Java – Concurrencia (2)

Problema del barbero durmiente

En ciencias de la computación, el problema del barbero durmiente es un problema de sincronización. El problema consiste en una barbería en la que trabaja un barbero que tiene un único sillón de barbero y varias sillas para esperar. Cuando no hay clientes, el barbero se sienta en una silla y se duerme. Cuando llega un nuevo cliente, éste o bien despierta al barbero o —si el barbero está afeitando a otro cliente— se sienta en una silla (o se va si todas las sillas están ocupadas por clientes esperando). El problema consiste en realizar la actividad del barbero sin que ocurran condiciones de carrera. La solución implica el uso de semáforos y objetos de exclusión mutua para proteger la sección crítica.

Proporcionamos el algoritmo que garantiza la sincronización entre el barbero y el cliente, pero puede llevar a inanición del cliente. `wait()` y `signal()` son funciones provistas por el semáforo equivalentes a `acquire()` y `reléase()` en la clase `Semaphore`.

- Se necesita:

```
Semáforo barberoListo = 0      // (Mutex, sólo 1 o 0)
Semáforo sillasAccesibles = 1 // (Mutex) Cuando sea 1, el número de sillas
                               // libres puede aumentar o disminuir
```

```
Semáforo clientes = 0          // Número de clientes en la sala de espera
int sillasLibres = N           // N es el número total de sillas
```

- Función barbero (Proceso/hilo-thread):

```
while(true) {                    // Ciclo infinito
  wait(clientes)    // Espera la señal de un hilo cliente para despertar.
  wait(sillasAccesibles)      // (Ya está despierto) Espera señal para
                              // poder modificar sillasLibres.
  sillasLibres += 1  // Aumenta en uno el número de sillas libres.
  signal(barberoListo)  // El barbero está listo para cortar y manda
                        // señal al hilo cliente.
  signal(sillasAccesibles)  // Manda señal para desbloquear el acceso a
                            // sillasLibres
  // Aquí el barbero corta el pelo de un cliente
  // (zona de código no crítico).
}
```

- Función cliente (Proceso/hilo-thread):

```
wait(sillasAccesibles)    // Espera la señal para poder acceder a
                          // sillasLibres.
if (sillasLibres > 0) {  // Si hay alguna silla libre, se sienta en una.
  sillasLibres -= 1  // Decrementando el valor de sillasLibres en 1.
  signal(clientes)    // Manda señal al barbero de que hay un cliente
                      // disponible.
  signal(sillasAccesibles)  // Manda señal para desbloquear el acceso a
                            // sillasLibres.
  wait(barberoListo)        // El cliente espera a que el barbero esté
                            // listo para atenderlo.
```

```
    // Se le corta el pelo al cliente.
}
else {                          // Si no hay sillas libres.
  signal(sillasAccesibles)      // Manda señal para desbloquear el acceso a
                                // sillasLibres.
  // El cliente se va de la barbería y
  // no manda la señal de cliente disponible.
}
```

Ejercicio

De acuerdo con el diseño algorítmico que se os ha proporcionado. Implementad en Java con la clase Semaphore el algoritmo.

Ejercicio

Proponed una solución al problema de inanición de los clientes. ¿Se puede hacer en Java?