Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Основи паралельного програмування

Лабораторна робота №3

**«Ада. Захищені модулі»**

Виконала:

студентка групи ІВ-71

Молчанова В.С.

Перевірив:

Корочкін О.В.

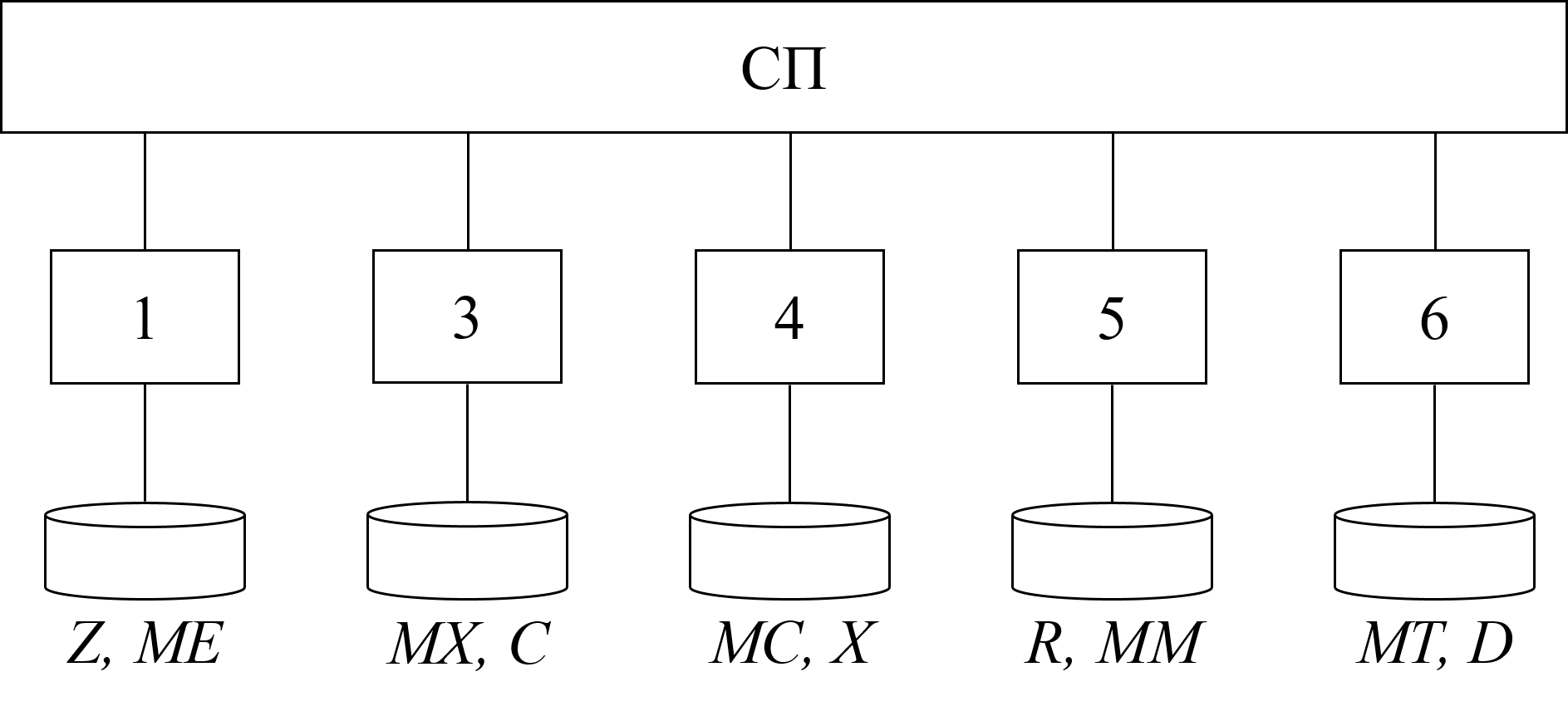
Київ

2020 р.

**Мета роботи**: розробка програми для ПКС зі СП.

## Завдання

* *Р=6*
* *Х = (D\*C)\*(R\*MT)\*ME - min(D)\*(Z\*MX)\*ММ + sort(Z\*MC)*



* Мова програмування: Ada
* Засоби: захищені модулі.

## Виконання роботи

### 1. Паралельний алгоритм

Алгоритм обчислення виразу:

1. 
2. 
3. 
4. 
5. 
6. 
7. 
8. 
9. 
10. 
11. 
12. 

Спільні ресурси: .

### 2. Алгоритми потоків

|  |  |  |
| --- | --- | --- |
|  | **Т4** | ТС КД |
| 1. | Ввести дані: *MC* |  |
| 2. | Очікувати введення даних в решті потоків |  |
| 3. | Копіювати *Z4=Z, R4=R* | КД |
| 4. |  |  |
| 5. |  |  |
| 6. |  |  |
| 7. |  |  |
| 8. | Очікувати завершення сортування в Т3 |  |
| 9. |  |  |
| 10. | Очікувати завершення сортування в Т2 |  |
| 11. |  |  |
| 12. | Очікувати завершення сортування в Т6 |  |
| 13. |  |  |
| 14. | Сигнал решті потоків про завершення сортування |  |
| 15. | Копіювати *a4=a, F4=F, m4=m, Y4=Y, S4=S* | КД |
| 16. | Обчислити |  |
| 17. | Очікувати завершення обчислення в решті потоків |  |
| 18. | Вивести *Х* |  |

|  |  |  |
| --- | --- | --- |
|  | **Т2** | ТС КД |
| 1. | Очікувати введення даних в решті потоків |  |
| 2. | Копіювати *Z2=Z, R2=R* | КД |
| 3. |  |  |
| 4. |  |  |
| 5. |  |  |
| 6. |  |  |
| 7. | Очікувати завершення сортування в Т1 |  |
| 8. |  |  |
| 9. | Сигнал T4 про завершення сортування\* |  |
| 10. | Очікувати завершення сортування\* в T4 |  |
| 9. | Копіювати *a2=a, F2=F, m2=m, Y2=Y, S2=S* | КД |
| 10. | Обчислити |  |
| 11. | Сигнал T4 про завершення обчислення |  |

|  |  |  |
| --- | --- | --- |
|  | **Т6** | ТС КД |
| 1. | Ввести дані: *MT, D* |  |
| 2. | Сигнал решті потоків про введення даних |  |
| 3. | Очікувати введення даних в решті потоків |  |
| 4. | Копіювати *Z6=Z, R6=R* | КД |
| 5. |  |  |
| 6. |  |  |
| 7. |  |  |
| 8. |  |  |
| 9. | Очікувати завершення сортування в Т5 |  |
| 10. |  |  |
| 9. | Сигнал T4 про завершення сортування\* |  |
| 10. | Очікувати завершення сортування\* в T4 |  |
| 9. | Копіювати *a6=a, F6=F, m6=m, Y6=Y, S6=S* | КД |
| 10. | Обчислити |  |
| 11. | Сигнал T4 про завершення обчислення |  |

|  |  |  |
| --- | --- | --- |
|  | **Тi, де** | ТС КД |
| 1. | Ввести дані:   * Якщо i=1: *Z, ME* * Якщо i=3: *MX, C* * Якщо i=5: *R, MM* |  |
| 2. | Сигнал решті потоків про введення даних |  |
| 3. | Очікувати введення даних в решті потоків |  |
| 4. | Копіювати *Zi=Z, Ri=R* | КД |
| 5. |  |  |
| 6. |  |  |
| 7. |  |  |
| 8. |  |  |
| 9. | Сигнал Т{i+1} про завершення сортування |  |
| 10. | Очікувати завершення сортування\* в T4 |  |
| 11. | Копіювати *ai=a, Fi=F, mi=m, Yi=Y, Si=S* | КД |
| 12. | Обчислити |  |
| 13. | Сигнал Т4 про завершення обчислення |  |

### 3. Структурна схема взаємодії потоків

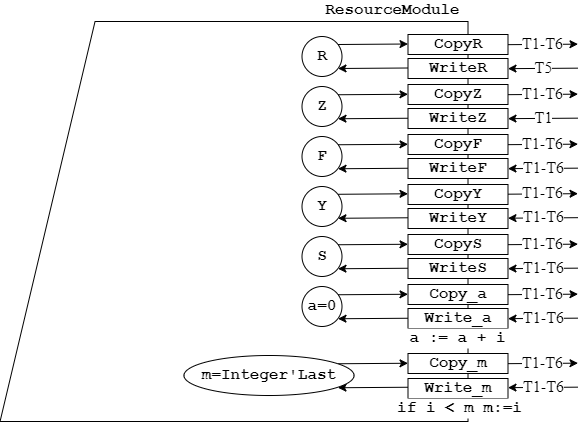


Рис 3.1 Структурна схема монітора для вирішення задачі взаємного виключення

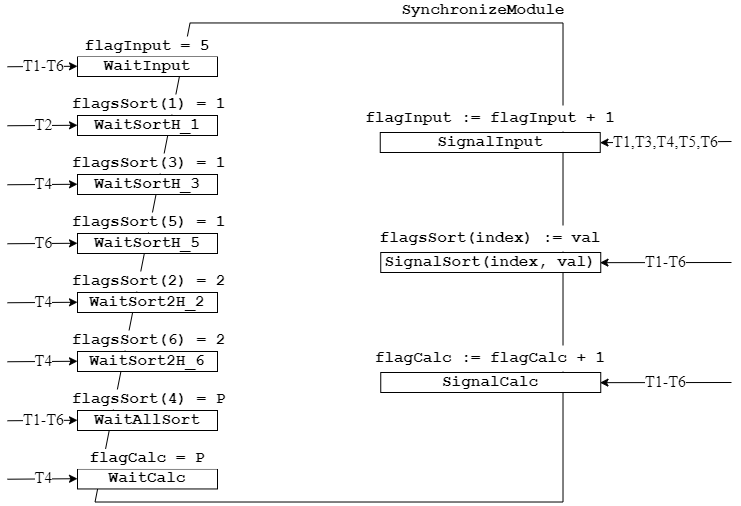


Рис 3.2 Структурна схема монітора для вирішення задачі синхронізації

### 4. Лістинг програми

#### main.adb

with Data; use Data;

with Ada.Text\_IO, Ada.Integer\_Text\_IO, Ada.Calendar, Ada.Float\_Text\_IO;

use Ada.Text\_IO, Ada.Integer\_Text\_IO, Ada.Calendar, Ada.Float\_Text\_IO;

procedure Main is

task type CalculateTask(taskNumber: Integer);

type CalculateTaskPointer is access CalculateTask;

type TasksArray is array(1..P) of CalculateTaskPointer;

type FlagArray is array (1..P) of Integer;

type indexes is range 1..6;

X, Z, R, D, C, T, F, Y: Vector := new Vector\_t;

MX, MC, MM, ME, MT: Matrix := new Matrix\_t;

tArray: TasksArray;

ch : Character;

Res: Matrix := new Matrix\_t;

StartTime, FinishTime: Time;

DiffTime: Duration;

-----------------------------------------------------------

--------------DESCRIPTION OF PROTECTED MODULES-------------

-----------------------------------------------------------

protected ResourceModule is

procedure WriteR(V: in Vector);

procedure WriteZ(V: in Vector);

procedure WriteF(V: in Vector);

procedure WriteY(V: in Vector);

procedure WriteS(V: in Vector);

procedure Write\_a(i: in Integer);

procedure Write\_m(i: in Integer);

function CopyR return Vector;

function CopyZ return Vector;

function CopyF return Vector;

function CopyY return Vector;

function CopyS return Vector;

function Copy\_a return Integer;

function Copy\_m return Integer;

private

Z, F, Y, S: Vector;

a: Integer := 0;

m: Integer := Integer'Last;

end ResourceModule;

protected SynchronizeModule is

procedure SignalInput;

procedure SignalSort(index, val: Integer);

procedure SignalCalc;

entry WaitInput;

entry WaitSortH\_1;

entry WaitSortH\_3;

entry WaitSortH\_5;

entry WaitSort2H\_2;

entry WaitSort2H\_6;

entry WaitAllSort;

entry WaitCalc;

private

flagInput: Integer := 0;

flagsSort: FlagArray := (0, 0, 0, 0, 0, 0);

flagCalc: Integer := 0;

end SynchronizeModule;

-----------------------------------------------------------

----------------BODIES OF PROTECTED MODULES----------------

-----------------------------------------------------------

protected body ResourceModule is

procedure WriteR(V: in Vector) is

begin

R := V;

end WriteR;

procedure WriteZ(V: in Vector) is

begin

Z := V;

end WriteZ;

procedure WriteF(V: in Vector) is

begin

F := V;

end WriteF;

procedure WriteY(V: in Vector) is

begin

Y := V;

end WriteY;

procedure WriteS(V: in Vector) is

begin

S := V;

end WriteS;

procedure Write\_a(i: in Integer) is

begin

a := a + i;

end Write\_a;

procedure Write\_m(i: in Integer) is

begin

if i < m then

m:=i;

end if;

end Write\_m;

function CopyR return Vector is

begin

return R;

end CopyR;

function CopyZ return Vector is

begin

return Z;

end CopyZ;

function CopyF return Vector is

begin

return F;

end CopyF;

function CopyY return Vector is

begin

return Y;

end CopyY;

function CopyS return Vector is

begin

return S;

end CopyS;

function Copy\_a return Integer is

begin

return a;

end Copy\_a;

function Copy\_m return Integer is

begin

return m;

end Copy\_m;

end ResourceModule;

protected body SynchronizeModule is

procedure SignalInput is

begin

flagInput := flagInput + 1;

end SignalInput;

procedure SignalSort(index, val: Integer) is

begin

flagsSort(index) := val;

end SignalSort;

procedure SignalCalc is

begin

flagCalc := flagCalc + 1;

end SignalCalc;

entry WaitInput

when flagInput = 5 is

begin

null;

end WaitInput;

entry WaitSortH\_1

when flagsSort(1) = 1 is

begin

null;

end WaitSortH\_1;

entry WaitSortH\_3

when flagsSort(3) = 1 is

begin

null;

end WaitSortH\_3;

entry WaitSortH\_5

when flagsSort(5) = 1 is

begin

null;

end WaitSortH\_5;

entry WaitSort2H\_2

when flagsSort(2) = 2 is

begin

null;

end WaitSort2H\_2;

entry WaitSort2H\_6

when flagsSort(6) = 2 is

begin

null;

end WaitSort2H\_6;

entry WaitAllSort

when flagsSort(4) = P is

begin

null;

end WaitAllSort;

entry WaitCalc

when flagCalc = P is

begin

null;

end WaitCalc;

end SynchronizeModule;

task body CalculateTask is

first, last: Integer;

Ri, Zi, Fi, Yi, Si: Vector;

ai: Integer := 0;

mi: Integer := Integer'Last;

a\_copy, m\_copy: Integer;

begin

Put\_Line("Task " & Integer'Image(taskNumber) & " started...");

first := GetFirst(taskNumber);

last := GetLast(taskNumber);

-- Input

if taskNumber = 1 then

InputVector(Z);

InputMatrix(ME);

ResourceModule.WriteZ(Z);

SynchronizeModule.SignalInput;

elsif taskNumber = 3 then

InputMatrix(MX);

InputVector(C);

SynchronizeModule.SignalInput;

elsif taskNumber = 4 then

InputMatrix(MC);

SynchronizeModule.SignalInput;

elsif taskNumber = 5 then

InputVector(R);

InputMatrix(MM);

ResourceModule.WriteR(R);

SynchronizeModule.SignalInput;

elsif taskNumber = 6 then

InputVector(D);

InputMatrix(MT);

SynchronizeModule.SignalInput;

end if;

SynchronizeModule.WaitInput;

Ri := ResourceModule.CopyR;

Zi := ResourceModule.CopyZ;

-- ai = D\*C; mi = min(D); Y = Z\*MX; T = Z\*MC

for i in first..last loop

ai := ai + D(i) \* C(i);

if D(i) < mi then

mi := D(i);

end if;

F(i) := 0;

Y(i) := 0;

T(i) := 0;

for j in 1..N loop

F(i) := F(i) + Ri(j)\*MT(j)(i);

Y(i) := Y(i) + Zi(j)\*MX(j)(i);

T(i) := T(i) + Zi(j)\*MC(j)(i);

end loop;

end loop;

ResourceModule.WriteF(F);

ResourceModule.WriteY(Y);

ResourceModule.Write\_a(ai);

ResourceModule.Write\_m(mi);

-- Sort

SortH(T, first, last);

SynchronizeModule.SignalSort(taskNumber, 1);

if (taskNumber = 4) then

SynchronizeModule.WaitSortH\_3;

MergeSort(T, GetFirst(3), GetLast(3), last);

SynchronizeModule.WaitSort2H\_2;

MergeSort(T, GetFirst(1), GetLast(2), last);

SynchronizeModule.WaitSort2H\_6;

MergeSort(T, 1, last, N);

ResourceModule.WriteS(T);

SynchronizeModule.SignalSort(4, P);

elsif (taskNumber = 2) then

SynchronizeModule.WaitSortH\_1;

MergeSort(T, GetFirst(1), GetLast(1), last);

SynchronizeModule.SignalSort(2, 2);

elsif (taskNumber = 6) then

SynchronizeModule.WaitSortH\_5;

MergeSort(T, GetFirst(5), GetLast(5), last);

SynchronizeModule.SignalSort(6, 2);

end if;

SynchronizeModule.WaitAllSort;

-- Copy data

ai := ResourceModule.Copy\_a;

Fi := ResourceModule.CopyF;

mi := ResourceModule.Copy\_m;

Yi := ResourceModule.CopyY;

Si := ResourceModule.CopyS;

-- Calculate X=a\*F\*ME-m\*Y\*MM+S

for k in first..last loop

X(k) := 0;

for j in 1..N loop

X(k) := X(k) + ai\*Fi(j)\*ME(j)(k) - mi\*Yi(j)\*MM(j)(k);

end loop;

X(k) := X(k) + Si(k);

end loop;

--Output

if taskNumber = 4 then

SynchronizeModule.SignalCalc;

SynchronizeModule.WaitCalc;

if N <= 6 then

OutputVector(X);

end if;

FinishTime := Clock;

DiffTime := FinishTime - StartTime;

Put("Time = ");

Put(Float(DiffTime), 1);

Put\_Line("");

else

SynchronizeModule.SignalCalc;

end if;

Put\_Line("Task " & Integer'Image(taskNumber) & " finished");

end CalculateTask;

begin

Get(ch);

StartTime := Clock;

for i in 1..P loop

tArray(i) := new CalculateTask(i);

end loop;

end Main;

#### data.ads

package Data is

N: Integer := 6;

P: constant Integer := 6;

H: Integer := N / P;

type Vector\_t is array(1..N) of Integer;

type Vector is access Vector\_t;

type Matrix\_t is array(1..N) of Vector;

type Matrix is access Matrix\_t;

procedure InputVector(V: out Vector);

procedure InputMatrix(M: out Matrix);

procedure OutputVector(V : in Vector);

procedure SortH(V : in out Vector; first, last: Integer);

procedure MergeSort(V: in out Vector; first, firstEnd, last: Integer);

function GetFirst(taskNumber: Integer) return Integer;

function GetLast(taskNumber: Integer) return Integer;

end Data;

#### data.adb

with Ada.INTEGER\_TEXT\_IO; use Ada.INTEGER\_TEXT\_IO;

with Ada.Text\_IO; use Ada.Text\_IO;

package body Data is

procedure InputVector(V: out Vector) is

begin

for i in 1 .. N loop

V(i) := 1;

end loop;

end InputVector;

procedure InputMatrix(M: out Matrix) is

begin

for i in 1 .. N loop

M(i) := new Vector\_t;

for j in 1 .. N loop

M(i)(j) := 1;

end loop;

end loop;

end InputMatrix;

procedure OutputVector(V : in Vector) is

begin

for i in 1..N loop

Ada.Integer\_Text\_IO.Put ( V(i) );

end loop;

Put\_Line("");

end OutputVector;

procedure SortH(V : in out Vector; first, last: Integer) is

buf: Integer;

begin

for i in first..last loop

for j in i..last loop

if V(i) > V(j) then

buf := V(i);

V(i) := V(j);

V(j) := buf;

end if;

end loop;

end loop;

end SortH;

procedure MergeSort(V: in out Vector; first, firstEnd, last: Integer) is

res: Vector := new Vector\_t;

i1: Integer := first;

i2: Integer := firstEnd+1;

i: integer := 1;

begin

while (i1 <= firstEnd or i2 <= last) loop

if V(i1) < V(i2) then

res(i) := V(i1);

i1 := i1 + 1;

if (i1 > firstEnd) then

while (i2 <= last) loop

i := i+1;

res(i) := V(i2);

i2 := i2 + 1;

end loop;

end if;

else

res(i) := V(i2);

i2 := i2 + 1;

if (i2> last) then

while (i1 <= firstEnd) loop

i := i+1;

res(i) := V(i1);

i1 := i1 + 1;

end loop;

end if;

end if;

i := i+1;

end loop;

--Write result of sort to the input vector

i1 := 1;

for j in first..last loop

V(j) := res(i1);

i1 := i1+1;

end loop;

end MergeSort;

function GetFirst(taskNumber: Integer) return Integer is

begin

return (taskNumber - 1) \* H + 1;

end GetFirst;

function GetLast(taskNumber: Integer) return Integer is

begin

return taskNumber \* H;

end GetLast;

end Data;