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 Garantía de Entrada?

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

Ejercicio 2. Considere 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 Garantía de Entrada.

b) Muestre que **no** cumple Mutex.

Ejercicio 3. Considere la siguiente propuesta para resolver el problema de exclusión mutua para 3 threads que utiliza las siguientes funciones y variables compartidas:

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

PedirTurno() {                    PasarTurno() {
  int turno = turnos;               actual = actual + 1;
  turnos = turnos + 1;            }
  return turno;
}
```

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

```
// seccion no critica
miTurno = PedirTurno();
while (actual != miTurno);
// seccion critica
PasarTurno();
// seccion no critica
```

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

Ejercicio 4. Considere la siguiente propuesta para resolver el problema de exclusión mutua para 2 procesos que utiliza las variables compartidas turno y flags:

```
global int turno = 0;
global boolean[] flags = {false, false};

thread {
  id = 0;  // o 1 resp.
  // seccion no critica
  otro = (id + 1) % 2;
  flags[id] = true;
  while (flags[otro])
    if (turno == otro) {
      flags[id] = false;
      while (turno != id);
      flags[id] = true;
    }

  // SECCION CRITICA

  turno = otro;
  flags[id] = false;
  // seccion no critica
}
```

Indique si esta propuesta resuelve el problema de exclusión mutua, explicitando cuáles condiciones se cumplen y cuáles no (justificando o mostrando una traza según corresponda).

Ejercicio 5. Considere 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 `id` (que toma valores entre 0 y $n - 1$). Cada thread utiliza el siguiente protocolo.

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

Indique si esta propuesta resuelve el problema de exclusión mutua para n procesos, explicitando cuáles condiciones se cumplen y cuáles no (justificando o mostrando una traza según corresponda).

Ejercicio 6. Dado el algoritmo de *Bakery*, muestre 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. Mencione al menos una propiedad que esta variante del algoritmo no cumple y muestre 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
}
```