problema clásico

El problema del productor y consumidor es un ejemplo clásico de programa concurrente y consiste en la situación siguiente: de una parte se produce algún producto (datos en nuestro caso) que se coloca en algún lugar (una cola en nuestro caso) para que sea consumido por otra parte. Como algoritmo obtenemos:

```
producer:
  forever
    produce(item)
    place(item)
```

```
consumer:
  forever
    take(item)
    consume(item)
```

requerimientos

Queremos garantizar que el consumidor no coja los datos más rápido de lo que los está produciendo el productor. Más concreta:

1. el productor puede generar sus datos en cualquier momento, pero no debe producir nada si no lo puede colocar
2. el consumidor puede coger un dato solamente cuando hay alguno
3. para el intercambio de datos se usa una estructura de datos compartida a la cual ambos tienen acceso,
4. si se usa una cola se garantiza un orden temporal
5. ningún dato no está consumido una vez haber sido producido (por lo menos se descarta...)

cola infinita

Si la cola puede crecer a una longitud infinita (siendo el caso cuando el consumidor consume más lento de lo que el productor produce), basta con la siguiente solución que garantiza exclusión mutua a la cola:

```
producer:
  forever
    produce(item)
    place(item)
    itemsReady.signal()
```

```
consumer:
  forever
    itemsReady.wait()
    take(item)
    consume(item)
```

donde `itemsReady` es un semáforo general que se ha inicializado al principio con el valor `0`.

corrección

El algoritmo es correcto, lo que se vee con la siguiente prueba. Asumimos que el consumidor adelanta el productor. Entonces el número de `wait()`s (terminados) tiene que ser más grande que el número de `signal()`s:

```
        #waits > #signals
==>     #signals - #waits < 0
==>     itemsReady < 0
```

y la última línea es una contradicción a la invariante del semáforo.

más participantes

Queremos ampliar el problema introduciendo más productores y más consumidores que trabajen todos con la misma cola. Para asegurar que todos los datos estén consumidos lo más rápido posible por algún consumidor disponible tenemos que proteger el acceso a la cola con un semáforo binario (llamado `mutex` abajo):

```
producer:
  forever
    produce(item)
    mutex.wait()
    place(item)
    mutex.signal()
    itemsReady.signal()
```

```
consumer:
  forever
    itemsReady.wait()
    mutex.wait()
    take(item)
    mutex.signal()
    consume(item)
```

cola finita

- Normalmente no se puede permitir que la cola crezca infinitamente, es decir, hay que evitar producción en exceso también.
- Como posible solución introducimos otro semáforo general (llamado `spacesLeft`) que cuenta cuantos espacios quedan libres en la cola.
- Se inicializa el semáforo con la longitud máxima permitida de la cola.
- Un productor queda bloqueado si ya no hay espacio en la cola y un consumidor señala su consumisión.

cola finita

```
producer:
  forever
    spacesLeft.wait()
    produce(item)
    mutex.wait()
    place(item)
    mutex.signal()
    itemsReady.signal()
```

```
consumer:
  forever
    itemsReady.wait()
    mutex.wait()
    take(item)
    mutex.signal()
    consume(item)
    spacesLeft.signal()
```

otras estructuras compartidas

- En un sistema con multiples productores y/o consumidores, puede ser difícil establecer un orden temporal con una semántica adecuada.
- Se puede aflojar la condición de usar una cola, y usar estructuras de datos que permitan más concurrencia.
- Un ejemplo simple serían vectores de contenedores inspeccionados en orden cíclico por los productores y consumidores.