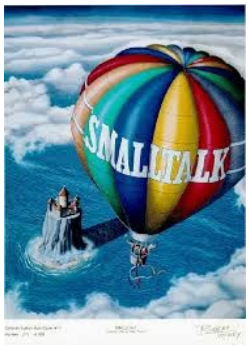
Departamento de Programación
Facultad de Informática
Universidad Nacional del Comahue

Programación Concurrente



Instrumentos de la concurrencia

Ejemplo

- En una rotisería

Rotisería

Chef

Comida

Cliente

Ejemplo

Rotisería
- comida Comida - chef Chef - micliente Cliente
Rotiseria (String nombreChef) +getComida() : Comida +getChef( ): Chef +getCliente(): Cliente +setComida(Comida c): void

Chef
- nroPlato int - rotiseria Rotiseria - nombre String
Chef (Rotisería roti, String nomChef) +run() : void

Cliente
- rotiseria Rotiseria - nombre String
Cliente (Rotiseria roti, String nomb) +toString() : String +run() : void

Comida
- nroOrden int Comida (int nro)
+toString() : String +getComida() : ordenNro

Ejemplo: Con bloqueos

Thread A

```
syncronized (this) {
  while comida != null{
   //hasta que el cliente
   // se lleven la comida
      this.wait();
      }
  }
 …..
syncronized (cliente) {
   // Creo mas comida
   // Le indico a la rotisería
   cliente.notify()
   }
```

Thread B

```
syncronized (this) {
  while comida == null{
      this.wait();
      }
  }
,,,
syncronized (chef) {
      vacio comida = null;
      chef.notify()
      }
```

Ejemplo: bloquear cliente y chef

Constructor

Crea el semáforo
Crea el cliente
Crea el chef

Crea los 2 threads
Comienza

Rotisería

- comida Comida
- chef Chef
- micliente Cliente
-semaforo

Atención la comida la tiene la rotisería
por más que la haga el chef y se la lleve el cliente

Ejemplo: Semáforo

```
run
MIENTRAS (no cierre la cocina)
 SI (no hay comida) ENTONCES
     semaforo bloquea

     Creo comida
     Espero un poco

     semaforo libera
   }
```

```
run
MIENTRAS (no cierre la cocina)
 SI (hay comida) ENTONCES
     semaforo bloquea

     Me llevo la comida
     Espero un poco

     semaforo libera
   }
```

Semáforos en Smalltalk

```
ALGORITMO proceso2
   REPETIR
         semaforo wait
         //código sección crítica
         semaforo signal
   HASTA …..
FIN ALGORITMO proceso2
```

```
ALGORITMO proceso1
   REPETIR
         semaforo wait
         //código de la sección crítica
         semaforo signal
   HASTA …..
FIN ALGORITMO proceso1
```

Semáforos en Smalltalk

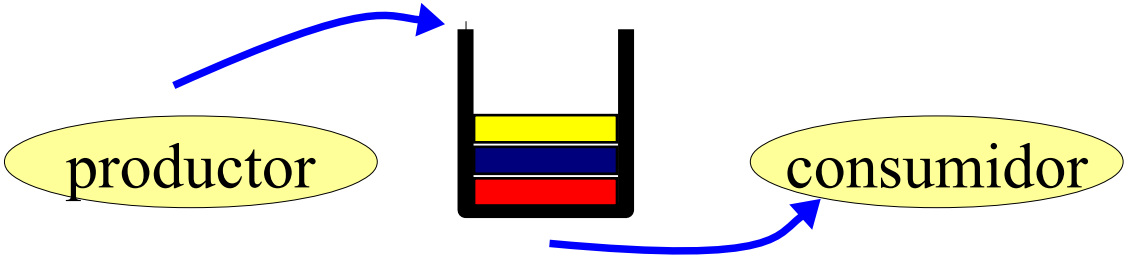
```
|semaforo valor proc1 proc2 |

semaforo := Semaphore new signal.
valor:= 10.
proc1 := [[valor < 500] whileTrue:   [
        semaforo wait.
        valor := valor + 5.
        semaforo signal.
        Transcript show: 'proc1- = ',valor asString; cr.].].

proc2 :=[ [valor < 500] whileTrue:  [
        semaforo wait.
       Transcript show: 'proc2 = ',valor asString; cr.
       valor := valor + 100.
       semaforo signal.].].

Transcript show: '--COMIENZO SIMULACION---';cr.
proc1 fork.
proc2 fork.
Transcript show: '--FIN DE SIMULACION--'; cr.
```

Productores/Consumidor - Semaforos

Supongamos una única producción

- Dos productores generan sus datos
- El consumidor puede tomar un dato sólo cuando hay
- Todo lo que se produce debe ser consumido
- DOS SEMAFOROS

El consumidor no debe consumir más rápido de lo que produce los productores



Productores/Consumidores - Sm

Productor1 → produce/ pone valor “primero”
Productor2 → produce/ pone valor “segundo”

Consumidor → obtiene el valor existente

Semaforo A
Semáforo B

Variable Compartida
Cambia e/Primero y Segundo

“poner”

```
putBloque :=   [:variable | semA wait.  varCompartida := variable. semB signal].
```

“obtener”

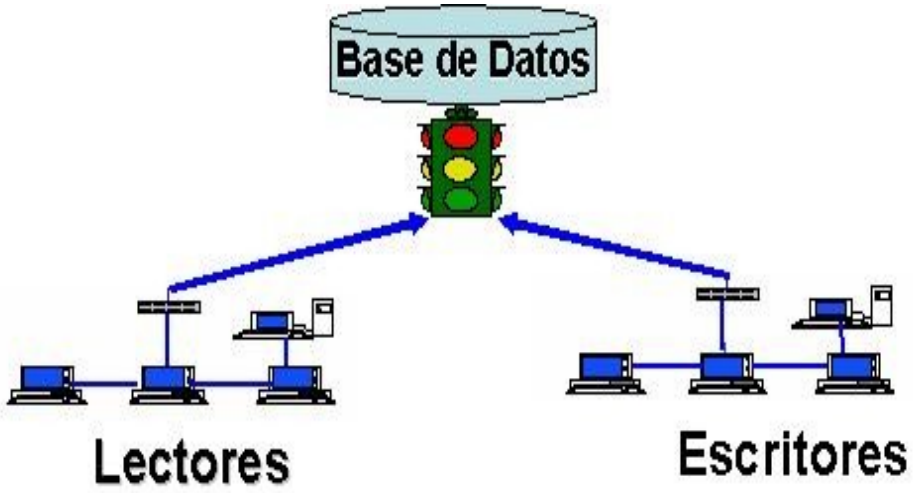
```
getBloque :=   [:variable | semB wait.  variable :=varCompartida . semA signal].
```

semaforoA := Semaphore new signal. **“Ej. SM 2 Productores – 1 Consumidor”**
semaforoB := Semaphore new.
varCompartida := nil.

```
putBloque :=   [:var | semaforoA wait.  varCompartida := var. semaforoB signal].
getBloque := [| var |  semaforoB wait.  variable := varCompartida. semaforoA signal. var].

prod1 := [[| variable | (Delay forSeconds: 3) wait.
          variable := 'Primero '. putBloque value: variable.
          Transcript show: 'pongo ',variable; cr. ] repeat].

prod2 := [ [| variable |(Delay forSeconds: 2) wait.
        variable := 'Segundo '.  putBloque value: variable.
       Transcript show: 'pongo ',variable; cr. ] repeat].

consumidor := [ [| variable |(Delay forSeconds: 1) wait.
        variable := getBloque value.  Transcript show: 'obtengo ',variable; cr. ] repeat].

        Transcript show: '--SIMULACION-------';cr.
        proc1 := prod1 fork.   proc2 := prod2 fork.  proc3 := consumidor fork.
        (Delay forSeconds: 10) wait.
        proc1 terminate. proc2 terminate.  proc3 terminate.
        Transcript show: '----------FIN DE SIMULACION-------'; cr.
```

Lectores/escritor en Java

Dos grupos de procesos (Lectores/escritor) que utilizan el mismo recurso (libro)



- Un grupo lectores, sólo leen los datos.
- El escritor escriben.
- Varios lectores pueden acceder simultáneamente al libro
- Se debe evitar que accedan simultáneamente un proceso escritor y cualquier otro proceso.

Lectores/escritores con BD

- Ejemplo: Un grupo de lectores/escritores quieren tener acceso a una bae de datos.
- Cuando un escritor quiere acceder a la base de datos ésta debe estar desocupada.

Utilizar:
-Semáforo,
-Lector,
-Escritor

- Llega un lector:
  - Si hay uno o varios lectores dentro de la BD, el lector puede acceder a la misma.
  - Si hay un escritor, entonces el lector deberá esperar a que el escritor acabe para entrar
- Llega un escritor:
  - Si hay uno o varios lectores dentro de la B.D., el escritor deberá esperar hasta que todos los lectores terminen.
  - Si hay un escritor, entonces el escritor que quiere entrar deberá esperar a que el escritor acabe con su tarea o finalice.

Lectores/Escritor en Java

DisparaLecturas

//Crea el semáforo
//Crea libro con el semafor
//Crea los lectores y escritor
//Crea los threads de lectores
//Crea el thread de escritor

Escritor
- nombre - libro - hojasEscritas
Escritor (String nom, Libro l) +run() : void

Libro
- totalPaginas - pagina - cantLectores - camtEstanLeyendo
Libro(int pags, int cant, Semaphore s) +leer() +terminaLeer() +escribe() +terminaEscribir() +finalizado() +hayEscrito()

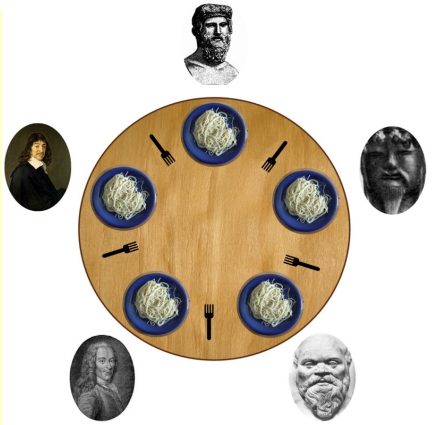
Lector
- libro - nombre -id -ciclo
Lector (String nom, int id, Libro l) +terminaLectura() : booleano +run() : void

Problema de sincronización:

Cena de filósofos

Evitar el interbloqueo o deadlock

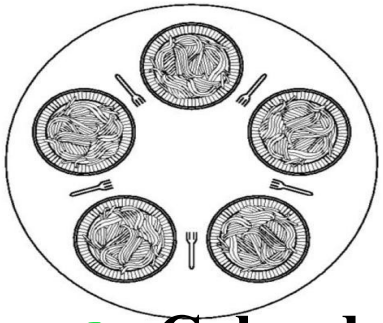
Cinco filósofos se sientan alrededor de una mesa y pasan su vida cenando y pensando. Cada filósofo tiene un plato de fideos y un tenedor a la izquierda de su plato. Para comer los fideos son necesarios dos tenedores y cada filósofo sólo puede tomar los que están a su izquierda y derecha.



Si cualquier filósofo toma un tenedor y el otro está ocupado, se quedará esperando, con el tenedor en la mano, hasta que pueda tomar el otro tenedor, para luego empezar a comer.

Si dos filósofos adyacentes intentan tomar el mismo tenedor a una vez, ambos compiten por tomar el mismo tenedor, y uno de ellos se queda sin comer.

Si todos los filósofos toman el tenedor que está a su derecha al mismo tiempo, entonces todos se quedarán esperando eternamente. Entonces los filósofos se morirán de hambre → ***Bloqueo mutuo o deadlock***

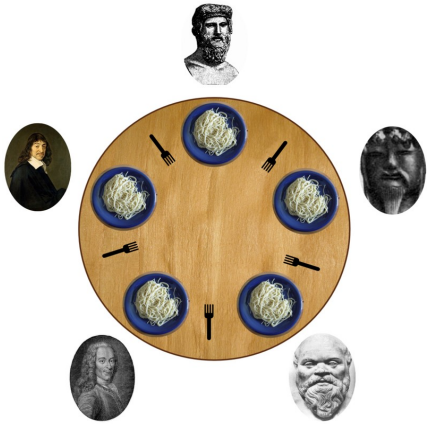


Cena de filósofos

- **Colas de tenedores:**
  - Cuando un filósofo quiere comer se pone en la cola de los dos tenedores que necesita.
  - Cuando un tenedor está libre lo toma.
  - Cuando toma los dos tenedores, come y deja libre los tenedores.
  - Cada vez que un filósofo tiene un tenedor espera un tiempo aleatorio para conseguir el segundo tenedor.
  - Si en ese tiempo no queda libre el segundo tenedor, suelta el que tiene y vuelve a ponerse en cola para sus dos tenedores.
  - El tiempo de espera debe ser aleatorio para evitar deadlock

Cena de filósofos

Filosofo
- tenedor derecho (semáforo) - tenedor izquierdo (semáforo) - posicionDerecho - posicionIzquierdo - nombre - id
Filosofo (posD, posI, nomb, id, semaforo) +pensar() : Comida +comer( ): Chef **+run()**: Cliente



Cena
- tenedores Semaphore[5] - filosofos f1, f2, f3, f4, f5
// Se crean los threads //Comienzan los threads

Problema de sincronización:

Barbero durmiente

En una barbería en la que trabaja un barbero que tiene un único sillón de barbero y varias sillas para esperar.

- Cuando no hay clientes, el barbero se sienta en una silla y se duerme.

-Cuando llega un nuevo cliente,

-si el barbero duerme: despierta al barbero o

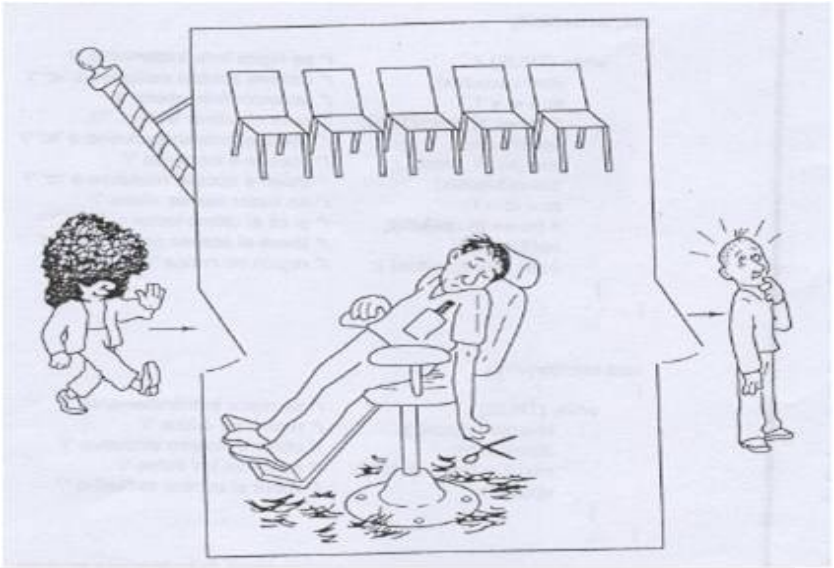
-si el barbero está afeitando a otro cliente: se sienta en una silla

(si todas las sillas están ocupadas por clientes esperando, se va).

La solución implica el uso de semáforos y exclusión mutua)

Barbero durmiente

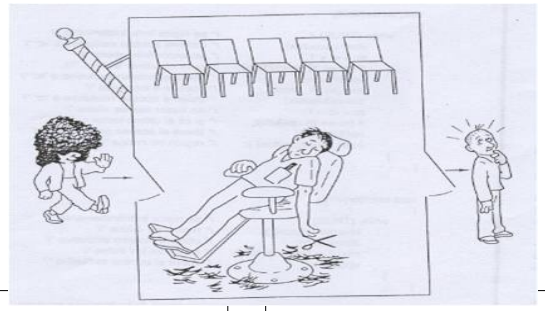
- No hay clientes → el barbero se sienta en la silla de barbero y se duerme
- Llega un cliente → despierta al barbero
- Llegan más clientes mientras el barbero está atendiendo, se sientan (si hay sillas vacías) o bien, salen de la peluquería (si están todas ocupadas)



Programar al barbero y a los clientes sin caer en condiciones de competencia

Objetos en la barbería: Barbero, Silla de barbero, N – sillas de clientes

Barbero dormilón - semáforo



Wait

- **Cliente**: Espera hasta que haya sitio para entrar en la peluquería
- **Cliente**: Espera una silla vacía
- **Cliente**: Espera hasta que la silla del barbero esté vacía
- **Barbero:** Espera hasta que el cliente esté en la silla
- **Cliente**: Espera hasta que se complete el corte de cabello
- **Barbero:** Espera hasta que el cliente se levante de la silla

Signal

- **Cliente**: Sale → avisa a un cliente que espera que puede entrar
- **Cliente**: Abandona la silla → le avisa a un cliente que espera silla
- **Barbero**: Avisa cuando su silla está vacía
- **Cliente**: avisa al barbero que ya está en la silla
- **Barbero**: Avisa al cliente cuando terminó de cortar el cabello
- **Cliente**: avisa al barbero cuando se levanta de la silla