Programación Concurrente

Práctica especial: Simulacro del 1er parcial

Ejercicio 1. [**Exclusión mutua**] Considere la siguiente propuesta para extender el algoritmo de Peterson a 3 threads:

```
global flags = {False, False, False};
global turno = 0;

thread (id): {
        izq = id-1%3;
        der = id+1%3;
        //SNC
        flags[id]=true;
        turno = der;
        while((flags[izq] && turno==izq) || (flags[der] && turno==der);
        //SC
(c) →   flags[id] = false;
        //SNC
}
```

a) Muestre que esta propuesta **no** cumple la condición de Mutex.

b) Muestre que esta propuesta **sí** tiene Garantía de Entrada (Idea: Considere el caso de un deadlock entre los 3 threads y el caso con sólo 2)

c) ¿Se cumple la condición de Mutex si se cambia la línea `turno=der` de su posición actual a inmediatamente después de la Sección Crítica?

Ejercicio 2. [**Semáforos**] Se desea diseñar un sistema de control para un local de lavado de autos automatizado. El proceso de lavado se lleva a cabo por maquinaria especializada localizada en las siguientes 5 estaciones: remojado, enjabonado, enjuague, secado, encerado.

Un vehículo avanza a través de las estaciones en secuencia. En cada estación sólo puede haber un vehículo en un momento dado, por lo que el sistema debe evitar que un vehículo avance hasta que la estación siguiente esté libre. No obstante, el sistema debe permitir el lavado simultáneo de múltiples vehículos siempre y cuando no haya riesgo de colisiones.

Resuelva los siguientes puntos teniendo en cuenta que (i) el proceso realizado por cada máquina no es una acción atómica, y por lo tanto, requiere de cierto tiempo durante el cuál el vehículo debe permanecer en la estación correspondiente; y (ii) el desplazarse de una estación a la siguiente también requiere de cierto tiempo durante el cuál se considera que el vehículo ocupa las dos estaciones.

a) De un programa utilizando semáforos que modele el escenario descrito. Utilice threads para modelar las máquinas lavadoras y los vehículos.

b) Extienda la solución anterior contemplando adicionalmente una nueva estación inicial y final. En la nueva estación inicial una aspiradora robótica (modelada con un thread aparte) asciende al vehículo e inicia un proceso de limpieza interno. La limpieza toma algún tiempo,

pero no previene que el auto avance por las estaciones. El robot sólo puede descender del vehículo en la última estación. Tenga en cuenta que el ascenso y descenso del robot toma algún tiempo y que el vehículo no debe desplazarse hasta que esos movimientos terminen. Asuma que hay 6 robots en funcionamiento.

c) Distinga los threads que puedan sufrir inanición en su solución al asumir que todos los semáforos utilizados son débiles (o *unfair*).

Ejercicio 3. [**Exclusión mutua**] Considere la siguiente propuesta inspirada en semáforos para la resolución del problema de exclusión mutua. Tome en cuenta que al momento de trabajar con memoria compartida, sólo se consideran atómicas la lectura y escritura de variables.

```
global int permisos = 1;

thread {
  //SNC
  bool tengoPermiso = False;
  while (!tengoPermiso)
    tengoPermiso = pedirPermiso();
  //SC
  devolverPermiso();
}
```

donde

```
public bool pedirPermiso {
  if (permisos > 0) {
    permisos--;
    return true;
  } else {
    return false;
  }
}

public void devolverPermiso() {
  permisos++;
}
```

a) Muestre que esta propuesta no cumple la propiedad de Mutex y justifique apropiadamente.

b) Muestre que esta propuesta tampoco cumple la propiedad de Garantía de Entrada (pista: Considere un escenario con 3 (o más) threads)

c) Suponga ahora que las funciones `pedirPermiso()` y `devolverPermiso()` son atómicas. Muestre que ambas propiedades ahora sí se cumplen.

Ejercicio 4. [**Semáforos**] En el juego de la ruleta múltiples apostadores pueden empezar sus apuestas hasta que el crupier dice "no va más" (i.e., un print). Luego de esto, y una vez que se terminan de hacer las apuestas que ya estaban en proceso, el crupier gira la ruleta y la bolilla (suponga una funcion `girarRuleta()` que retorna un valor entre 0 y 36) y canta el valor. A partir del resultado se premia a los apostadores que hayan acertado algún número, y luego comienza otra ronda de apuestas.

Resuelva los siguientes puntos usando semáforos y considerando: (i) que el crupier espera algún tiempo antes de dar por terminada una ronda de apuestas; ($\ddot{u}$) que la bolilla gira por algún tiempo antes de indicar el resultado, ($\ddot{\ddot{m}}$) que el pago de los premios también requiere de algún tiempo (i.e., no pueden considerarse acciones instantáneas), y ($\dot{w}$) sólo se debe esperar que los pagos se realicen si se hizo alguna apuesta en esa ronda.

a) Modele el escenario descrito asumiendo hay un sólo (thread) apostador, con un capital inicial dado y que en cada ronda realiza una única apuesta con una ficha. Por simplicidad sólo consideramos apuestas sobre los números entre 0 y 36 (i.e., sin color ni otras categorías especiales). Un pleno (acierto de un número) paga un monto equivalente a 36 veces la apuesta efectuada. Una vez cantado el número ganador, el apostador debe calcular su nuevo capital y gritar "Gané!" o "Perdí!" dependiendo del resultado de su apuesta. Si se queda sin capital exclama "No, mi casa!" y se retira (terminando la ejecución del thread que lo modela). Para elegir el número, puede suponer una función `elegirNumero()` en el apostador.

b) Extienda su solución anterior considerando que puede haber una cantidad arbitraria de apostadores accediendo simultáneamente a la mesa de apuestas, pero en exclusión mutua con el crupier. En este punto asigne la prioridad en el acceso a la mesa a los apostadores. Recuerde esperar que *todas* las personas que hayan apostado hayan gritado su resultado antes de iniciar la siguiente ronda.

c) Modifique su solución anterior asignando esta vez la prioridad al crupier.