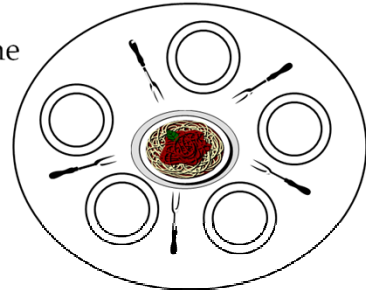


Lección 6: Ejemplos de programación con semáforos

- El problema de la cena de los filósofos
- El problema de los lectores y escritores
- Ejercicios
- Gestión de concurrencia mediante paso de testigo (implementación con semáforos)

El problema de la cena de los filósofos

- El problema:
 - cada filósofo, alternativamente, piensa y come
 - se necesitan dos tenedores para comer
 - el de la izquierda y el de la derecha
 - la cantidad de espaguetis ¡es infinita!
 - hay que evitar que se mueran de hambre
 - por la cabezonería de los filósofos
- Salta a la vista que:
 - dos filósofos vecinos no pueden comer a la vez
 - no más de dos filósofos pueden comer a la vez
- Objetivo: desarrollar un programa concurrente que simule el sistema



Process Filósofo::
loop forever
piensa
coge tenedores
come
deja tenedores

El problema de la cena de los filósofos

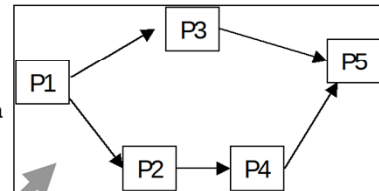
- La solución “natural” se bloquea
- Causas de bloqueo en sistemas que comparten recursos
 - un (unidad de) recurso sólo lo puede usar un proceso a la vez
 - el sistema es “no preemptive”
 - los estados de los procesos son del tipo “hold-and-wait”
 - aparece una espera circular
- ¿Cómo se puede corregir el sistema para que no haya bloqueo?

Ejercicios

- **Ejercicio 1:** Un grafo de precedencias de tareas es un grafo dirigido acíclico que establece un orden de ejecución de tareas de un programa concurrente, de manera que una tarea no puede empezar a ejecutarse hasta que las que sean anteriores en el grafo hayan terminado. Por ejemplo, si consideramos el grafo de la figura 1, la tarea P4 no puede empezar a ejecutarse hasta que P2 y P1 terminen, mientras que P5 no puede empezar hasta que P1, P2, P3 y P4 hayan terminado.

Asumamos que cada tarea ejecuta el siguiente código:

```
espera a que las tareas anteriores terminen
ejecuta el cuerpo de la tarea
avisa de su terminación a quien corresponda
```



- 1) Usando semáforos, escribir el programa concurrente correspondiente al diagrama de la figura
- 2) Esquematizar un método general de sincronización para un grafo de precedencias general. Calcular el número de semáforos que el método utilizaría. Este número se puede poner, por ejemplo, como función del número de tareas, de arcos, etc.

Ejercicios

- **Ejercicio 2:**
Considerar el siguiente programa concurrente. Calcular para él el conjunto de los posibles valores finales para la variable x

integer x := 0		
semaphore s1 := 1		
semaphore s2 := 0		
Process P1	Process P2	Process P3
wait(s2)	wait(s1)	wait(s1)
wait(s1)	x := x*x	x := x+3
x := 2*x	signal(s1)	signal(s2)
signal(s1)		signal(s1)

El paso del testigo

- Veamos una forma (eficiente) de implementar exclusiones mutuas y sincronización por condición con semáforos binarios
- Necesitamos:

$E_1: \langle S_i \rangle$ $E_2: \langle \text{await } B_j \quad S_j \rangle$

- Un semáforo binario para asegurar la ejecución en exclusión mutua: *mutex* (valor inicial 1)
- Para cada j, un semáforo binario b_j y un contador d_j , a cero

El paso del testigo

- Cambiar

```
E1: wait(mutex)
    Si
    AVISAR
```

```
E2: wait(mutex)
    if no Bj
        dj:=dj+1
        signal(mutex)
        wait(bj)
    Sj
    AVISAR
```

- donde

```
AVISAR:
--mutex.V=0
switch
  B1 y d1>0: d1:=d1-1;signal(b1)
  . . . .
  Bn y dn>0: dn:=dn-1;signal(bn)
  otherwise: signal(mutex)
```

Ejercicios

- **Ejercicio 3:** Implementar un semáforo general en base a uno (o varios) semáforo binarios
- **Ejercicio 4 (para después):** Problema de uso de recursos: n procesos compiten por el uso de un recurso, del que se disponen de k unidades. Las peticiones de uso del recurso son del tipo:
 - reserva(r)***: --necesito se me concedan, “de golpe”, r unidades del recurso
 - libera(l)***: --libero, “de golpe”, l unidades del recurso, que previamente se me habían concedido

El paso del testigo

- Aplicarlo al siguiente ejemplo

recurso r := 6

Process P

```
...  
r.reserva(2)  
...  
r.reserva(3)  
...  
r.libera(3)  
...  
r.libera(2)  
...
```

Process Q

```
...  
r.reserva(3)  
...  
r.libera(3)  
...
```

Ejercicios

- **Ejercicio 5:** Considerar el siguiente programa, que presenta un problema: nada impide que se use una variable en una expresión con un valor todavía indefinido.
- Se pide lo siguiente. Utilizando semáforos, completar el código de dicho programa de manera que nunca una variable sea usada en una expresión antes de que se le haya asignado un valor, ya sea mediante una instrucción leer o una asignación.
- Explicar cómo se generalizaría dicha solución para cualquier programa. ¿Podría un programa escrito de acuerdo a la propuesta anterior llegar a bloquearse? Si es así, escribir un ejemplo sencillo de bloqueo. Si no es posible, justificar "de manera convincente" por qué es así.

```
integer a,b,c,d
Process P1::
  leer(a);
  c := a+b
  b := 2*d

Process P2::
  leer(c)
  leer(d)
  b := a+d+c

Process P3::
  d := 2*a+c
```

El problema de los lectores/escritores

- **Problema:**

- dos tipos de procesos para acceder a una base de datos:
 - lectores: consultan la BBDD
 - escritores: consultan y modifican la BBDD
- cualquier transacción aislada mantiene la consistencia de la BBDD
- cuando un escritor accede a la BBDD, es el único proceso que la puede usar
- varios lectores pueden acceder simultáneamente

El problema de los lectores/escritores

```
semaphore permiso := 1
```

```
Process lector(i:1..n)::
```

```
loop forever
```

```
--protocolo entrada
```

```
read a data
```

```
--protocolo salida
```

```
Process escritor(i:1..m)::
```

```
loop forever
```

```
produce a data
```

```
--protocolo entrada
```

```
write the data
```

```
--protocolo salida
```

El problema de los lectores/escritores

```
semaphore permiso := 1
```

```
Process lector(i:1..n)::
```

```
loop forever
```

```
wait(permiso)
```

```
read a data
```

```
signal(permiso)
```

```
Process escritor(i:1..m)::
```

```
loop forever
```

```
produce a data
```

```
wait(permiso)
```

```
write the data
```

```
signal(permiso)
```

El problema de los lectores/escritores. Prioridad

```
Process Lector(i:1..n)::
```

```
loop
```

```
....
```

```
if escritor escribiendo OR  
                        esperando  
    esperarParaLeer
```

```
end if
```

```
avisar "sig." lector espera  
leeBaseDatos
```

```
if último AND escritor  
                esperando
```

```
    avisarle
```

```
end if
```

```
....
```

```
end loop
```

```
Process Escritor(i:1..m)::
```

```
loop
```

```
....
```

```
if escritor escribiendo OR  
    leyendo  
    esperarParaEsc
```

```
end if
```

```
escribeBaseDatos
```

```
if escritor esperando
```

```
    avisarle
```

```
else if lector esperando
```

```
    avisarle
```

```
end if
```

```
end if
```

```
....
```

```
end loop
```

El pro

```
integer numLec:=0, numLE:=0, numEE:=0
boolean escribiendo := false
semaphore puedesLeer:=0, puedesEscribir:=0
```

```
Process Lector(1,1,1,1)
loop
  ....
  if escritor escribiendo OR
    esperando
    esperarParaLeer
  end if
  avisar "sig." lector espera
  leeBaseDatos
  if último AND escritor
    esperando
    avisarle
  end if
  ....
end loop
```

```
Process Escritor(1,1,1,1)
loop
  ....
  if escritor escribiendo OR
    leyendo
    esperarParaEsc
  end if
  escribeBaseDatos
  if escritor esperando
    avisarle
  else if lector esperando
    avisarle
  end if
  end if
  ....
end loop
```

El pro

```
integer numLec:=0, numLE:=0, numEE:=0
boolean escribiendo := false
semaphore puedesLeer:=0, puedesEscribir:=0
semaphore mutex:=1
```

Process L
loop

```
.....
if escritor escribiendo OR
    esperando
    esperarParaLeer
end if
avisar "sig." lector espera
leeBaseDatos
if último AND escritor
    esperando
    avisarle
end if
.....
```

end loop

loop

```
.....
if escritor escribiendo OR
    leyendo
    esperarParaEsc
end if
escribeBaseDatos
if escritor esperando
    avisarle
else if lector esperando
    avisarle
end if
end if
.....
```

end loop


```
integer numLec:=0, numLE:=0, numEE:=0
boolean escribiendo := false
semaphore mutex:=1, puedesLeer:=0, puedesEscribir:=0
```

```
Process Lector(i:1..m)::
```

```
loop
```

```
....
```

```
wait(mutex)
```

```
if escribiendo OR numEE>0
```

```
    numLE := numLE+1
```

```
    signal(mutex)
```

```
    wait(puedesLeer)
```

```
    numLE := numLE-1
```

```
end if
```

```
numLec := numLec+1
```

```
if numLE>0
```

```
    signal(puedesLeer)
```

```
else
```

```
    signal(mutex)
```

```
end if
```

```
end loop
```

Prioridad a
escriitores

```
....
```

```
leeBaseDeDatos
```

```
wait(mutex)
```

```
numLec := numLec-1
```

```
if numLec=0 AND numEE>0
```

```
    signal(puedesEscribir)
```

```
else
```

```
    signal(mutex)
```

```
end if
```

```
....
```

```
...
Process Escritor(i:1..n)::
loop
```

```
....
wait(mutex)
if numLec>0 OR escribiendo
  numEE := numEE+1
  signal(mutex)
  wait(puedesEscribir)
  numEE := numEE-1
end if
escribiendo := TRUE
signal(mutex)
escribirBBDD
```

```
end loop
```

esritores

Prioridad a
esritores

```
....
wait(mutex)
if numEE>0
  signal(puedesEscribir)
else escribiendo := FALSE
  if numLE>0
    signal(puedesLeer)
  else
    signal(mutex)
  FSi
end if
```

```
integer numLec:=0, numLE:=0, numEE:=0
boolean escribiendo := false
semaphore mutex:=1, puedesLeer:=0, puedesEscribir:=0
```

```
Process Lector(i:1..m)::
```

```
loop
```

```
....
```

```
wait(mutex)
```

```
if escribiendo OR numEE>0
```

```
  numLE := numLE+1
```

```
  signal(mutex)
```

```
  wait(puedesLeer)
```

```
  numLE := numLE-1
```

```
end if
```

```
numLec := numLec+1
```

```
AVISAR()
```

```
end loop
```

Prioridad a
escriitores

```
....
```

```
leeBaseDeDatos
```

```
wait(mutex)
```

```
numLec := numLec-1
```

```
AVISAR()
```

```
....
```



```
...
Process Escritor(i:1..n)::
loop
```

```
....
wait(mutex)
if numLec>0 OR escribiendo
  numEE := numEE+1
  signal(mutex)
  wait(puedesEscribir)
  numEE := numEE-1
end if
escribiendo := TRUE
signal(mutex)
escribirBBDD
```

```
end loop
```

esritores

Prioridad a
esritores

```
....
wait(mutex)
if numEE>0
  signal(puedesEscribir)
else escribiendo := FALSE
  if numLE>0
    signal(puedesLeer)
  else
    signal(mutex)
  FSi
end if
```

Ejercicios

- **Ejercicio 6:** Plantéese una solución al problema de los filósofos de manera que cada filósofo toma los dos tenedores simultáneamente, y también los deja simultáneamente
 - Nótese que de esta manera se evitan los estados de «hold-and-wait» y, por tanto, no puede haber estados de bloqueo
-

Ejer

Ejercicio 2 (2.0 ptos.)

El listado que aparece a continuación corresponde a un programa concurrente en el que nP procesos permutan valores de un vector de enteros. Se pide completar el código, utilizando semáforos, de manera que no haya interferencias entre los procesos en el acceso a las componentes del vector. No es aceptable una secuencialización total del programa.

```
-----  
      constant integer n := ... --n>0  
                nP := ... --nP>0  
      integer array[1..n] datos := ... --lo que sea  
  
process P(i: 1..nP)::  
  integer i1,i2,temp  
  
  loop forever  
    get(i1,i2) --1<=i1<=n AND 1<=i2<=n  
    temp := datos[i1]  
    datos[i1] := datos[i2]  
    datos[i2] := temp  
  end loop  
end process  
-----
```