(N);

B = R->Add(Z, 3 \* H, 4 \* H);

//B->Add(R, Z, H, 2 \* H);

ReleaseSemaphore(hSemaphoreArray[3], 4, NULL);

WaitForMultipleObjects(4, hSemaphoreArray, true, INFINITE);

int min4 = 99999;

for (int i = 3 \* H; i < 4 \* H; i++)

{

if (B->elementAt(i) < min4)

{

min4 = B->elementAt(i);

}

}

WaitForSingleObject(mutex, INFINITE);

//stuff with min here

if (min4 < min)

{

min = min4;

}

ReleaseMutex(mutex);

SetEvent(hEventArray2[2]);

//?? WaitForMultipleObjects(4, hEventArray2, true, INFINITE);

std::cout << "Thread 4 finished\n";

}

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

{

//Events for input

hEventArray[0] = CreateEvent(NULL, true, false, (LPCWSTR) "Event1");

hEventArray[1] = CreateEvent(NULL, true, false, (LPCWSTR) "Event2");

hEventArray[2] = CreateEvent(NULL, true, false, (LPCWSTR) "Event3");

//Events for output

hEventArray2[0] = CreateEvent(NULL, true, false, (LPCWSTR) "Event4");

hEventArray2[1] = CreateEvent(NULL, true, false, (LPCWSTR) "Event5");

hEventArray2[2] = CreateEvent(NULL, true, false, (LPCWSTR) "Event6");

//Semaphores for intermediate calcultaions

hSemaphoreArray[0] = CreateSemaphore(NULL, 0, 4, (LPCWSTR) "Sem1");

hSemaphoreArray[1] = CreateSemaphore(NULL, 0, 4, (LPCWSTR) "Sem2");

hSemaphoreArray[2] = CreateSemaphore(NULL, 0, 4, (LPCWSTR) "Sem3");

hSemaphoreArray[3] = CreateSemaphore(NULL, 0, 4, (LPCWSTR) "Sem4");

mutex = CreateMutex(NULL, 0, (LPCWSTR) "Mut");

semaphore = CreateSemaphore(NULL, 1, 1, (LPCWSTR) "Sem");

InitializeCriticalSection(&CriticalSection);

hThreadArray[0] = CreateThread(

NULL, // default security attributes

0, // use default stack size

(LPTHREAD\_START\_ROUTINE)f1, // thread function name

NULL, // argument to thread function

0, // use default creation flags

&dwThreadIdArray[0]); // returns the thread identifier

hThreadArray[1] = CreateThread(

NULL, // default security attributes

0, // use default stack size

(LPTHREAD\_START\_ROUTINE)f2, // thread function name

NULL, // argument to thread function

0, // use default creation flags

&dwThreadIdArray[1]); // returns the thread identifier

hThreadArray[2] = CreateThread(

NULL, // default security attributes

0, // use default stack size

(LPTHREAD\_START\_ROUTINE)f3, // thread function name

NULL, // argument to thread function

0, // use default creation flags

&dwThreadIdArray[2]); // returns the thread identifier

hThreadArray[3] = CreateThread(

NULL, // default security attributes

0, // use default stack size

(LPTHREAD\_START\_ROUTINE)f4, // thread function name

NULL, // argument to thread function

0, // use default creation flags

&dwThreadIdArray[3]); // returns the thread identifier

int k;

std::cin >> k;

return 0;

}