GII-ADE-M. Relación de Problemas. Tema 2-2. 25/10/24

62. Un algoritmo para el cual sólo pudiésemos demostrar que cumple las 4 condiciones de Dijkstra, ¿qué tipo de propiedades concurrentes satisfacería: (a) seguridad, (b) vivacidad, (c) equidad? Justificar las respuestas.

63. En algunas aplicaciones es necesario tener exclusión mutua entre procesos con la particularidad de que puede haber como mucho `n` procesos en una sección crítica, con `n` arbitrario y fijo, pero no necesariamente igual a la unidad, sino posiblemente mayor. Diseña una solución para este problema basada en el uso de espera ocupada y cerrojos. Estructura dicha solución como un par de subrutinas (usando una misma estructura de datos en memoria compartida), una para el protocolo de entrada y otro el de salida, e incluye el pseudocódigo de las mismas.

64. ¿Podría pensarse que una posible solución al problema de la exclusión mutua, sería el siguiente algoritmo que no necesita compartir una variable `turno` entre los 2 procesos? Demostrar (sí o no) se satisfacen las siguientes propiedades:

    (a) ¿la exclusión mutua? (propiedad de seguridad)

    (b) ¿la ausencia de interbloqueo? (propiedad de alcanzabilidad)

```
//variables compartidas y valores iniciales
var b0 : boolean := false; //true si P0 quiere acceder o esta en SC
    b1 : boolean := false ; //true si P1 quiere acceder o esta en SC
```

```
Process P0 ;
  begin
        while true do begin
        //protocolo de entrada:
        b0 := true ; //indica quiere
            entrar
          while b1 do begin //si el
              otro tambien:
                b0 := false ;//cede
                    temporalmente }
                while b1 do begin end
                      //espera
                 b0 := true ; //
                    vuelve a cerrar
                    paso
                end
          //seccion critica ....
          //protocolo de salida
                b0 := false ;
          //resto sentencias ....
          end
        end
```

```
Process P1 ;
   begin
        while true do begin 3
        //protocolo de entrada:
        b1 := true ; //indica quiere
            entrar
        while b0 do begin //si el
            otro tambien:
          b1 := false ; //cede
              temporalmente
          while b0 do begin end //
              espera
                 b1 := true ;//vuelve
                     a cerrar paso
          end
        //seccion critica ....
        //protocolo de salida
        b1 := false ;
        //resto sentencias ....
        end
  end
```

65. Al siguiente algoritmo se le conoce como solución de Hyman al problema de la exclusión mutua (fue publicado en una revista de impacto en 1966[1]). ¿Es correcta dicha solución?

[1] Harris Hyman, "Comments on a problem in concurrent programming control", Communications of the ACM, v.9 n.1, p.45, 1966

```
//variables compartidas y valores iniciales
var c0 : integer := 1 ; c1 : integer := 1 ; turno : integer := 1 ;
```

```
process P0 ;
  begin
        while true do begin
         c0 := 0 ;
         while turno != 0 do begin
          while c1 = 0 do begin end
         turno := 0 ;
         end
         //seccion critica
         c0 := 1 ;
         //resto sentencias }
  end
end
```

```
process P1 ;
  begin
        while true do begin
        c1 := 0 ;
        while turno != 1 do begin
         while c0 = 0 do begin end
        turno := 1 ;
        end
        //seccion critica
        c1 := 1 ;
        //resto sentencias
 end
end
```

66. Supongamos el algoritmo de exclusión mutua que expresamos a continuación. Tenemos los procesos: 0, 1, ... n − 1. Cada proceso i tiene una variable $s[i]$, inicializada a 0, que puede tomar los valores 0/1. El proceso i puede entrar en la sección crítica si:

$$s[i] \; != \; s[i-1] \;\; para\, i > 0;$$
$$s[0] \;\; = \;\; s[n-1] \;\; para\, i = 0;$$

Tras ejecutar su sección crítica, el proceso i deberá hacer:

$$s[i] \;\; = \;\; s[i-1] \;\; para\, i > 0;$$
$$s[0] \;\; = \;\; (s[0]+1)\, mod\, 2 \;\; para\, i == 0;$$

67. Se tienen 2 procesos concurrentes que representan 2 máquinas expendedoras de tickets (señalan el turno en que ha de ser atendido el cliente), los números de los tickets se representan por dos variables n1 y n2 que valen inicialmente 0. El proceso con el número de ticket más bajo entra en su sección crítica. En caso de tener 2 números iguales se procesa primero el proceso número 1.

    (a) Demostrar que se verifica la ausencia de interbloqueo (propiedad de *alcanzabilidad* de la sección crítica), la ausencia de inanición (propiedad de *vivacidad*) y la exclusión mutua (una propiedad de *seguridad*).

    (b) Demostrar que las asignaciones `n1:=1` y `n2:=1` son ambas necesarias.

```
//variables compartidas y valores iniciales
var n1 : integer := 0 ; n2 : integer := 0 ;
```

```
process P1 ;
  begin
        while true do begin
        n1 := 1 ; // E1.1
        n1 := n2+1 ; // L1.1 ; E2.1
        while n2 != 0 and // L2.1
          n2 < n1 do begin end;//L3.1
        // seccion critica } { SC.1 }
          n1 := 0 ; // E3.1
        // resto sentencias  { RS.1 }
        end
  end
```

```
process P2 ;
  begin
        while true do begin
        n2 := 1 ; // E1.2
        n2 := n1+1 ; // L1.2 ; E2.2
        while n1 != 0 and //L2.2
          n1<= n2 do begin end;//L3.2
        // seccion critica  { SC.2 }
          n2 := 0 ; // E3.2
        // resto sentencias  { RS.2 }
  end
end
```

68. El siguiente programa es una solución al problema de la exclusión mutua para 2 procesos. Discutir la corrección de esta solución: si es correcta, entonces probarlo. Si no fuese correcta, escribir escenarios que demuestren que la solución es incorrecta.

```
//variables compartidas y valores iniciales
var c0 : integer := 1; c1 : integer := 1 ;
```

```
process P0 ;
      begin
        while true do begin
              repeat
                c0 := 1-c1 ;
              until c1 != 0 ;
              // seccion critica
              c0 := 1 ;
              //resto sentencias }
              end
      end
```

```
process P1 ;
      begin
        while true do begin
              repeat
                c1 := 1-c0 ;
              until c0 != 0 ;
              // seccion critica
              c1 := 1 ;
              // resto sentencias
        end
      end
```

69. Con respecto al algoritmo de Peterson para N–procesos: ¿sería posible que llegaran 2 procesos a la etapa N-2, 0 procesos a la etapa N-3 y en todas las etapas anteriores existiera al menos 1 proceso? Justificar la respuesta.

70. En el *algoritmo de Peterson* para N procesos y considerando cualquier escenario de ejecución de dicho algoritmo, el número máximo de turnos que tiene que esperar cualquier proceso para entrar en sección crítica es N-1 turnos.

71. Con respecto al algoritmo de la siguiente figura (algoritmo de Dijkstra para N procesos), demostrar la falsedad de la siguiente proposición: *si un conjunto de procesos está intentando pasar simultáneamente el primer bucle* (5)*, y el proceso que tiene el turno está* pasivo*, entonces siempre conseguirá entrar primero en sección crítica el proceso de dicho grupo que consiga asignar la variable* turno *en último lugar*.

```
var turn :0..N-1;
flag: array [0 .. N-1] of (pasivo,
solicitando, enSC);
flag:= pasivo;
Process P(i);
begin
(1)   <resto instrucciones>
(2)   repeat
(3)      flag[i] := solicitando;
(4)      j := turn;
(5)      while (turno !=i) do
(6)         if (flag[turno] =  pasivo) then
(7)            turn:=i;
(8)         endif;
(9)    enddo;
(10)    flag[i] := enSC;
(11)    j := 0;
(12)    while ( j < N )and
        ( (j = i)or flag[j]!= enSC )do
              j := j + 1;
(14)    enddo;
```

```
(15) until (j >= N);
<<seccion critica>>
(16) flag[i] := pasivo;
end
```

72. El algoritmo de la figura siguiente (algoritmo de Knuth para N-procesos) resuelve el problema de la exclusión mutua para N-procesos, para lo cual utiliza N variables booleanas, `flag: array[ 0..N - 1 ] of ( solicitando, enSC, pasivo );` una variable `turn: 0..n - 1` y la variable local `j`.

    (a) Demostrar que el algoritmo de Knuth verifica todas las propiedades exigibles a un programa concurrente, incluyendo la de equidad.

    (b) Escribir un escenario en el que 2 procesos consiguen pasar el bucle de la instrucción (5), suponiendo que el turno lo tiene inicialmente el proceso `p(0)`.

```
var flag: array [0 .. N-1] of (pasivo,
solicitando, enSC);
flag:= pasivo;
turn := 0;
Process P(i);
begin
(1)   <resto instrucciones>
(2)   repeat
(3)      flag[i] := solicitando;
(4)      j := turn;
(5)      while (j !=i) do
(6)         if flag[j] !=  pasivo then
(7)            j := turn
(8)         else j := ( j - 1 ) mod N;
(9)         endif;
(10)    enddo;
(11)    flag[i] := enSC;
(12)    j := 0;
(13)    while ( j < N )and
        ( (j = i)or flag[j]!= enSC )do
              j := j + 1;
(14)    enddo;
(15) until (j >= N);
(16) turn := i;
<<seccion critica>>
(17) j := ( turn + 1 ) mod N;
(18) turn := j;
(19) flag[i] := pasivo;
end
```

73. Si en el algoritmo de Dijkstra se cambia la instrucción (6) por esta otra: `if flag[turno] !=SC`, entonces el algoritmo dejaría de ser correcto. Indicar qué propiedad(es) de corrección fallaría(n) y justificar por qué.

74. Si en el algoritmo de Knuth se hacen las siguientes sustituciones:

    - La condición de la instrucción `until` de (15) por la condición: `(j >= N)and( turno=i or flag[turno]= pasivo)`

- Se inserta el siguiente bucle después de la instrucción (17):

```
while (j !=turn )and( flag[j]=pasivo) do
(19)     j := j + 1;
(20) enddo;
```

(a) Verificar las propiedades de exclusión mutua, alcanzabilidad de la sección crítica, vivacidad y equidad del algoritmo.

(b) Calcular el número de turnos máximo que puede llegar a tener que esperar un proceso que quiera entrar en su sección crítica con el algoritmo anterior.

75. Demostrar que las instrucciones (13)-(16) del algoritmo de exclusión mutua distribuido de Ricart-Agrawala no necesitan ser protegidas dentro de la sección crítica definida por las operaciones wait(), signal() del semáforo "s".

```
ns: 0..+INF;
mns: 0..INF;
numrepesperadas:0..n-1;
intentaSC: boolean;
prioridad: boolean;
repretrasadas: array[1..N] of boolean;
```

```
Process Pi;
begin
(1) wait(s);
(2) intentaSC:= TRUE;
(3) ns:= mns +1;
(4) numrepesperadas:=n-1;
(5) signal(s);
(6) for j:= 1 to n do
(7) if j <> i then
(8) send(j, pet, ns, i);
(9) wait(sinc);
 <<seccion critica>>
(10) wait(s);
(11) intentaSC:= FALSE;
(12) signal(s);
(13) for j:=1 to n do
(14) if repretrasadas[j] then begin
(15)    repretrasadas[j]:= FALSE;
(16)    send(j, rep);
(17)    end;
end
```

```
Process Pet(i);
begin
(1)     receive(pet, k, j);
(2)     wait(s);
(3)     mns:= max(mns, k);
(4)     prioridad:= (intentaSC) and (
   k>ns or (k=ns and i<j));
(5)     if prioridad then
   repretrasadas[j]:= TRUE
        else send(j, rep);
(6)     signal(s);
end

Process Rep(i);
begin
(1)     receive(rep);
(2) numrepesperadas:= numrepesperadas
    - 1;
(3) if numrepesperadas= 0 then signal
   (sinc);
end
```

76. Suponer que el algoritmo de Suzuki-Kasami para resolver el problema de la exclusión mutua distribuida para n-procesos se modifica como aparece en la siguiente figura. Explicar por qué dejaría de ser correcto el algoritmo, relacionándolo con cada una de las propiedades de corrección que se demuestran para el algoritmo original.

```
token_presente:boolean:=FALSE;
enSC:boolean:= FALSE;
peticion:array[1..n] of boolean:= FALSE;
//En el algoritmo original ->peticion: array[1..N] of 0..+INF
//ademas se declara otro array-> token: array[1..N] of 0..+INF
```

```
Process P(i);
begin
(0) wait(s);
(1) if NOT token_presente then begin
(2) broadcast(pet, i);
(3) receive(acceso);
(4) token_presente:= TRUE;
(5) end;
(6) enSC:= TRUE;
(7) signal(s);
    <<seccion critica>>
(8) enSC:=FALSE;
(9) wait(s);
(10) for j:= i+1 to n, 1 to i-1 do
(11) if peticion[j] and
   token_presente       then begin
(12) token_presente:= FALSE;
(13) send(j, acceso);
(14) peticion[j]:= FALSE;
(15)end;
(16) signal(s);
end
```

```
Process Pet(i);
begin
(1)      receive(pet, j);
(2)      wait(s);
(3)      peticion[j]:= TRUE;
(4)      if token_presente and NOT
   enSC then
<<< repetir (10)- (16) >>>
end
```