Міністерство освіти і науки України

Національний технічний університет України

«Київський політехнічний інститут»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

**Лабораторна робота №2**

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

**на тему «Win32»**

Виконав:

студент ІІI курсу

групи ІО-53

Лісовий В. О.

Перевірив:

доц. Корочкін О. В.

Київ – 2018

**Тема**

«Програмування для комп’ютерних систем зі спільною пам’яттю. Win32. Семафори, м’ютекси, події, критичні секції»

**Технічне завдання**

Розробити програму для розв’язання ПКС із СП (структура на рис. 1) математичної задачі: A = sort(e \* Z + d \* S \* (MO \* MH)).

Мова програмування: C/C++.

Бібліотека: Win32.

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



Рис. 1 Структурна схема ПКС

**Виконання роботи**

**Етап 1. Побудова паралельного алгоритму**

1. A­H = e \* ZH + d \* S \* (MO \* MHH);
2. AH = sort(AH);
3. A = merge (AH, AH).

Спільний ресурс: e, d, S, MO.

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

|  |  |  |
| --- | --- | --- |
| № | Задача Т1 |  |
| 1 | Ввід даних (S) |  |
| 2 | Сигнал Т2, Т3, Т4 про завершення вводу даних | S2,1, S3,1, S4,1 |
| 3 | Чекати завершення вводу даних в Т2, Т3, Т4 | W2,1, W3,1, W4,1 |
| 4 | Копіювати e1=e, d1=d | КД |
| 5 | Копіювати S1=S, MO1=MO | КД |
| 6 | Обчислення A­H = e1 \* ZH + d1 \* S1 \* (MO1 \* MHH) |  |
| 7 | Обчислення AH = sort(AH) |  |
| 8 | Чекати завершення сортування в Т2 | W2,2 |
| 9 | Обчислення A = merge(AH, AH) |  |
| 10 | Сигнал Т4 про завершення злиття | S4,3 |

|  |  |  |
| --- | --- | --- |
| № | Задача Т2 |  |
| 1 | Ввід даних (MH) |  |
| 2 | Сигнал Т1, Т3, Т4 про завершення вводу даних | S1,1, S3,1, S4,1 |
| 3 | Чекати завершення вводу даних в Т1, Т3, Т4 | W1,1, W3,1, W4,1 |
| 4 | Копіювати e2=e, d2=d | КД |
| 5 | Копіювати S2=S, MO2=MO | КД |
| 6 | Обчислення A­H = e2 \* ZH + d2 \* S2 \* (MO2 \* MHH) |  |
| 7 | Обчислення AH = sort(AH) |  |
| 8 | Сигнал Т1 про завершення сортування | S1,2 |

|  |  |  |
| --- | --- | --- |
| № | Задача Т3 |  |
| 1 | Ввід даних (e, d, MO) |  |
| 2 | Сигнал Т1, Т2, Т4 про завершення вводу даних | S1,1, S2,1, S4,1 |
| 3 | Чекати завершення вводу даних в Т2, Т3, Т4 | W1,1, W2,1, W4,1 |
| 4 | Копіювати e3=e, d3=d | КД |
| 5 | Копіювати S3=S, MO3=MO | КД |
| 6 | Обчислення A­H = e3 \* ZH + d3 \* S3 \* (MO3 \* MHH) |  |
| 7 | Обчислення AH = sort(AH) |  |
| 8 | Сигнал Т4 про завершення сортування | S4,2 |

|  |  |  |
| --- | --- | --- |
| № | Задача Т4 |  |
| 1 | Ввід даних (A, Z) |  |
| 2 | Сигнал Т1, Т2, Т3 про завершення вводу даних | S1,1, S2,1, S3,1 |
| 3 | Чекати завершення вводу даних в Т1, Т2, Т3 | W1,1, W2,1, W3,1 |
| 4 | Копіювати e4=e, d4=d | КД |
| 5 | Копіювати S4=S, MO4=MO | КД |
| 6 | Обчислення A­H = e4 \* ZH + d4 \* S4 \* (MO4 \* MHH) |  |
| 7 | Обчислення AH = sort(AH) |  |
| 8 | Чекати завершення сортування в Т3 | W3,2 |
| 9 | Обчислення A = merge(AH, AH) |  |
| 8 | Чекати завершення злиття в Т1 | W1,3 |
| 10 | Обчислення A = merge(AH, AH) |  |
| 11 | Вивід результату А |  |

**Етап 3. Розроблення структурної схеми взаємодії задач**

На структурній схемі взаємодії задач уведено такі семафори

* Semaphore1 – семафор для синхронізації з завершенням введення в Т1
* Semaphore2 – семафор для синхронізації з завершенням введення в Т2
* Semaphore3 – семафор для синхронізації з завершенням введення в Т3
* Semaphore4 – семафор для синхронізації з завершенням введення в Т4
* Mutex – м’ютекс для керування доступом до спільних ресурсів
* Cs – критична секція для керування доступом до спільних ресурсів
* Event2\_1 – подія для синхронізації Т1 з завершенням сортування в Т2
* Event3\_4 – подія для синхронізації Т4 з завершенням сортування в Т3
* Event1\_4 – подія для синхронізації Т4 з завершенням злиття в Т1

**Етап 4. Розробка програми**

#include "stdafx.h"

#include <windows.h>

#include <iostream>

using std::cout;

using std::cin;

int n;

const int num = 1;

int e, d;

int \*A, \*Z, \*S;

int \*MO, \*MH;

HANDLE Semaphore1 = CreateSemaphore(NULL, 0, 3, NULL);

HANDLE Semaphore2 = CreateSemaphore(NULL, 0, 3, NULL);

HANDLE Semaphore3 = CreateSemaphore(NULL, 0, 3, NULL);

HANDLE Semaphore4 = CreateSemaphore(NULL, 0, 3, NULL);

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

CRITICAL\_SECTION CS;

HANDLE Event2\_1 = CreateEvent(NULL, TRUE, FALSE, NULL); // from t2 to t1 about finish sorting

HANDLE Event3\_4 = CreateEvent(NULL, TRUE, FALSE, NULL); // from t3 to t4 about finish sorting

HANDLE Event1\_4 = CreateEvent(NULL, TRUE, FALSE, NULL); // from t1 to t4 about finish merging

void thread1() {

cout << "Thread 1 start\n";

// enter data

S = (int \*)malloc(n \* 4);

fillVector(num, S, n);

// signal to t2, t3, t4

ReleaseSemaphore(Semaphore1, 3, NULL);

// wait t2, t3, t4

WaitForSingleObject(Semaphore2, INFINITE);

WaitForSingleObject(Semaphore3, INFINITE);

WaitForSingleObject(Semaphore4, INFINITE);

//copy shared variables

WaitForSingleObject(Mutex, INFINITE);

int e1 = e;

int d1 = d;

ReleaseMutex(Mutex);

//copy shared variables

EnterCriticalSection(&CS);

int \*S1 = S;

int \*MO1 = MO;

LeaveCriticalSection(&CS);

F(A, e1, Z, d1, S1, MO1, MH, 0, n/4, n);

vectorSort(A, 0, n / 4);

// wait untill t2 finish sorting

WaitForSingleObject(Event2\_1, INFINITE);

merge(A, 0, n / 4, n / 4);

// signal to t4 about finish merging

SetEvent(Event1\_4);

cout << "Thread 1 finish\n";

}

void thread2() {

cout << "Thread 2 start\n";

MH = (int \*)malloc(n \* n \* 4);

fillMatrix(num, MH, n);

// signal to t1, t3, t4

ReleaseSemaphore(Semaphore2, 3, NULL);

// wait t1, t3, t4

WaitForSingleObject(Semaphore1, INFINITE);

WaitForSingleObject(Semaphore3, INFINITE);

WaitForSingleObject(Semaphore4, INFINITE);

//copy shared variables

WaitForSingleObject(Mutex, INFINITE);

int e2 = e;

int d2 = d;

ReleaseMutex(Mutex);

//copy shared variables

EnterCriticalSection(&CS);

int \*S2 = S;

int \*MO2 = MO;

LeaveCriticalSection(&CS);

F(A, e2, Z, d2, S2, MO2, MH, n / 4, n / 2, n);

vectorSort(A, n /4, n / 2);

// signal to t1 about finish sorting

SetEvent(Event2\_1);

cout << "Thread 2 finish\n";

}

void thread3() {

cout << "Thread 3 start\n";

e = 1;

d = 1;

MO = (int \*)malloc(n \* n \* 4);

fillMatrix(num, MO, n);

// signal to t1, t2, t4

ReleaseSemaphore(Semaphore3, 3, NULL);

// wait t1, t2, t4

WaitForSingleObject(Semaphore1, INFINITE);

WaitForSingleObject(Semaphore2, INFINITE);

WaitForSingleObject(Semaphore4, INFINITE);

//copy shared variables

WaitForSingleObject(Mutex, INFINITE);

int e3 = e;

int d3 = d;

ReleaseMutex(Mutex);

//copy shared variables

EnterCriticalSection(&CS);

int \*S3 = S;

int \*MO3 = MO;

LeaveCriticalSection(&CS);

F(A, e3, Z, d3, S3, MO3, MH, n / 2, 3 \* n / 4, n);

vectorSort(A, n / 2, 3 \* n / 4);

// signal to t4 about finish sorting

SetEvent(Event3\_4);

cout << "Thread 3 finish\n";

}

void thread4() {

cout << "Thread 4 start\n";

A = (int \*)malloc(n \* 4);

Z = (int \*)malloc(n \* 4);

fillVector(num, Z, n);

// signal to t1, t2, t3

ReleaseSemaphore(Semaphore4, 3, NULL);

// wait t1, t2, t3

WaitForSingleObject(Semaphore1, INFINITE);

WaitForSingleObject(Semaphore2, INFINITE);

WaitForSingleObject(Semaphore3, INFINITE);

//copy shared variables

WaitForSingleObject(Mutex, INFINITE);

int e4 = e;

int d4 = d;

ReleaseMutex(Mutex);

//copy shared variables

EnterCriticalSection(&CS);

int \*S4 = S;

int \*MO4 = MO;

LeaveCriticalSection(&CS);

F(A, e4, Z, d4, S4, MO4, MH, 3 \* n / 4, n, n);

vectorSort(A, 3 \* n / 4, n);

// wait untill t3 finish sorting

WaitForSingleObject(Event3\_4, INFINITE);

merge(A, n / 2, 3 \* n / 4, n / 4);

// wait untill t1 finish merging

WaitForSingleObject(Event1\_4, INFINITE);

merge(A, 0, n / 2, n / 2);

if (n < 5) {

cout << "A = ";

printVector(A, n);

}

cout << "Thread 4 finish\n";

}

void freeMemory() {

free(A);

free(Z);

free(S);

free(MO);

free(MH);

}

int main()

{

cout << "Main start\n";

cout << "Enter n\n";

cin >> n;

InitializeCriticalSection(&CS);

HANDLE hThread[4];

DWORD Tid1, Tid2, Tid3, Tid4;

hThread[0] = CreateThread(NULL, 300000000, (LPTHREAD\_START\_ROUTINE)thread1, NULL, 0, &Tid1);

hThread[1] = CreateThread(NULL, 300000000, (LPTHREAD\_START\_ROUTINE)thread2, NULL, 0, &Tid2);

hThread[2] = CreateThread(NULL, 300000000, (LPTHREAD\_START\_ROUTINE)thread3, NULL, 0, &Tid3);

hThread[3] = CreateThread(NULL, 300000000, (LPTHREAD\_START\_ROUTINE)thread4, NULL, 0, &Tid4);

WaitForMultipleObjects(4, hThread, TRUE, INFINITE);

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

CloseHandle(hThread[i]);

}

DeleteCriticalSection(&CS);

cout << "Main finish\n";

freeMemory();

int i;

cout << "Enter num to exit\n";

cin >> i;

return 0;

}

//-------------------------------------------------------------------------------------------

// Created by Foresstt on 28-Oct-17.

//--------------------------------------------------------------------------------------------

#include "stdafx.h"

void printVector(int \*A, int n) {

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

printf("%d ", A[i]);

printf("\n");

}

void printMatrix(int \*A, int n) {

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

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

printf("%d ", A[i\*n + j]);

printf("\n");

}

printf("\n");

}

void fillVector(int num, int \*A, int n) {

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

A[i] = num;

}

void fillMatrix(int num, int \*A, int n) {

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

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

A[i\*n + j] = num;

}

void matrixMultiplication(int \*A, int \*B, int \*C, int from, int to, int n) {

int buf;

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

for (int j = from; j < to; j++) {

buf = 0;

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

buf += A[i\*n + k] \* B[k\*n + j];

}

C[i\*n + j] = buf;

}

}

}

void vectorMatrixMultiplication(int \*A, int \*B, int \*C, int from, int to, int n) {

int buf;

for (int i = from; i < to; i++ ) {

buf = 0;

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

buf += A[j] \* B[j\*n + i];

}

C[i] = buf;

}

}

void vectorSort(int \*A, int from, int to) {

int index;

int min;

for (int i = from; i < to; i++) {

index = i;

min = A[i];

for (int j = i + 1; j < to; j++) {

if (A[j] < min) {

index = j;

min = A[j];

}

}

if (index != i) {

A[index] = A[i];

A[i] = min;

}

}

}

void merge(int \*A, int leftIndex, int rightIndex, int size) {

int index = leftIndex;

int leftSize = leftIndex + size;

int rightSize = rightIndex + size;

int \* buf = (int \*)malloc(size \* 2 \* 4);

int i;

for (i = 0 ; leftIndex < leftSize && rightIndex < rightSize; i++) {

if (A[leftIndex] < A[rightIndex]) {

buf[i] = A[leftIndex];

leftIndex++;

}

else {

buf[i] = A[rightIndex];

rightIndex++;

}

}

while (leftIndex < leftSize) {

buf[i++] = A[leftIndex++];

}

while (rightIndex < rightSize) {

buf[i++] = A[rightIndex++];

}

for (i = 0; i < size \* 2; i++)

A[index++] = buf[i];

free(buf);

}

void F(int \*A, int e, int \*Z, int d, int \*S, int \*MO, int \*MH, int from, int to, int n) {

int \*MBuf = (int \*)malloc(n \* n \* 4);

matrixMultiplication(MO, MH, MBuf, from, to, n);

vectorMatrixMultiplication(S, MBuf, A, from, to, n);

for (int i = from; i < to; i++) {

A[i] = A[i] \* d + e \* Z[i];

}

free(MBuf);

}

//--------------------------------------------------------------------------------------------

// Created by Foresstt on 28-Oct-17.

//--------------------------------------------------------------------------------------------

void printVector(int \*A, int n);

void fillVector(int num, int \*A, int n);

void fillMatrix(int num, int \*A, int n);

void vectorSort(int \*A, int from, int to);

void merge(int \*A, int leftIndex, int rightIndex, int size);

void F(int \*A, int e, int \*Z, int d, int \*S, int \*MO, int \*MH, int from, int to, int n);