

Lección 8: Sincronización por barrera

- Sincronización por barrera: esquema del problema
- Implementación mediante contadores compartidos
- Implementación mediante “Flags” y coordinadores
- Ejemplos

Sincronización por barrera

- Muchos problemas de cálculo se pueden resolver de acuerdo al siguiente esquema:
- Ejemplos:
 - procesamiento de imágenes
 - funciones definidas sobre mallas de puntos
 - resolución de sistemas de ecuaciones
 - problemas de optimización, etc.
- La ejecución de “esperar al resto de procesos” es una sincronización
- Este tipo se denomina “sincronización por barrera”

Mq solución no alcanzada
lanzar ***n*** procesos, cada uno
“con su parte de los datos”
esperar al resto de procesos
FMq

Contadores compartidos

- Una solución incompleta
(con una variable auxiliar para su verificación)

```
Vars cont:Ent:=0
```

```
P(i:1..n)::
```

```
Mq TRUE
```

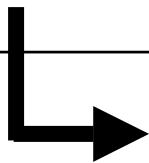
```
  código a
```

```
  ejecutar por P(i)
```

```
<cont:=cont+1>
```

```
<await cont=n>
```

```
FMq
```



```
FA(cont,1)
```

```
Mq cont<>n seguir FMq
```

```
Vars iter:vector(1..n) de
```

```
  Bool:=(1..n,FALSE)
```

```
  cont:Ent:=0
```

```
--CONT:  $\forall i \in \{1..n\}$ .
```

```
  (iter(j)  $\rightarrow$  cont=n)
```

```
P(i:1..n)::
```

```
Mq TRUE
```

```
  código a ejecutar por P(i)
```

```
<cont:=cont+1>
```

```
<await cont=n
```

```
  iter(i) :=TRUE>
```

```
FMq
```

```

task type fetchAndAdd is
  entry FA(inc: in integer;
          var: out integer);
  entry val(var: out integer);
end fetchAndAdd;

task body fetchAndAdd is
  valor: integer := 0;
begin
  loop
    select
      accept FA(inc: in integer;var:
               out integer) do
        var := valor;
        valor := valor+inc;
      end FA;
    or
      accept val(var: out integer) do
        var := valor;
      end val;
    or
      terminate;
    end select;
  end loop;
end fetchAndAdd;

```

```
cont: fetchAndAdd(0)
```

```
P(i:1..n)::
```

```
Mq TRUE
```

```
  código a
```

```
    ejecutar por P(i)
```

```
  <cont:=cont+1>
```

```
  <await cont=n>
```

```
FMq
```

Contadores compartidos

- Limitaciones de la solución propuesta:
 - el contador “cont” ha de ponerse a “0” antes de la siguiente iteración
 - usar dos contadores
 - dejar la tarea para uno de los procesos (el primero, por ejemplo)
 - la ejecución de “**Mq cont<>n . . .**” exige accesos continuos al contador
 - es necesaria una instrucción “FA”
 - puede que todos los procesos tengan que esperar por uno
 - accediendo constantemente a la misma variable

“Flags” y coordinadores

```
Vars finIter:vector(1..n) de Ent:=(1..n,0)
    seguir:vector(1..n) de Ent:=(1..n,0)
```

```
P(i:1..n)::
```

```
Mq TRUE
```

```
    código para P(i)
```

```
    finIter(i) := 1
```

```
    <await seguir(i)=1>
```

```
    seguir(i) := 0
```

```
FMq
```

```
Coordinador::
```

```
Mq TRUE
```

```
    Pt i:=1..n
```

```
        <await finIter(i)=1>
```

```
        finIter(i) := 0
```

```
    FPT
```

```
    Pt i:=1..n
```

```
        seguir(i) := 1
```

```
    FPT
```

```
FMq
```

“Flags” y coordinadores

- Las variables tipo “**finIter**” y “**seguir**” se denominan “**variables Flag**”
 - cada variable es “levantada” por un proceso para indicar algo al coordinador
- **Principios de la sincronización con variables Flag:**
 - el proceso que espera a que una variable esté abierta es el único proceso que la cierra
 - un “flag” no se puede abrir hasta que esté cerrado

Ejemplo

- Algoritmo de tratamiento paralelo de datos: itera, en paralelo, sobre las componentes de un array
- Problema: dado un vector de n enteros, obtener en otro las sumas parciales de sus prefijos

```
Consts n=....  
Tipos vectN=vector(1..n) de Ent  
  
Accion prefijos(E a:vect;S s:vect)  
--Q: TRUE  
--R:  $\forall \alpha \in \{1..n\}. s(\alpha) = \sum_{\beta \in \{1..\alpha\}} v(\beta)$ 
```


Una primera aproximación

	1	2	3	4	5	6
1	1	3	5	7	9	11
2	1	3	6	10	14	18
4	1	3	6	10	15	21

```
Accion prefijos (E a:vectN; S s:vectN)
  Vars aux:vectN
P(i:1..n)::
  Vars dist:Ent:=1
  s(i):=a(i)
  barrera para todos
  --SUMA:s(i)=a(i-dist+1)+..+a(i)
Mq dist<n
  aux(i):=s(i)
  barrera para todos
  Si (i-dist)>=1
    s(i):=aux(i-dist)+s(i)
  FSi
  barrera para todos
  dist:=2*dist
FMq
```

Sólo
notación