Sincronización entre Procesos II

Leopoldo Taravilse[1]

[1]Departamento de Computación
Sistemas Operativos

11/09/2014

Mutual Exclusion (Mutex o Exclusión Mutua)

Lograr el acceso exclusivo a una variable o porción de código.

Mutual Exclusion (Mutex o Exclusión Mutua)

```
1 int critica = 0;
2
3 A():
4     critica++;
5     printf("%d\n",critica);
6
7 B():
8     critica--;
9     printf("%d\n",critica);
```

Mutual Exclusion (Mutex o Exclusión Mutua)

```
1  int critica = 0;
2  sem mutex = Semaphore(1);
3
4  A():
5      mutex.wait();
6      critica++;
7      printf("%d\n",critica);
8      mutex.signal();
9
10 B():
11     mutex.wait();
12     critica--;
13     printf("%d\n",critica);
14     mutex.signal();
```

Rendezvous (Encuentro)

Rendezvous (Encuentro)

Dados dos procesos A y B, logra que ambos se encuentren en determinado punto del código antes de seguir ejecutando.

Rendezvous (Encuentro)

```
1  int critica = 0;
2
3  A():
4      critica++;
5      printf("%d\n",critica);
6
7  B():
8      critica--;
9      printf("%d\n",critica);
```

Cómo hacemos para que los dos procesos impriman 0?

Rendezvous (Encuentro)

```
1  int critica = 0;
2
3  A():
4      critica++;
5      printf("%d\n",critica);
6
7  B():
8      critica--;
9      printf("%d\n",critica);
```

Cómo hacemos para que los dos procesos impriman 0?

Pensemos el código como algo más general, tenemos dos segmentos de código en cada proceso, y queremos que el primer segmento de cada proceso se ejecute antes que el segundo segmento de código de cada proceso.

Rendezvous (Encuentro)

```
1 A():
2     f1();
3     f2();
4
5 B():
6     g1();
7     g2();
```

Rendezvous (Encuentro)

```
1 A():
2     f1();
3     f2();
4
5 B():
6     g1();
7     g2();
```

Hint: tenemos dos semáforos *a* y *b* inicializados en cero que nos dicen cuándo ya se ejecutó *f1* y *f2*

Rendezvous (Encuentro)

```
1  A():
2      f1();
3      a.signal();
4      b.wait();
5      f2();
6
7  B():
8      g1();
9      b.signal();
10     a.wait();
11     g2();
```

Barrera

Es la idea de *Rendezvous* extendida a **N** procesos.

Barrera

```
1 A(i):
2     f(i);
3     g(i);
```

Barrera

```
1 A(i):
2     f(i);
3     g(i);
```

Cómo hacemos para que cada proceso ejecute su *g(i)* una vez que todos los procesos ejecutaron su *f(i)*?

Barrera

```
1  int n = N;
2  sem mutex = Semaphore(1);
3  sem b = Semaphore(0);
4  A(i):
5      f(i);
6      mutex.wait(); //Acceso exclusivo
7      n--;
8      yo = n;
9      mutex.signal();
10     if (yo == 0) // Soy el ultimo
11         b.signal();
12     b.wait();
13     b.signal(); // <- wait+signal = turnstile
14     g(i);
```

Barrera

```
1  int n = N;
2  sem mutex = Semaphore(1);
3  sem b = Semaphore(0);
4  A(i):
5      f(i);
6      mutex.wait(); //Acceso exclusivo
7      n--;
8      yo = n;
9      mutex.signal();
10     if (yo == 0) // Soy el ultimo
11         b.signal(N);
12     b.wait();
13     g(i);
```

Cola ordenada

Los procesos deben acceder a la sección critica en orden FIFO.

Cola ordenada

```
1 P(i):
2     f(i); //<-- Seccion critica
```

```
1 Q():
2     while(true)
3         g();
4         // <-- Aca corren los P en orden.
```

Cola ordenada

```
1 P(i):
2     f(i); //<-- Seccion critica
```

```
1 Q():
2     while(true)
3         g();
4         // <-- Aca corren los P en orden.
```

cómo hacemos para que los procesos P ejecuten *f(i)* en el orden en que llegaron, luego de que Q ejecute *g()*?

Cola ordenada

```
1  cola c = CrearColaVacia();
2  sem mutex = Semaphore(1);
3  sem colaNoVacia = Semaphore(0);
4
5  Q():
6      while(true)
7          g();
8          colaNoVacia.wait();
9          m.wait();
10         c.top().signal();
11         m.signal();
```

```
1  P(i):
2      sem yo = Semaphore(0);
3      m.wait();
4      n = c.size();
5      c.push(yo);
6      m.signal();
7      if (n == 0)
8          colaNoVacia.signal();
9      yo.wait();
10     f();
11     m.wait();
12     c.pop();
13     if (!c.empty())
14         c.top().signal();
15     m.signal();
```

Cola ordenada

```
1  cola c = CrearColaVacia();
2  sem mutex = Semaphore(1);
3  sem colaNoVacia = Semaphore(0);
4
5  Q():
6      while(true)
7          g();
8          colaNoVacia.wait();
9          m.wait();
10         c.top().signal();
11         m.signal();
```

```
1  P(i):
2      sem yo = Semaphore(0);
3      m.wait();
4      n = c.size();
5      c.push(yo);
6      m.signal();
7      if (n == 0)
8          colaNoVacia.signal();
9      yo.wait();
10     f();
11     m.wait();
12     c.pop();
13     if (!c.empty())
14         c.top().signal();
15     m.signal();
```

Otras estructuras

La misma idea se puede usar para simular cualquier estructura que tenga un orden definido de cómo sacar elementos. Por ejemplo, una pila, o una cola de prioridad que puede estar implementada con un heap.