Programación Concurrente

Práctica 1: Modelo de cómputo y Exclusión Mutua

1. Dado el siguiente programa

```
global x = 0
global y = 0

thread          thread
   y = x + 1       x = y + 1
```

   a) Mostrar una ejecución en la que los valores de las variables globales al final de la ejecución del programa sean $x = 2$ e $y = 1$.

   b) Dar el diagrama de transición de estados.

2. Dado el siguiente programa

```
global n = 0

thread                  thread
   local temp;             local temp;
   do 5 times              do 5 times
      temp = n                temp = n
      n = temp + 1            n = temp + 1
```

   - Dar el diagrama de transición de estados.

3. Asumir que la función f tiene una raíz entera, es decir, $f(x) = 0$ para algún valor x entero. A continuación proponemos distintos programas para encontrar tal raíz. Consideraremos a un programa correcto si ambos threads terminan cuando uno de ellos ha encontrado la raíz. Para cada programa, decir si es correcto o no, justificando la respuesta.

   Programa A

```
global found

thread                          thread
   local i = 0;                    local j = 1;
   found = false;                  found = false;
   while (not found)               while (not found)
      i = i + 1;                      j = j - 1;
      found = (f(i) == 0)             found = (f(j) == 0)
```

   Programa B

```
global found = false

thread                          thread
   local i = 0;                    local j = 1;
   while (not found)               while (not found)
      i = i + 1;                      j = j - 1;
      found = (f(i) == 0)             found = (f(j) == 0)
```

   Programa C

```
global found = false

thread                      thread
   local i = 0;                local j = 1;
   while (not found)           while (not found)
      i = i + 1;                  j = j - 1;
      if (f(i) == 0)              if (f(j) == 0)
         found = true                found = true
```

4. Considerar el siguiente programa

```
global n = 0

thread              thread
   while (n < 2)       n = n + 1;
      print (n)        n = n + 1;
```

a) ¿Aparece el valor 2 necesariamente en la salida?

b) ¿Cuántas veces puede aparecer 2 en la salida?

c) ¿Cuántas veces puede aparecer 1 en la salida?

d) ¿Cuál es la longitud de la secuencia mas corta que puede ser mostrada?

5. Considerar el siguiente programa

```
global n = 0

thread                  thread
   while (n < 1)           while (n >= 0)
      n = n + 1               n = n - 1
```

a) Existe una ejecución en la que el loop en el thread de la izquierda ejecute exactamente una vez? Justificar

b) Existe una ejecución en la que el loop en el thread de la izquierda no termina?.

6. Considerar el siguiente programa

```
global n = 0
global flag = false

thread                      thread
   while (not flag)            while (not flag)
      n = 1 - n                   if n = 0
                                     flag = true
```

a) Cuáles son los posibles valores de n cuando el programa termina. Justificar.

b) Puede una ejecución del programa no terminar?

7. Considerar la siguiente versión del algoritmo de bakery para dos procesos.

```
                shared np = 0; nq= 0;

thread p                                thread q
  while(true){                            while(true){
     np = nq+1                               nq = np+1
     while (nq != 0 && np > nq){};           while (np != 0 && nq > np){};
     // seccion critica                      // seccion critica
     np= 0                                   np= 0
  }                                       }
```

Resuelve este algoritmo el problema de la exclusión mutua? Justificar.

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

```
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
```

a) Mostrar que esta propuesta no resuelve el problema de la exclusión mutua. Indicar claramente cuales condiciónes son violadas e ilustrarlas dando una traza.

b) ¿Qué sucede si PedirTurno y LiberarTurno son operaciones atómicas?

9. 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`)

```
shared entrando[N] = {false , ..., false}
shared numero[N] = {0, ..., 0}


Thread(id)
// seccion no critica
entrando[id] = true
numero[id] = 1 + max(numero[1], ..., [n])
entrando[id] = false

for (j = 1; j <= n; ++j)
   while (entrando[j]);
   while (numero[j] != 0  && (numero[j] < numero[id] ||
                              (numero[j] == numero[id])));
// seccion critica
numero[id] = 0
// seccion no critica
```

10. Considere la operación atómica fetch-and-add definida de la siguiente manera:

```
fetch-and-add(compartida, local, x)
   local = compartida
   compartida = compartida + x
```

   y el siguiente algoritmo para resolver el problema de la exclusión mutua que utiliza las variables compartidas ticket y turno, ambas inicializadas en 0.

```
int miturno
// Seccion no critica
fetch-and-add(ticket, miturno, 1)
while (turno != miturno) {}
//Seccion Critica
fetch-and-add(ticket, miturno, -1)
//Seccion no Critica
```

   Indique si el algoritmo resuelve el problema de la exclusión mutua o no. Justifique (en caso negativo indicar que condición es violada y mostrar una traza).