Programación Concurrente

Práctica 2: Exclusión Mutua

1. Considerar la siguiente 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() {
  turno  = turnos;
  turnos = turnos + 1;
  return turno;
}

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

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

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

   Mostrar que esta propuesta no resuelve el problema de la exclusión mutua. Indicar claramente cuál/es condición/es son violadas e ilustrar mostrando una traza.

2. Considerar la siguiente 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 $n > 2$. Indique claramente cuál/es condición/es son violadas e ilustre mostrando una traza.

3. Considerar la siguiente propuesta para resolver el problema de la exclusión mutua para n procesos (con $n > 2$) que utiliza las siguientes variables compartidas:

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

   y la siguiente función auxiliar

```
boolean AlgunOtroVerdadero(id) {
  result = false;
  for (i : range(0,n-1))
    if (i != id)
      result = result || flags[i];
  return result;
}
```

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;
while (AlgunOtroVerdadero(threadId));
// seccion critica
flags[threadid] = false;
// seccion no critica
...
```

Mostrar que esta propuesta no resuelve el problema de la exclusión mutua para n procesos, con $n > 2$ si la operación `AlgunOtroVerdadero` no es atómica. Indicar claramente cuál/es condición/es son violadas e ilustrar mostrando una traza.

4. 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
}
```