Sincronización entre procesos (aka: semáforos)

Ignacio Vissani => Federico Raimondo => **Patricio Inzaghi**

DC - FCEyN - UBA

Sistemas Operativos, 2c-2015

¿Dónde estamos? ¿De dónde venimos?

Primero repasemos brevemente lo que vieron en la teórica (hace 30 minutos).

¿Qué es una *race condition*?

Defecto en un proceso, donde el resultado del mismo depende inesperadamente del orden en que se ejecuten ciertos eventos.

¿Cuál es el output de los siguientes procesos A y B corriendo simultáneamente y con memoria compartida?

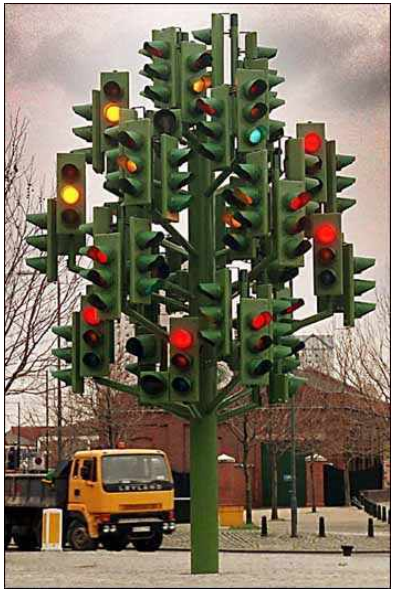
x comienza inicializado en 0.

A

```
x = x + 1;
print(x);
```

B

```
x = x + 1;
```



- ¿Qué es un semáforo?
  Es una variable (o tipo abstracto de datos) que permite controlar el acceso de múltiples procesos a un recurso común en un ambiente de programación paralela.
- ¿Es lo mismo que usar un entero y fijarme qué valor tiene?
  No, es escencial que las primitivas sobre semáforos sean atómicas.

Recordemos cuáles son las primitivas:

Primitivas

- `sem_create(int value)`: Devuelve un nuevo semáforo inicializado en `value`. (Otras formas: `Semaphore(value)`, `new Semaphore(value)`, etc.)
- `sem_wait(semaphore sem)`: Mientras el valor sea menor o igual a 0 se bloquea esperando un `signal`. Luego decrementa el valor de `sem`. (Otras formas: `wait(sem)`, `P(sem)`, `sem.wait()`, etc.)
- `sem_signal(semaphore sem, [int n = 1])`: Incrementa en uno el valor del semáforo `sem` y despierta a *alguno*[1] de los procesos que están esperando en ese semáforo. (Otras formas: `signal(sem, [n])`, `V(sem)`, `sem.signal([n])`, etc.)

[1] En general las bibliotecas de semáforos no garantizan en qué orden se despertará a los procesos que están esperando en un semáforo.

Ejercicio

Se tienen 3 procesos A, B y C. Construya el código con semáforos de manera tal que la secuencia sea ABC,ABC,ABC,...

Solución:

Uso 3 semáforos, sem_A, sem_B y sem_C. Sus valores de inicialización son:

sem_A = 1, sem_B = 0, sem_C = 0

```
while(true):
sem_A.wait()
// Sección crítica
A()
// Fin sección crítica
sem_B.signal()
```

```
while(true):
sem_B.wait()
// Sección crítica
B()
// Fin sección crítica
sem_C.signal()
```

```
while(true):
sem_C.wait()
// Sección crítica
C()
// Fin sección crítica
sem_A.signal()
```

Ejercicio

¿Y si quiero que la secuencia sea BCA,BCA,BCA,...?

Solución:

Cambio los valores de inicialización de los semáforos por
$sem\_A = 0$, $sem\_B = 1$, $sem\_C = 0$

Ejercicio

¿Y si quiero que la secuencia sea BBCA,BBCA,BBCA,...?

Solución:

Uso 3 semáforos, sem_A, sem_B y sem_C. Sus valores de inicialización son:

sem_A = 0, sem_B = 2, sem_C = 0

```
while(true):
sem_A.wait()
// Sección crítica
A()
// Fin sección crítica
sem_B.signal()
sem_B.signal()
```

```
while(true):
sem_B.wait()
// Sección crítica
B()
// Fin sección crítica
sem_C.signal()
```

```
while(true):
sem_C.wait()
sem_C.wait()
// Sección crítica
C()
// Fin sección crítica
sem_A.signal()
```

Ejercicio

Se tienen N procesos, $P_0, P_1, ..., P_{N-1}$ (donde N es un parámetro). Se los quiere sincronizar de manera que la secuencia de ejecución sea $P_i, P_{i+1}, ..., P_{N-1}, P_0, ..., P_{i-1}$

Solución:

Global

```
Semaphore semaforos[N];
for(int j = 0; j < N; j++)
  semaforos[j] = Semaphore(0);
semaforos[i] = Semaphore(1);
```

En cada P_i

```
semaforos[i].wait()
// Algo
semaforos[(i + 1)% n].signal()
```

Ejercicio

Suponga que se tienen N procesos P_i, cada uno de los cuales ejecuta un conjunto de sentencias a_i y b_i. Se los quiere sincronizar de manera tal que los b_i se ejecuten después de que se hayan ejecutado todos los a_i

Solución:

Global

```
mutex = Semaphore(1);
cuantosLlegaron = 0;
barrera = Semaphore(0);
```

y en cada P_i...

```
ai();
mutex.wait();
cuantosLlegaron++;
if (cuantosLlegaron == N)
  barrera.signal(N);
mutex.signal();

barrera.wait();

bi();
```

Ejercicio

En un sistema, para determinado recurso exclusivo se ha definido una política de acceso FIFO. Se tienen una serie de procesos que desean acceder a ese recurso. Escribir el código de sincronización de cada uno de los procesos para asegurar que el acceso respeta la política adoptada.

Tip para la solución

Usar una cola :)

Tip para la solución 2

Hay que garantizar acceso exclusivo a la cola.

Solución:

Global

```
Cola c = crearCola()
Semaphore mutex =
Semaphore(1)
```

P_i

```
Semaphore yo = Semaphore(0)
mutex.wait()
n = long(c)
c.push(yo)
mutex.signal()
if (n == 0)
  yo.signal()
yo.wait()
f() // Usar el recurso
mutex.wait()
c.pop()
if (!c.empty()):
  c.top().signal()
mutex.signal()
```

¿Cuándo está en deadlock un conjunto de procesos?

Deadlock

Situación en la que dos o más acciones en competencia están esperando mutuamente que la otra concluya, y entonces ninguna lo hace.

Condiciones (Coffman)

- Exclusión mutua.
- *Hold & Wait.*
- *No preemption.*
- Espera circular.

Ejemplos Visuales

Exclusión mutua:

Ejemplos Visuales

Hold & Wait:

Ejemplos Visuales

No Preemption:

Ejemplos Visuales

Espera circular:

Deadlock

So it begins, the great battle of our time.

Un ejemplo típico de deadlock es

P_1

```
sem1.wait();
sem2.wait();
// Sección crítica
sem2.signal();
sem1.signal();
```

P_2

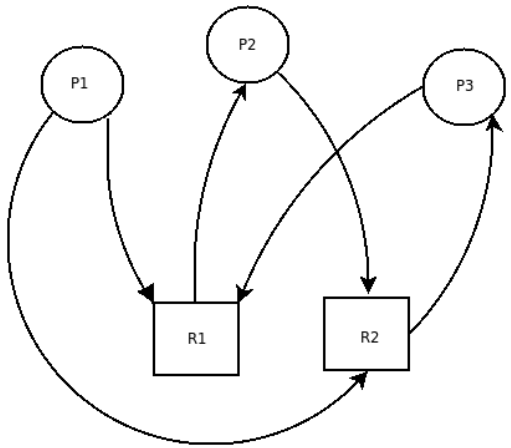
```
sem2.wait();
sem1.wait();
// Sección crítica
sem1.signal();
sem2.signal();
```

Ejercicio

En un sistema conviven 3 procesos y 2 recursos (R1 y R2) de uso exclusivo. Los 3 procesos necesitan tener los recursos a la vez para completar su ejecución. ¿Puede haber deadlock?

Solución:

Sí.



Ejercicio

¿Y si ahora R1 puede ser compartido por hasta tres procesos?

Solución:
No.

Ejercicio

¿Cuál o cuáles de las condiciones de Coffman no se cumple?

Recordemos las condiciones de Coffman

1. Exclusión mutua
2. Hold & Wait
3. Sin desalojo
4. Espera circular

Solución:
No hay exclusión mutua, porque a los efectos del problema R1 puede ser usado por todos los procesos al mismo tiempo.

Ejercicio

En un sistema hay tres procesos (P1, P2 y P3) y tres recursos (R1, R2, R3). Los tres recursos son de uso exclusivo. Se sabe que P1 requiere los tres recursos, P2 requiere de R1 y R2 y P3 sólo require R3.

1. ¿El sistema está libre de deadlock?
2. ¿P3 influye en que el sistema está o no libre de deadlock?
3. Si me aseguro que P2 no podrá pedir ningún recurso hasta que P1 haya liberado todos sus recursos ¿El sistema está libre de deadlock? ¿Por qué?

La próxima clase: Más ejercicios de sincronización difíciles