Programación Concurrente

Práctica 2: Exclusión Mutua

Ejercicio 1.

a) ¿Cómo se puede probar que un programa con múltiples threads no cumple con la condición de Mutex?

b) ¿Cómo se puede probar que un programa con múltiples threads no cumple con la condición de Ausencia de Live/DeadLocks?

c) ¿Cómo se puede probar que un programa con múltiples threads no cumple con la condición de Garantía de Entrada?

d) ¿Es válido probar que **sí** se cumple alguna de estas condiciones mostrando una traza?

Ejercicio 2. Considerar la siguiente propuesta para garantizar exclusión Mutua con 2 threads:

```
global boolean[] flags = {false, false};

thread {                          thread {
  id = 0;                           id = 1;
  // seccion no critica             // seccion no critica
  otro = (id + 1) % 2;              otro = (id + 1) % 2;
  while (flags[otro]);              while (flags[otro]);
  flags[id] = true;                 flags[id] = true;
  // seccion critica                // seccion critica
  flags[id] = false;                flags[id] = false;
  // seccion no critica             // seccion no critica
}                                 }
```

a) Explique por qué, en este caso, sí hay Asusencia de Deadlocks y Garantía de Entrada.

b) Muestre que no se cumple con la propiedad de Exclusión Mutua.

Ejercicio 3. Considerar la propuesta para resolver el problema de exclusión mutua para n procesos que utiliza las siguientes funciones y variables compartidas:

```
global int actual = 0, turnos = 0;

PedirTurno() {                    LiberarTurno() {
  turno  = turnos;                  actual = actual + 1;
  turnos = turnos + 1;              turnos = turnos - 1;
  return turno;                   }
}
```

Considere, también, que cada thread ejecuta el siguiente protocolo:

```
// seccion no critica
turno = PedirTurno();
while (actual != turno);
// seccion critica
LiberarTurno();
// seccion no critica
```

Mostrar que esta propuesta no resuelve el problema de la exclusión mutua. Explicar cuáles condiciones se cumplen y cuáles no, justificando o mostrando una traza según corresponda.

Ejercicio 4. Considerar una extensión del algoritmo de Peterson para n procesos (con $n > 2$) que utiliza las siguientes variables compartidas:

```
global boolean[] flags = replicate(n,false);
global int turno = 0;
```

Además, cada thread está identificado por el valor de la variable local `threadId` (que toma valores entre 0 y $n-1$) . Cada thread utiliza el siguiente protocolo.

```
...
// seccion no critica
flags[threadId] = true;
otro = (threadId+1) % n;
turno = otro;
while (flags[otro] && turno==otro);
// seccion critica
flags[threadId] = false;
// seccion no critica
...
```

Mostrar que esta variación no resuelve el problema de la exclusión mutua para n procesos, con $n > 2$. Explicar cuáles condiciones se cumplen y cuáles no, justificando o mostrando una traza según corresponda.

Ejercicio 5. Dado el algoritmo de *bakery*, mostrar que la condición `j < id` en el segundo while es necesaria. Es decir, muestre que el algoritmo que se obtiene al eliminar esta condición no resuelve el problema de la exclusión mutua. Mencionar al menos una propiedad que viola el algoritmo obtenido y mostrar una traza de ejecución que lo evidencie. Por comodidad, damos a continuación el algoritmo modificado (donde se eliminó la condición `j < id`)

```
global boolean[] entrando = replicate(N,false);
global int[] numero = replicate(N,0);

thread {
  // seccion no critica
  entrando[threadId] = true;
  numero[threadId] = 1 + maximum(numero);
  entrando[threadId] = false;
  for (j : range(0,n-1)) {
    while (entrando[j]);
    while (numero[j] != 0 &&
          (numero[j] < numero[threadId] ||
          (numero[j] == numero[threadId])));
```

```
    }
    // seccion critica
    numero[threadId] = 0;
    // seccion no critica
}
```