Sincronización entre procesos (semáforos)

Pablo De Cristóforis

DC - FCEyN - UBA

Sistemas Operativos, 2c-2014

Primero repasemos un poco lo que vieron en la teórica...

Primero repasemos un poco lo que vieron en la teórica...

¿Qué es la *monoprogramación*?

Primero repasemos un poco lo que vieron en la teórica...

¿Qué es la *monoprogramación*?

Sólo se ejecuta un programa (proceso) a la vez y no se ejecuta otro hasta que se termine el anterior.

Primero repasemos un poco lo que vieron en la teórica...

Primero repasemos un poco lo que vieron en la teórica...

¿Qué es una *race condition*?

Primero repasemos un poco lo que vieron en la teórica...

¿Qué es una *race condition*?

El resultado del la ejecución depende inesperadamente del orden en que se ejecuten ciertos procesos.

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

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

A

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

B

```
x = 4;
```

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

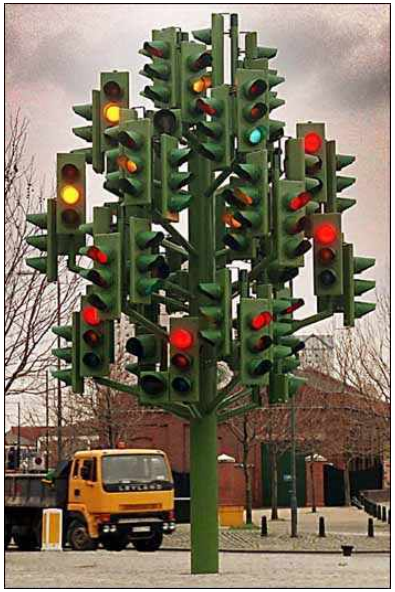
x comienza inicializado en 0.

A

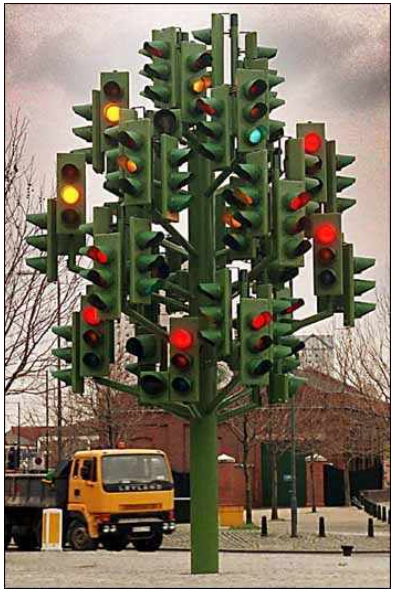
```
x = x + 1;
print(x);
```

B

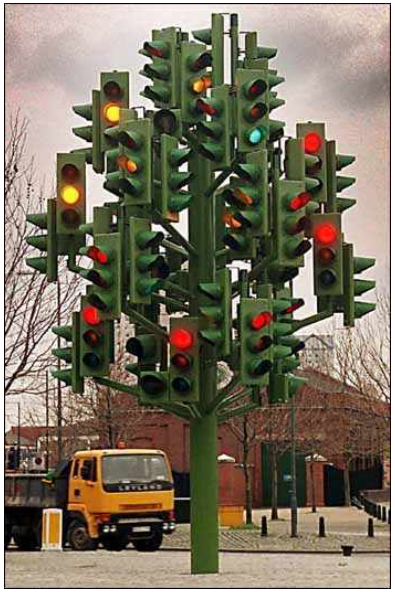
```
x = x + 1;
```



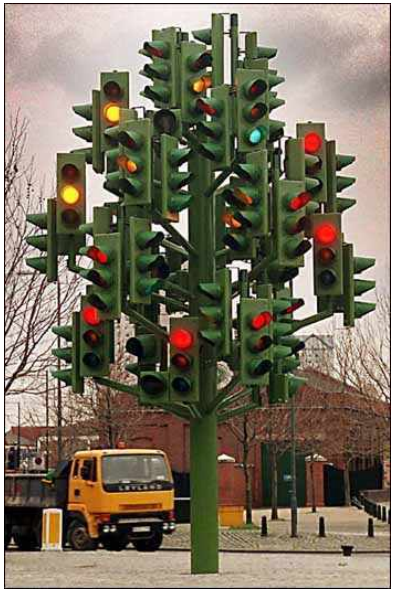
- ¿Qué es un semáforo?



- ¿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 ejecución concurrente.



- ¿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 ejecución concurrente.
- ¿Es lo mismo que usar un entero y fijarme qué valor tiene?



- ¿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 ejecución concurrente.
- ¿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:

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

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.)

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

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.)

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

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.

Un poco de Neerlandés:

- **Verhogen:** Incrementar $\rightarrow$ **Signal**
- **Proolag:** Tratar de reducir $\rightarrow$ **Wait**

Ejercicio

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

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

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

P(sem_A)	P(sem_B)	P(sem_C)
// Algo	// Algo	// Algo
V(sem_B)	V(sem_C)	V(sem_A)

Ejercicio

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

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 BCA,BCA,BCA,...?

Solución:

Cambio los valores de inicialización de los semáforos por sem_A = 0, sem_B = 1, sem_C = 0

P(sem_A)	P(sem_B)	P(sem_C)
// Algo	// Algo	// Algo
V(sem_B)	V(sem_C)	V(sem_A)

Ejercicio

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

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

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

P(sem_A)	P(sem_B)	P(sem_C)
// Algo	// Algo	P(sem_C)
V(sem_B)	V(sem_C)	// Algo
V(sem_B)		V(sem_A)

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 $P_i, P_{i+1}, ..., P_{N-1}, P_0, ..., P_{i-1}$ (donde i es otro parámetro).

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 $P_i, P_{i+1}, ..., P_{N-1}, P_0, ..., P_{i-1}$ (donde i es otro parámetro).

Solución:

Global

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

En cada P_i

```
P(semaforos[i])
// Algo
V(semaforos[(i + 1) % N])
```

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

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.

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)
Semaphore colaNoVacia =
Semaphore(0)
```

Q

```
while(true):
  colaNoVacia.wait()
  mutex.wait()
  c.top().signal()
  mutex.signal()
```

P_i

```
Semaphore yo = Semaphore(0)
mutex.wait()
n = long(c)
c.push(yo)
mutex.signal()
if (n == 0)
  colaNoVacia.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?

¿Cuándo está en *deadlock* un conjunto de procesos?

Deadlock o bloqueo mutuo

Es una situación en la que dos o más procesos que compiten por recursos están cada uno a la espera de que el otro termine, y por lo tanto, ninguna de los dos termina.

Condiciones de Coffman

- Exclusión mutua
- Retención y espera (*Hold & Wait*)
- No desalojo (*No Preemption*)
- Espera ciruclar

Ejemplos Visuales

Exclusión mutua: existencia de al menos de un recurso compartido por los procesos, al cual sólo puede acceder uno simultáneamente.

Ejemplos Visuales

Retención y espera (*Hold & Wait*): al menos un proceso ha adquirido un recurso, y lo retiene mientras espera por otro recurso que ya ha sido asignado a otro proceso.

Ejemplos Visuales

No desalojo (*No Preemption*): los recursos sólo podrán ser liberados voluntariamente por los procesos que los están usando, no hay desalojo forzoso.

Ejemplos Visuales

Espera circular: P_0 está esperando un recurso adquirido por P_1, que está esperando por P_2,... ,que está esperando por P_N, que está esperando un recurso adquirido por P_0

Deadlock

Un ejemplo típico de deadlock es:

`sem1 y sem2 comienzan inicializados en 1.`

P_1

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

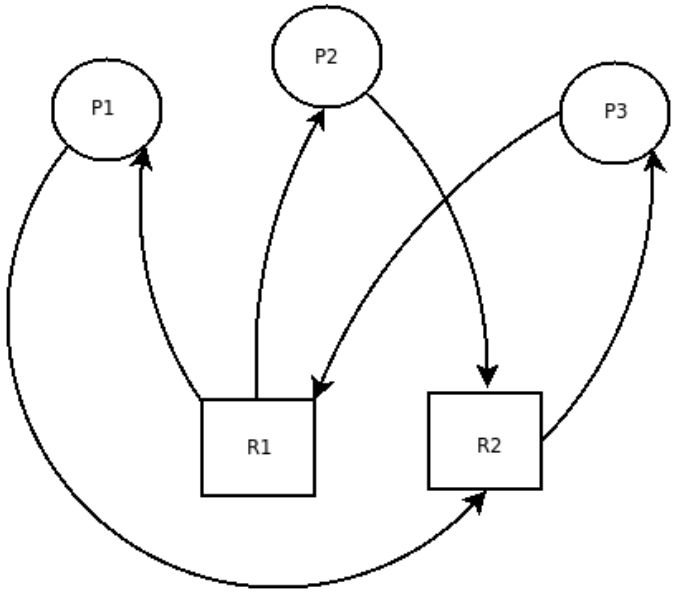
P_2

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

Representación de deadlocks usando grafos

Grafos dirigidos donde los procesos se representan con nodos circulares y los recursos con nodos rectangulares.

P1 $\rightarrow$ R2 representa que P1 necesita el recurso R2.
R1 $\rightarrow$ P1 representa que R1 está asignado a P1.

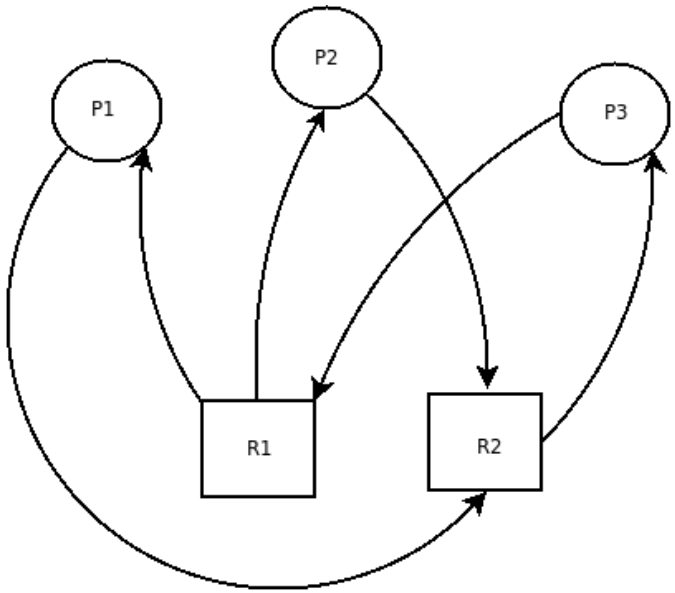


Ejercicio

En un sistema conviven 3 procesos y 2 recursos. Uno de los recursos (R2) es de uso exclusivo y el otro (R1) puede ser compartido por hasta dos procesos. ¿Puede haber deadlock?

Ejercicio

En un sistema conviven 3 procesos y 2 recursos. Uno de los recursos (R2) es de uso exclusivo y el otro (R1) puede ser compartido por hasta dos procesos. ¿Puede haber deadlock?

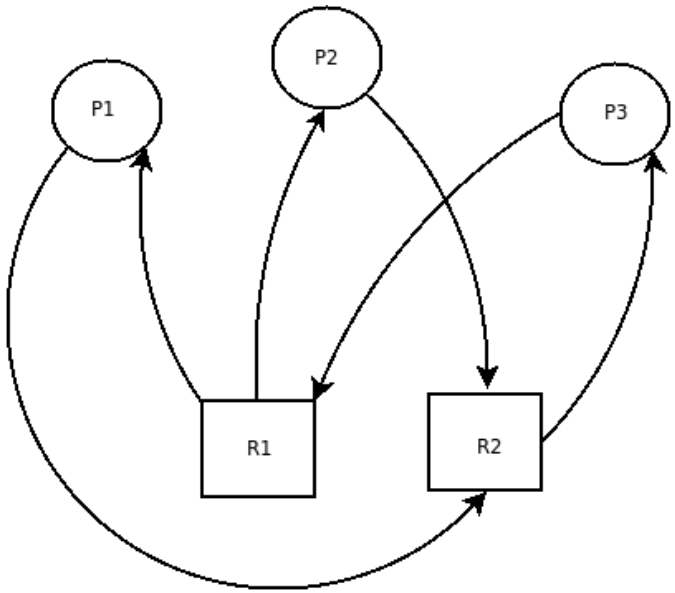


Ejercicio

En un sistema conviven 3 procesos y 2 recursos. Uno de los recursos (R2) es de uso exclusivo y el otro (R1) puede ser compartido por hasta dos procesos. ¿Puede haber deadlock?

Solución:

Sí.



Ejercicio

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

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?

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

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. No se olviden de comenzar a hacer la Práctica 3. Lean el libro "The Little Book of Semaphores".

La próxima clase: Más ejercicios de sincronización difíciles. No se olviden de comenzar a hacer la Práctica 3. Lean el libro "The Little Book of Semaphores".