Міністерство освіти і науки України

Національний технічний університет України

«Київський політехнічний інститут»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

**Лабораторна робота №3**

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

**на тему «C#»**

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. C#»

**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#.

Засоби взаємодії: 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. A2H = merge (AH, AH).
4. A4H = merge(A2H, A2H);
5. A = merge(A4H, A2H);

Спільний ресурс: e, d, S, MO.

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

|  |  |  |
| --- | --- | --- |
| № | Задача Т1 |  |
| 1 | Ввід даних (S) |  |
| 2 | Сигнал Т2-6 про завершення вводу даних | S2-6,1 |
| 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 сортування | S4,3 |

|  |  |  |
| --- | --- | --- |
| № | Задача Т2 |  |
| 1 | Ввід даних (MH) |  |
| 2 | Сигнал Т1, Т3-6 про завершення вводу даних | S1,1, S3-6,1 |
| 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, Т4-6 про завершення вводу даних | S1,1, S2,1, S4-6,1 |
| 3 | Чекати завершення вводу даних в Т1, Т2, 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 Т1-3, T5-6 про завершення вводу даних | S1-3,1, S5-6,1 |
| 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 A2H = merge(AH, AH) |  |
| 8 | Чекати завершення сортування в Т1 | W1,3 |
| 10 | Обчислення A4H = merge(A2H, A2H) |  |
| 11 | Чекати завершення сортування в Т5 | W5,4 |
| 12 | Обчислення A = merge(A4H, A2H) |  |
| 13 | Вивід результату А |  |

|  |  |  |
| --- | --- | --- |
| № | Задача Т5 |  |
| 1 | Чекати завершення вводу даних в Т1-4 | W1-4,1 |
| 2 | Копіювати e5=e, d5=d | КД |
| 3 | Копіювати S5=S, MO5=MO | КД |
| 4 | Обчислення A­H = e5 \* ZH + d5 \* S5 \* (MO5 \* MHH) |  |
| 5 | Обчислення AH = sort(AH) |  |
| 6 | Чекати завершення сортування в Т6 | W6,2 |
| 7 | Обчислення A2H = merge(AH, AH) |  |
| 8 | Сигнал Т4 про завершення сортування | S4, 4 |

|  |  |  |
| --- | --- | --- |
| № | Задача Т6 |  |
| 1 | Чекати завершення вводу даних в Т1-4 | W1-4,1 |
| 2 | Копіювати e5=e, d5=d | КД |
| 3 | Копіювати S5=S, MO5=MO | КД |
| 4 | Обчислення A­H = e5 \* ZH + d5 \* S5 \* (MO5 \* MHH) |  |
| 5 | Обчислення AH = sort(AH) |  |
| 6 | Сигнал Т5 про завершення сортування | S4, 2 |

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

На структурній схемі взаємодії задач уведено такі семафори

* manualEvencts[0] – подія для синхронізації з завершенням введення в Т1
* manualEvencts[1] – подія для синхронізації з завершенням введення в Т2
* manualEvencts[2] – подія для синхронізації з завершенням введення в Т3
* manualEvencts[3] – подія 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
* Event6\_5 – подія для синхронізації Т5 з завершенням сортування в Т6
* Event1\_4 – подія для синхронізації Т4 з завершенням злиття в Т1
* Event5\_4 – подія для синхронізації Т4 з завершенням сортування в Т5

**Етап 4. Розробка програми**

using System;

using System.Threading;

namespace Lab3

{

class Program

{

static void Main(string[] args)

{

Console.WriteLine("Main start");

int size = Convert.ToInt32(Console.ReadLine());

Functions threadFunctions = new Functions(size);

Thread[] threads = new Thread[6];

threads[0] = new Thread(() => threadFunctions.Task1(0));

threads[1] = new Thread(() => threadFunctions.Task2(1));

threads[2] = new Thread(() => threadFunctions.Task3(2));

threads[3] = new Thread(() => threadFunctions.Task4(3));

threads[4] = new Thread(() => threadFunctions.Taski(4));

threads[5] = new Thread(() => threadFunctions.Taski(5));

Console.WriteLine("Main start");

for (int i = 0; i < 6; i++)

threads[i].Start();

for (int i = 0; i < 6; i++)

threads[i].Join();

Console.WriteLine("Main finish");

Console.ReadLine();

}

}

}

using System;

using System.Threading;

namespace Lab3

{

public class Functions

{

private readonly int \_size;

private readonly float \_h;

int[] A, Z, S;

int[] MO, MH;

int e, d;

private ManualResetEvent[] manualEvencts = new ManualResetEvent[4]; // manual events for input

private Object cs = new Object(); // cs for coping data

private Mutex mutex = new Mutex(); // mutex for coping data

private AutoResetEvent ev2\_1 = new AutoResetEvent(false);

private AutoResetEvent ev3\_4 = new AutoResetEvent(false);

private AutoResetEvent ev6\_5 = new AutoResetEvent(false);

private AutoResetEvent ev1\_4 = new AutoResetEvent(false);

private AutoResetEvent ev5\_4 = new AutoResetEvent(false);

public void Task1(int id)

{

Console.WriteLine("Thread1 start");

fillVector(1, S); // input S

manualEvencts[id].Set();

WaitHandle.WaitAll(manualEvencts);

int e1, d1;

int[] S1;

int[] MO1;

lock(cs)

{

e1 = e;

d1 = d;

}

mutex.WaitOne();

S1 = S;

MO1 = MO;

mutex.ReleaseMutex();

F(A, e1, Z, d1, S1, MO1, MH, id \* \_size / 6, (id + 1) \* \_size / 6);

vectorSort(A, id \* \_size / 6, (id + 1) \* \_size / 6);

ev2\_1.WaitOne(); // wait t2 finish sorting

float hl = (float)\_size / 6;

int leftSize = (int)hl;

int rightSize = (int)(hl \* 2) - leftSize;

int rightIndex = leftSize;

merge(A, 0, rightIndex, leftSize, rightSize, leftSize + rightSize);

ev1\_4.Set(); // signal to t4 about finish merging

Console.WriteLine("Thread1 end");

}

public void Task2(int id)

{

Console.WriteLine("Thread2 start");

fillMatrix(1, MH);

manualEvencts[id].Set();

WaitHandle.WaitAll(manualEvencts);

int e2, d2;

int[] S2;

int[] MO2;

lock (cs)

{

e2 = e;

d2 = d;

}

mutex.WaitOne();

S2 = S;

MO2 = MO;

mutex.ReleaseMutex();

F(A, e2, Z, d2, S2, MO2, MH, id \* \_size / 6, (id + 1) \* \_size / 6);

vectorSort(A, id \* \_size / 6, (id + 1) \* \_size / 6);

ev2\_1.Set();

Console.WriteLine("Thread2 end");

}

public void Task3(int id)

{

Console.WriteLine("Thread3 start");

e = 1;

d = 1;

fillMatrix(1, MO);

manualEvencts[id].Set();

WaitHandle.WaitAll(manualEvencts);

int e3, d3;

int[] S3;

int[] MO3;

lock (cs)

{

e3 = e;

d3 = d;

}

mutex.WaitOne();

S3 = S;

MO3 = MO;

mutex.ReleaseMutex();

F(A, e3, Z, d3, S3, MO3, MH, id \* \_size / 6, (id + 1) \* \_size / 6);

vectorSort(A, id \* \_size / 6, (id + 1) \* \_size / 6);

ev3\_4.Set(); // signal to t4 about finish merging

Console.WriteLine("Thread3 end");

}

public void Task4(int id)

{

Console.WriteLine("Thread4 start");

fillVector(1, Z);

Z[0] = 4;

manualEvencts[id].Set();

WaitHandle.WaitAll(manualEvencts);

int e4, d4;

int[] S4;

int[] MO4;

lock (cs)

{

e4 = e;

d4 = d;

}

mutex.WaitOne();

S4 = S;

MO4 = MO;

mutex.ReleaseMutex();

F(A, e4, Z, d4, S4, MO4, MH, id \* \_size / 6, (id + 1) \* \_size / 6);

vectorSort(A, id \* \_size / 6, (id + 1) \* \_size / 6);

ev3\_4.WaitOne();

float hl = (float)\_size / 6;

int leftIndex = (int)(2 \* hl);

int leftSize = (int)hl;

int rightSize = (int)(hl \* 2) - leftSize;

int rightIndex = leftIndex + leftSize;

merge(A, leftIndex, rightIndex, leftSize, rightSize, leftSize + rightSize);

ev1\_4.WaitOne();

hl = (float)\_size / 3;

leftIndex = 0;

leftSize = (int)hl;

rightSize = (int)(hl \* 2) - leftSize;

rightIndex = leftIndex + leftSize;

merge(A, leftIndex, rightIndex, leftSize, rightSize, leftSize + rightSize);

ev5\_4.WaitOne();

leftIndex = 0;

leftSize = 2 \* \_size / 3;

rightSize = \_size - leftSize;

rightIndex = leftIndex + leftSize;

merge(A, leftIndex, rightIndex, leftSize, rightSize, leftSize + rightSize);

printVector(A);

Console.WriteLine("Thread4 end");

}

public void Taski(int id)

{

Console.WriteLine("Thread" + (id+1) + " start");

WaitHandle.WaitAll(manualEvencts);

int ei, di;

int[] Si;

int[] MOi;

lock (cs)

{

ei = e;

di = d;

}

mutex.WaitOne();

Si = S;

MOi = MO;

mutex.ReleaseMutex();

F(A, ei, Z, di, Si, MOi, MH, id \* \_size / 6, (id + 1) \* \_size / 6);

vectorSort(A, id \* \_size / 6, (id + 1) \* \_size / 6);

if ((id + 1) == 6) // t6

{

ev6\_5.Set(); // signal from t6 to t5 about finish merging

}

if ((id + 1) == 5) // t5

{

ev6\_5.WaitOne(); // t6 wait t5 finish sorting

float hl = (float)\_size / 6;

int leftSize = (int)hl;

int rightSize = (int)(hl \* 2) - leftSize;

int rightIndex = leftSize;

merge(A, 0, rightIndex, leftSize, rightSize, leftSize + rightSize);

ev5\_4.Set(); // signal to t4 about finish merging

}

Console.WriteLine("Thread" + (id + 1) + " end");

}

public Functions(int size)

{

\_size = size;

\_h = (float)size / 6;

A = new int[size];

Z = new int[size];

S = new int[size];

MO = new int[size \* size];

MH = new int[size \* size];

for (int i = 0; i < manualEvencts.Length; i++)

manualEvencts[i] = new ManualResetEvent(false);

}

void printVector(int[] A)

{

for (int i = 0; i < \_size; i++)

Console.Write("{0} ", A[i]);

Console.Write("\n");

}

void fillVector(int num, int[] A)

{

for (int i = 0; i < \_size; i++)

A[i] = num;

}

void fillMatrix(int num, int[] A)

{

for (int i = 0; i < \_size; i++)

for (int j = 0; j < \_size; j++)

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

}

void matrixMultiplication(int[] A, int[] B, int[] C, int from, int to)

{

int buf;

for (int i = 0; i < \_size; i++)

{

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

{

buf = 0;

for (int k = 0; k < \_size; k++)

{

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

}

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

}

}

}

void vectorMatrixMultiplication(int[] A, int[] B, int[] C, int from, int to)

{

int buf;

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

{

buf = 0;

for (int j = 0; j < \_size; j++)

{

buf += A[j] \* B[j \* \_size + 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 leftSize, int rightSize, int size)

{

int index = leftIndex;

leftSize += leftIndex;

rightSize += rightIndex;

int[] buf = new int[size];

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];

i++;

leftIndex++;

}

while (rightIndex < rightSize)

{

buf[i] = A[rightIndex];

i++;

rightIndex++;

}

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

{

A[index] = buf[i];

index++;

}

}

void F(int[] A, int e, int[] Z, int d, int[] S, int[] MO, int[] MH, int from, int to)

{

int[] MBuf = new int[\_size \* \_size];

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

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

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

{

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

}

}

}

}