



CONCURRENCIA

Semáforos

Guillermo Román Díez
`groman@fi.upm.es`

Universidad Politécnica de Madrid

Curso 2016-2017



PROBLEMAS DE LA ESPERA ACTIVA

- ▶ Los algoritmos de espera activa pueden garantizar la exclusión mutua, pero... :
 - ▶ Son difíciles de implementar
 - ▶ Son difíciles de probar su corrección
 - ▶ Malgastan CPU
 - ▶ Dependen de la arquitectura donde ejecutan
 - ▶ Son difíciles de escalar a diferentes tipos de procesos
 - ▶ ...

Es necesario encontrar una solución mejor para implementar la exclusión mutua



Semáforo

“es un tipo abstracto de datos que permite detener o dejar “pasar” un proceso dependiendo del valor de su contador interno”

- ▶ Detiene los threads (no hace espera activa)
- ▶ Permite desbloquear uno de los hilos que se encuentren bloqueados
- ▶ Se basa en `test_and_set`
- ▶ Es de más alto nivel que la espera activa $\Rightarrow$ simplifica los protocolos de entrada y salida en la sección crítica
- ▶ Resuelve fácilmente la exclusión mutua
- ▶ Permite verificar más fácilmente la corrección de las soluciones propuestas



- ▶ Podríamos ver cierta analogía con un semáforo de tráfico (un poco lejana...)
 - ▶ Si está “abierto” permite el paso del thread
 - ▶ Si está “cerrado” detiene el thread
- ▶ Tiene 1 atributo contador de tipo entero
 - ▶ Se usa para decidir si un proceso puede “pasar” o no
 - ▶ El valor del contador se inicializa en el constructor
- ▶ Se puede inicializar a cualquier valor mayor o igual que 0
- ▶ Tiene 2 operaciones:
 - ▶ P() (await, acquire)
 - ▶ V() (signal, release)



Semaphore s...

```
void await(){  
    if(s.contador>0)  
        s.contador --;  
    else  
        block(); // Duerme el thread actual  
}
```

- ▶ Cuando el contador tiene un valor mayor que 0, únicamente se decrementa el contador
- ▶ Cuando el contador vale 0 entonces el proceso se queda bloqueado



Semaphore s...

```
void signal(){
    if(!procesos_bloqueados)
        s.contador ++;
    else
        unblock(t); // Despierta un thread t
                    // que estuviera bloqueado
                    // (no-determinista)
}
```

- ▶ Cuando no hay procesos bloqueados se incrementa el contador
- ▶ Cuando hay procesos bloqueados se desbloquea uno de esos procesos



- ▶ La decisión del thread que se **despierta** en un semáforo es **no-determinista**
 - ▶ Por esto, a nivel teórico podría darse el caso de que un thread sufra inanición
- ▶ En las implementaciones se suele hacer en orden de llegada
 - ▶ La implementación de Java tiene un flag para garantizar justicia
 - ▶ Nuestra implementación de *cclib* es “justa”
- ▶ Para razonar sobre el comportamiento del semáforo debemos suponer que cualquier hilo que esté bloqueado puede ser desbloqueado en un `signal`



¿VERDADERO O FALSO?

Un semáforo se puede inicializar con un valor del contador negativo



¿VERDADERO O FALSO?

X Un semáforo se puede inicializar con un valor del contador negativo



¿VERDADERO O FALSO?

X Un semáforo se puede inicializar con un valor del contador negativo

Un semáforo inicializado con un valor mayor que 0 puede tener $s.\text{contador} < 0$



¿VERDADERO O FALSO?

- X Un semáforo se puede inicializar con un valor del contador negativo
- X Un semáforo inicializado con un valor mayor que 0 puede tener $s.\text{contador} < 0$



¿VERDADERO O FALSO?

- X Un semáforo se puede inicializar con un valor del contador negativo
- X Un semáforo inicializado con un valor mayor que 0 puede tener `s.contador < 0`
Si hay un thread bloqueado, entonces `s.contador == 0`



¿VERDADERO O FALSO?

- X Un semáforo se puede inicializar con un valor del contador negativo
- X Un semáforo inicializado con un valor mayor que 0 puede tener `s.contador < 0`
- ✓ Si hay un thread bloqueado, entonces `s.contador == 0`



¿VERDADERO O FALSO?

- X Un semáforo se puede inicializar con un valor del contador negativo
- X Un semáforo inicializado con un valor mayor que 0 puede tener `s.contador < 0`
- ✓ Si hay un thread bloqueado, entonces `s.contador == 0`
Si `s.contador == 0` implica que hay un thread bloqueado



¿VERDADERO O FALSO?

- X Un semáforo se puede inicializar con un valor del contador negativo
- X Un semáforo inicializado con un valor mayor que 0 puede tener `s.contador < 0`
- ✓ Si hay un thread bloqueado, entonces `s.contador == 0`
- X Si `s.contador == 0` implica que hay un thread bloqueado



¿VERDADERO O FALSO?

- X Un semáforo se puede inicializar con un valor del contador negativo
- X Un semáforo inicializado con un valor mayor que 0 puede tener `s.contador < 0`
- ✓ Si hay un thread bloqueado, entonces `s.contador == 0`
- X Si `s.contador == 0` implica que hay un thread bloqueado
Si `s.contador > 0` implica que no hay threads bloqueados



¿VERDADERO O FALSO?

- X Un semáforo se puede inicializar con un valor del contador negativo
- X Un semáforo inicializado con un valor mayor que 0 puede tener `s.contador < 0`
- ✓ Si hay un thread bloqueado, entonces `s.contador == 0`
- X Si `s.contador == 0` implica que hay un thread bloqueado
- ✓ Si `s.contador > 0` implica que no hay threads bloqueados



¿VERDADERO O FALSO?

- X Un semáforo se puede inicializar con un valor del contador negativo
 - X Un semáforo inicializado con un valor mayor que 0 puede tener `s.contador < 0`
 - ✓ Si hay un thread bloqueado, entonces `s.contador == 0`
 - X Si `s.contador == 0` implica que hay un thread bloqueado
 - ✓ Si `s.contador > 0` implica que no hay threads bloqueados
- A través de un semáforo se puede bloquear un thread distinto al que está ejecutando



¿VERDADERO O FALSO?

- X Un semáforo se puede inicializar con un valor del contador negativo
- X Un semáforo inicializado con un valor mayor que 0 puede tener `s.contador < 0`
- ✓ Si hay un thread bloqueado, entonces `s.contador == 0`
- X Si `s.contador == 0` implica que hay un thread bloqueado
- ✓ Si `s.contador > 0` implica que no hay threads bloqueados
- X A través de un semáforo se puede bloquear un thread distinto al que está ejecutando



¿VERDADERO O FALSO?

- X Un semáforo se puede inicializar con un valor del contador negativo
 - X Un semáforo inicializado con un valor mayor que 0 puede tener `s.contador < 0`
 - ✓ Si hay un thread bloqueado, entonces `s.contador == 0`
 - X Si `s.contador == 0` implica que hay un thread bloqueado
 - ✓ Si `s.contador > 0` implica que no hay threads bloqueados
 - X A través de un semáforo se puede bloquear un thread distinto al que está ejecutando
- A través de un semáforo se puede desbloquear un thread distinto al que está ejecutando



¿VERDADERO O FALSO?

- X Un semáforo se puede inicializar con un valor del contador negativo
- X Un semáforo inicializado con un valor mayor que 0 puede tener `s.contador < 0`
- ✓ Si hay un thread bloqueado, entonces `s.contador == 0`
- X Si `s.contador == 0` implica que hay un thread bloqueado
- ✓ Si `s.contador > 0` implica que no hay threads bloqueados
- X A través de un semáforo se puede bloquear un thread distinto al que está ejecutando
- ✓ A través de un semáforo se puede desbloquear un thread distinto al que está ejecutando



¿VERDADERO O FALSO?

- X Un semáforo se puede inicializar con un valor del contador negativo
 - X Un semáforo inicializado con un valor mayor que 0 puede tener `s.contador < 0`
 - ✓ Si hay un thread bloqueado, entonces `s.contador == 0`
 - X Si `s.contador == 0` implica que hay un thread bloqueado
 - ✓ Si `s.contador > 0` implica que no hay threads bloqueados
 - X A través de un semáforo se puede bloquear un thread distinto al que está ejecutando
 - ✓ A través de un semáforo se puede desbloquear un thread distinto al que está ejecutando
- El valor máximo de `s.contador` no está acotado



¿VERDADERO O FALSO?

- X Un semáforo se puede inicializar con un valor del contador negativo
- X Un semáforo inicializado con un valor mayor que 0 puede tener `s.contador < 0`
- ✓ Si hay un thread bloqueado, entonces `s.contador == 0`
- X Si `s.contador == 0` implica que hay un thread bloqueado
- ✓ Si `s.contador > 0` implica que no hay threads bloqueados
- X A través de un semáforo se puede bloquear un thread distinto al que está ejecutando
- ✓ A través de un semáforo se puede desbloquear un thread distinto al que está ejecutando
- ✓ El valor máximo de `s.contador` no está acotado



INTERBLOQUEO (DEADLOCK)

Deadlock

“Describe una situación en la cual hay dos o más procesos bloqueados esperándose entre sí indefinidamente”

- ▶ Hay más de un proceso compitiendo por el mismo recurso y ambos están esperando a que el otro termine con el recurso, y ninguno de ellos lo hace



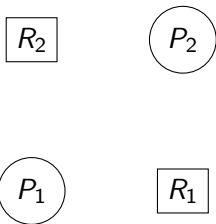
INTERBLOQUEO (DEADLOCK)

- ▶ Se pueden intentar detectar mediante un grafo donde:
 - ▶ Los recursos y los procesos son nodos
 - ▶ Cuando un proceso adquiere un recurso $\Rightarrow$ arista del recurso al proceso
 - ▶ Cuando un proceso solicita un recurso $\Rightarrow$ arista del proceso al recurso



INTERBLOQUEO (DEADLOCK)

- ▶ Se pueden intentar detectar mediante un grafo donde:
 - ▶ Los recursos y los procesos son nodos
 - ▶ Cuando un proceso adquiere un recurso $\Rightarrow$ arista del recurso al proceso
 - ▶ Cuando un proceso solicita un recurso $\Rightarrow$ arista del proceso al recurso



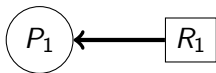


INTERBLOQUEO (DEADLOCK)

- ▶ Se pueden intentar detectar mediante un grafo donde:
 - ▶ Los recursos y los procesos son nodos
 - ▶ Cuando un proceso adquiere un recurso $\Rightarrow$ arista del recurso al proceso
 - ▶ Cuando un proceso solicita un recurso $\Rightarrow$ arista del proceso al recurso



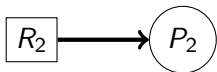
1) P_1 adquiere R_1





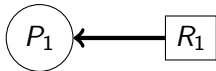
INTERBLOQUEO (DEADLOCK)

- ▶ Se pueden intentar detectar mediante un grafo donde:
 - ▶ Los recursos y los procesos son nodos
 - ▶ Cuando un proceso adquiere un recurso $\Rightarrow$ arista del recurso al proceso
 - ▶ Cuando un proceso solicita un recurso $\Rightarrow$ arista del proceso al recurso



1) P_1 adquiere R_1

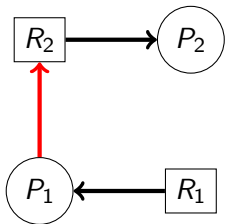
2) P_2 adquiere R_2





INTERBLOQUEO (DEADLOCK)

- ▶ Se pueden intentar detectar mediante un grafo donde:
 - ▶ Los recursos y los procesos son nodos
 - ▶ Cuando un proceso adquiere un recurso $\Rightarrow$ arista del recurso al proceso
 - ▶ Cuando un proceso solicita un recurso $\Rightarrow$ arista del proceso al recurso

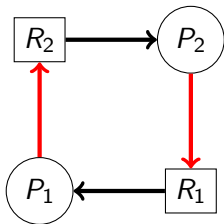


- 1) P_1 adquiere R_1
- 2) P_2 adquiere R_2
- 3) P_1 solicita R_2



INTERBLOQUEO (DEADLOCK)

- ▶ Se pueden intentar detectar mediante un grafo donde:
 - ▶ Los recursos y los procesos son nodos
 - ▶ Cuando un proceso adquiere un recurso $\Rightarrow$ arista del recurso al proceso
 - ▶ Cuando un proceso solicita un recurso $\Rightarrow$ arista del proceso al recurso



- 1) P_1 adquiere R_1
- 2) P_2 adquiere R_2
- 3) P_1 solicita R_2
- 4) P_2 solicita R_1

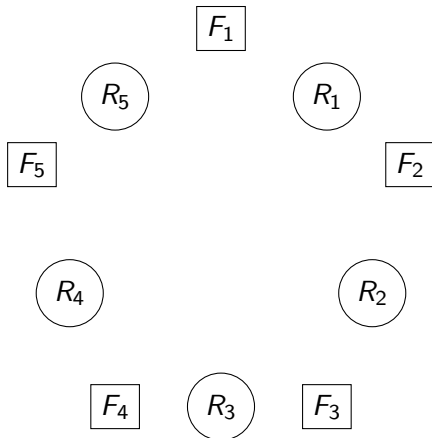


FILÓSOFOS CENANDO (DINING PHILOSOPHERS)



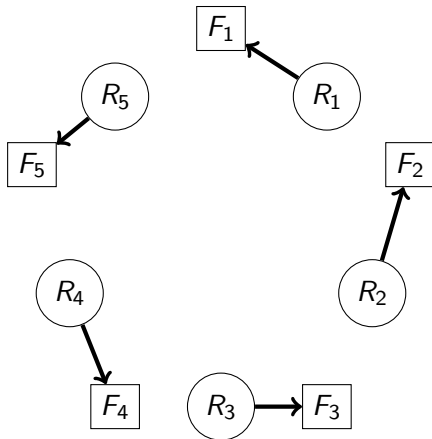


FILÓSOFOS CENANDO (DINING PHILOSOPHERS)





FILÓSOFOS CENANDO (DINING PHILOSOPHERS)





FILÓSOFOS CENANDO (DINING PHILOSOPHERS)

