Tema 2

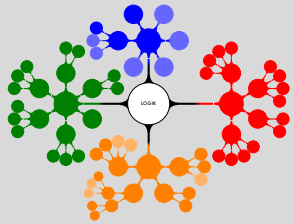
Resolución de la segunda relación de problemas. SCD-GIIM

Sistemas Concurrentes y Distribuidos (SCD)

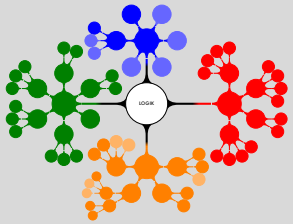
Asignatura *Sistemas Concurrentes y Distribuidos*

Fecha 21 octubre 2021

Departamento de Lenguajes y Sistemas Informáticos
Universidad de Granada



P51



La **propiedad de reentrancia** asegura que cada proceso {P1, P2} "ve" su propia copia de la variable "n" y del contador de programa "IP"

```
procedure entrada() ;
  (1)begin
   (2)n := n+1 ; { ha
llegado un proceso mas }
   (3)if n < 2 then { si
es el primero: }
    (4) s.wait() {
esperar al segundo }
   (3')else begin { si
es el segundo: }
    (4')n := 0; {
inicializa el contador }
    (5')s.signal() {
despertar al primero }
   end
End;
```

```
Monitor Barrera2;
var n : integer; { num. de proc.
que han llegado desde el signal }
s : condition ; { cola donde
espera el segundo }
```

Procedure entrada()

call

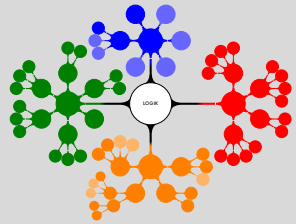
enter monitor

s.wait() ->
exit monitor

stack
P2
n==2
IP==2

stack
P1
n==1
IP==4

P52



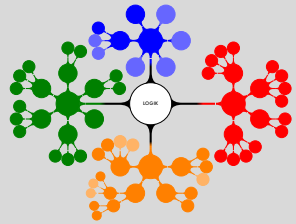
suposiciones:

- Hay que conocer cuántos recursos de cada tipo quedan libres cuando entra un proceso al monitor
- Hay que separar en colas de espera diferentes a los procesos de distinto tipo
- I.M.: $libres[tipo] > 0 \Rightarrow \neg$ cola[tipo].queue() $\wedge$ $libres[tipoA] > 0 \wedge libres[tipoB] > 0 \Rightarrow$ $\neg$ ambos.queue()
- Condiciones lógicas de sincronización:

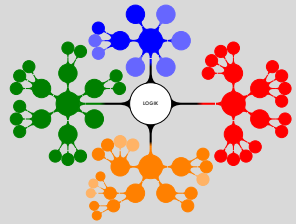
$$C_{tipo} \equiv libres[tipo] > 0,\ tipo = \{tipoA, tipoB\} \equiv \{1, 2\};$$
$$C_3 \equiv libres[tipoA] > 0 \wedge libres[tipoB] > 0$$

P52-a

```
Monitor DosRecursosv1;
var libres: array[1..2] of integer;
    cola: array[1..2] of condition;
procedure pedirRecurso(tipo: integer)
begin
  if libres[tipo] == 0 then
     cola[tipo].wait();
     // C_tipo == true
  libres[tipo]--;
end;
procedure liberarRecursotipo integer)
 begin
   libres[tipo]++;
   // C_tipo == true
   cola[tipo].signal();
 end;
begin
  libres[1]= N1;
  libres[2]= N2;
   //Se cumple el IM
end;
```

P52-b



Escenario de interbloqueo entre dos procesos intentando acceder a los 2 recursos secuencialmente (en distinto orden)

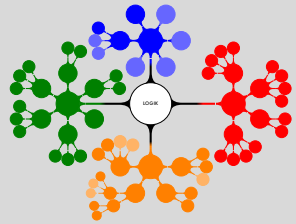
```
Process P1
  begin
    DosRecursosv1.pedirRecurso(1);
    DosRecursosv1.pedirRecurso(2);
  end;
Process P2
  begin
    DosRecursosv1.pedirRecurso(2);
    DosRecursosv1.pedirRecurso(1);
  end;

cobegin P1; P2 coend;
```

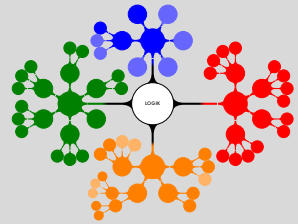
P52-c

```
Monitor DosRecursosv2;
var libres: array[1..2] of integer;
    cola: array[0..2] of condition;
procedure pedirRecurso(tipo: integer)
begin
  if (tipo==0) then begin
    if libres[1]==0 or libres[2]==0 then
       cola[0].wait();
       //condicion sincronizacion C_3 == true
    libres[1]--1; libres[2]--;
  end
    else begin
      if libres[tipo]==0 then
        cola[tipo].wait();
        //condicion sincronizacion C_tipo == true
      libres[tipo]--;
    end
end;
```

P52-c

```
procedure liberarRecursotipo integer)
 var otrotipo: integer = (1+(tipo % 2));
 begin
   libres[tipo]++;
   if (libres[otrotipo] > 0)
   //C_3 == true
      cola[0].signal();
      else
        //C_tipo == true
        cola[tipo].signal();
 end;
begin
  libres[1]= N1;
  libres[2]= N2;
  //Se cumple el IM
end;
```



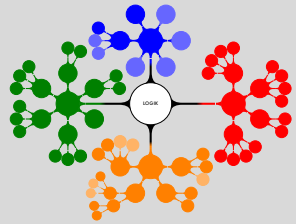
suposiciones:

- Se pueden ejecutar concurrentemente los procesos de carácter lector (”lectores”), pero se excluyen con cualquier proceso de carácter escritor (”escritores”)
- Los escritores se excluyen con lectores y escritores
- I.M.:

$$(num\_lectores == 0 \lor num\_escritores == 0)$$
$$\land$$
$$0 \leq num\_escritores \leq 1$$

- Condiciones lógicas *generales* de sincronización (sin tener en cuenta prioridades):
  1. $C\_e \equiv escribiendo == false \land num\_lectores == 0$
  2. $C\_l \equiv escribiendo == false$
- Las prioridades a los procesos lectores o escritores no se pueden expresar en el invariante y se han de programar adicionalmente

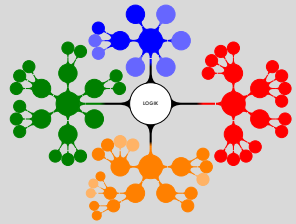
P53



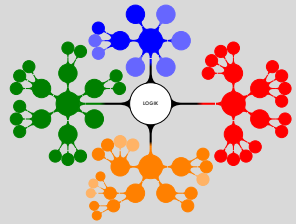

```
Monitor LectoresEscritores //prioridad lectores
  var escribiendo: boolean;
      num_lectores: integer;
      lectores, escritores: condition;
  procedure escritura_ini()
  begin
    if escribiendo or num_lectores > 0 then
       escritores.wait();
    //C_e == true
    escribiendo= true;
  end;
  procedure escritura_fin()
  begin
    escribiendo= false;
    //C_l==true && C_e==true ya que num_lectores==0 aqui
    if lectores.queue() then
      //se ha de programar prioridad a los lectores
      lectores.signal();
      else  escritores.signal();
  end;
```

P53

```
procedure lectura_ini()
begin
  if escribiendo then
     lectores.wait();
     //C_l == true
  num_lectores++;
  //C_l==true
  lectores.signal();
end;
procedure lectura_fin()
begin
  num_lectores--;
  if num_lectores == 0 then
    //C_e == true porque habia algun proceso leyendo
    // y, por tanto, escribiendo==false aqui
    escritores.signal();
end;
begin
  num_lectores= 0;
  escribiendo= false;
  //Se cumple el IM
end;
```

P54

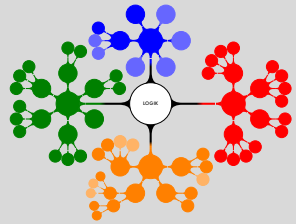


suposiciones

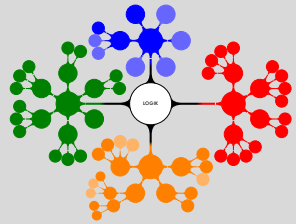
- I.M. $N\_cruzando == 0 \vee S\_cruzando == 0$
- Condiciones lógicas de sincronización:
  1. $C\_norte \equiv S\_cruzando == 0$
  2. $C\_sur \equiv N\_cruzando == 0$

P54-a

```
Monitor Puente
  var N_cruzando, S_cruzando: integer;
      norte, sur: condition;
  procedure EntrarCocheDelNorte
  begin
    if S_cruzando >0 then
      norte.wait();
    //C_norte == true
    N_cruzando ++ ;
    //C_norte == true
    norte.signal();  end;
  procedure SalirCocheNorte
  begin
    N_cruzando --;
    if N_cruzando == 0 then
    //C_sur == true
      sur.signal();  end;
  procedure EntrarCocheDelSur
  //totalmente simetrico al EntrarCocheDelNorte
  procedure SalirCocheDelSur
  // totalmente simetrico al SalirCocheDelNorte
begin
 N_cruzando= 0;
 S_cruzando= 0;
end;
```

P54-b

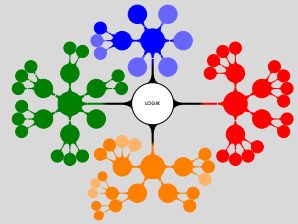


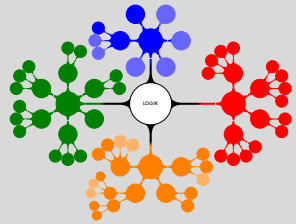
Para impedir que los coches de un lado monopolicen el puente

- Se puede establecer un máximo de coches que crucen en línea de cada lado antes de dar paso a los del lado opuesto
- N_pueden y S_pueden se inicializan a N_0 (== 10) y se van disminuyendo si hay coches esperando del otro lado para cruzar el puente
- Se podría modificar el I.M. del monitor para tener en cuenta esta restricción
- Las condiciones lógicas de sincronización se convierten en:
  1. $C\_norte' \equiv S\_cruzando == 0 \wedge N\_pueden > 0$
  2. $C\_sur' \equiv N\_cruzando == 0 \wedge S\_pueden > 0$

P54-b

```
Monitor PuenteConLimite
  var N_cruzando, S_cruzando, N_pueden, S_pueden: integer
      norte, sur: condition;
  procedure EntrarCocheDelNorte
  begin
    if (S_cruzando >0 or N_pueden == 0) then
      norte.wait();
    //C'_norte == C_norte == true && N_pueden > 0
    N_cruzando ++ ;
    if sur.queue() then N_pueden -- ;
    if N_pueden > 0 then
    //C'_norte == C_norte == true && N_pueden > 0
       norte.signal();    end;
  procedure SalirCocheNorte
  begin
    N_cruzando --;
    if N_cruzando == 0 then begin
      S_pueden= 10;
    //C'_sur ==C_sur == true && S_pueden > 0
      sur.signal();
    end;  end;
...
 begin N_cruzando= 0; N_pueden= 10;
 S_cruzando= 0; S_pueden= 10;  // Se cumple el IM
 end;
```

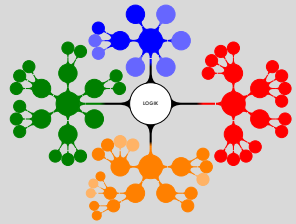


suposiciones

- 2 variables condición: cocinero, salvajes y una variable entera que cuenta el número de misioneros en la olla
- I.M.: $num\_misioneros > 0 \Rightarrow \neg salvajes.queue() \wedge num\_misioneros == 0 \Rightarrow \neg cocinero.queue()$
- Condición lógica de sincronización:
  1. $C\_salvajes \equiv num\_misioneros > 0$
  2. $C\_cocinero \equiv num\_misioneros == 0$
- Hay que resolverlo con 2 procedimientos (`Dormir()` y `RellenarOlla()`) que llamará el proceso cocinero y hay que impedir que se pierda la señal que despierta a los salvajes cuando se rellena la olla ($\Rightarrow$ hacer uso de la semántica SU de señales de monitores para conseguirlo)

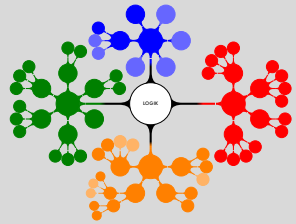
P55

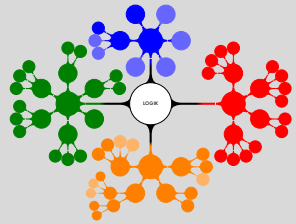
```
Monitor Olla;
var num_misioneros: integer;
    salvajes, cocinero: condition;
procedure ServirseUnMisionero()
begin
  if num_misioneros == 0 then begin
    //C_cocinero == true
    cocinero.signal();// con seniales SU pierde el monitor
                      // pero mantiene la prioridad frente
                      // a procesos en cola de entrada
    salvajes.wait();
    //C_salvajes == true
    end;
 num_misioneros -- ;
 if (num_misioneros  > 0) then
    //C_salvajes == true
    salvajes.signal();
    else //no quedan misioneros en la olla
      //C_cocinero == true
      cocinero.signal()
    end;
end;
```

P55

```
procedure Dormir()
begin
  if (num_misioneros > 0) then
    cocinero.wait();
   //C_cocinero == true
end; //sale del monitor
procedure RellenarOlla()
//tiene que pasar por la cola de entrada
begin
  num_misioneros= M;
  //C_salvajes == true
  salvajes.signal();
end;
begin  num_misioneros= M; //Se cumple el IM
end;
process Salvaje[i:1 .. N]()
begin
  while true do begin
    Olla.ServirseUnMisionero(); //Comer
  end   end;
process Cocinero()
begin
  while true do begin
    Olla.Dormir();
    Olla.RellenarOlla();  end  end;
```

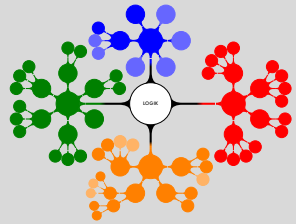


suposiciones:

- Se resuelve con una variable booleana `libre` (que indica: "R no está ocupado") y una cola para bloquear a los procesos cuando está ocupado R
- I.M.:
  $libre == true \Rightarrow \forall i : 1 \leq i \leq n : peticion[i] == false \wedge$
  $\exists i : 1 \leq i \leq n : peticion[i] == true \Rightarrow libre == false \wedge$
  $0 \leq \sum_{i=1}^{n} num\_peticion[i] == true \leq 1$ (suponer aritmetización de los valores lógicos: true, false)
- Condición lógica de sincronización:
  1. $C : \; libre == true \wedge \sum_{i=1}^{n} num\_peticion[i] == 0$

P57

```
MonitorRecursoR;
 var libre: boolean; recurso: condition;
 procedure pedir(i: integer)
   begin
     if (not libre) then
       recurso.wait(i);
     //C == true
     libre= false;
     //peticion[i] == true
   end;
 procedure liberar()
   begin
     libre= true;
     //C  == true
     //peticion[i] == false
     recurso.signal();
   end;
 begin
   libre= true; //Se cumple el IM porque peticion[i]==
       false , para todo "i"
 end;
```

P57-bis

Nuevo enunciado:

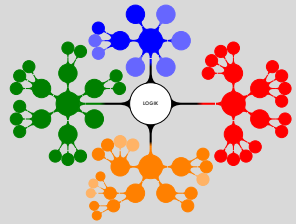
Suponer ahora que “n" procesos comparten “m" recursos. Antes de utilizar un recurso, el proceso P_i llama a la operación `function pedir(id_proceso:1..n):integer`, que devuelve el número de recurso libre. En caso de que no existiera ningún recurso libre en ese momento, la llamada al método anterior ocasiona que el proceso P_i se quede esperando 1 recurso. Los procesos después de utilizar una recurso llaman a `procedure devolver(id_recurso:1..m)`. Cuando un proceso deja de utilizar un recurso, éste queda *libre* y si hay varios procesos esperando un recurso, se le concederá al proceso de mayor prioridad (menor índice) de los que esperan. La prioridad se mantiene como en el enunciado del ejercicio 57.

suposiciones:

- I.M. que resuelve el acceso seguro de ”n” procesos a ”m” recursos:

$$\exists j : 1 \leq j \leq m : libre[j] == true \quad \rightarrow \quad \forall i : 1 \leq i \leq n : peticion[i] == false$$
$$\wedge \ \exists i : 1 \leq i \leq n : peticion[i] == true \quad \rightarrow \quad \forall j : 1 \leq j \leq m : libre[j] == false$$

- Condición lógica de sincronización:
  1. $C[i] : \exists j : 1 \leq j \leq m : libre[j] == true \wedge peticion[i] == false$ (más prioritaria de las pendientes)

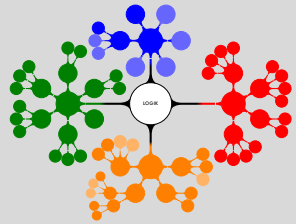


P57-bis

```
Monitor Recursos;
var
   libre: array[1..m] of boolean;
   peticion: array[1..n] of boolean;
   I:integer;//para guardar indice recurso liberado
   no_peticion: condicion;
```

```
function pedir(id_proceso
    :1..n):integer;
  var k:integer;
begin
  repeat
    k:= k + 1;
  until(libre[k] or k==m);
  if (not libre[k]) then
    while(peticion[
        id_proceso]) do
        begin
      no_peticion.signal();
      no_peticion.wait();
  //C[id_proceso] == true
      end
        else I=k;
  libre[I]:= false;
  return I;
end
```

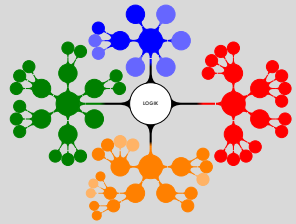
```
procedure devolver(
    id_recurso:1..m);
  var j: integer;
begin
  repeat
    j:= j+1;
until (peticion[j]or j==n);
  if (peticion[j]) then
     begin
    I:= id_recurso;
    peticion[j]:= false;
    //C[j] == true
    no_peticion.signal();
    end
    else libre[I]:= true;
end
```

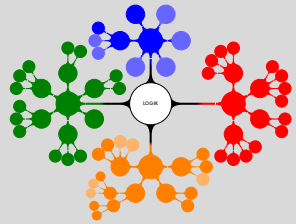
P58

suposiciones:

- Un persona A no puede salir de la habitación si no hay dentro, al menos, 10 personas B
- Un persona B no puede salir de la habitación si no había dentro, al menos, 1 persona A y 9 personas B
- En cuanto se dan las condiciones de salida, no se dejan entrar más personaas a la habitación hasta que no salga el grupo (1 persona de tipo A y 10 personas de tipo B)
- En la simulación, las personas se representan por procesos y no se requiere que estos se desbloqueen en orden FIFO y suponemos semántica de señales SU
- I.M.: num_procesosB_dentro < 10 or num_procesosA_dentro < 1
- Condición lógica de sincronización:
  1. C_B : $num\_procesosB\_dentro == 10 \wedge num\_procesosA\_dentro > 0$
  2. C_A : $num\_procesosB\_dentro \geq 10 \wedge num\_procesosA\_dentro == 1$



P58

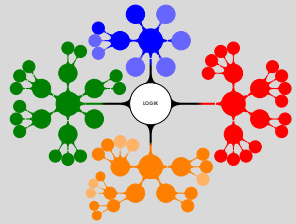



```
Monitor Habitacion;
var num_procesosB_dentro, num_procesosA_dentro: integer;
    num_procesosB_saliendo,num_procesosA_saliendo:integer;
    dentroA, dentroB: condition;
```

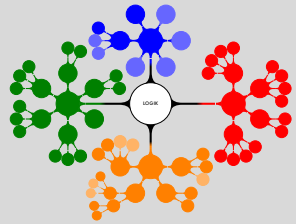
```
procedure entraSaleA()
 begin
   num_procesosA_dentro++;
   if (num_procesosB_dentro < 10) then
    dentroA.wait();
    \\CA == true
   num_procesosA_saliendo ++ ;
   \\CB == true
   dentroB.signal();
end;
```

P58

```
procedure entraSaleB()
 begin
   num_procesosB_dentro ++ ;
   if (num_procesosA_dentro==0 or num_procesosB_dentro<10)
     then dentroB.wait();
   \\CB == true
   if ( num_procesosA_saliendo == 0) then
      \\CA == true
      dentroA.signal();
   if ( num_procesosB_saliendo < 10) then
     \\CB == true
     dentroB.signal();
   else \\ Se han de modificar todas las variables
        \\ para que no se cuelen mientras salen
      Inicia();
 end;
begin Inicia()
  num_procesosB_dentro-=10;num_procesosA_dentro-=1;
  num_procesosB_saliendo,num_procesosA_saliendo= = {0,0};
  //Se cumple el IM
end;
```

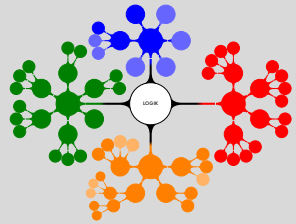
P60-a



suposiciones:

- I.M. : $0 \leq num\_tenedores\_ocupados \leq 5$
- Condición lógica de sincronización:
  $\forall i : 0 \leq i \leq 4 : C[i] : num\_tenedores\_ocupados < 5$
  $\wedge$ ($tenedor\_ocupado[i] == false \vee$
  $tenedor\_ocupado[i + 1 \ mod \ 5] == false$)
- No tener en cuenta la situación de interbloqueo que puede ocurrir con una primera solución al problema

P60-a

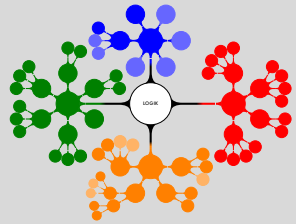



```
Monitor FilosofosNoSeguro;
var
   ten_ocupado: array[0..4] of boolean;
   tenedor: array[0..4] of condition;
```

```
procedure coge_tenedor(
    num_ten, num_proc :
    integer )
begin
  if ten_ocup[num_ten] then
   tenedor[num_ten].wait();
    //C[i] == true
  ten_ocup[num_ten]= true ;
end
```

```
procedure liberar_tenedor(
    num_ten:integer);
begin
  ten_ocup[num_ten]= false;
  //C[num_ten]==true
 tenedor[num_ten].signal();
  //avisar a alguno que lo
      esperaba, si hay
end
```

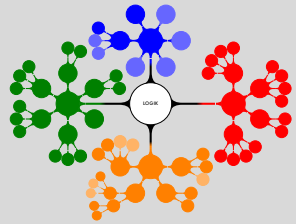
P60-c



suposiciones (2da solución):

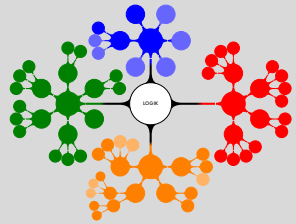
- I.M. : $0 \leq num\_filosofos\_prep\_avanzar \leq 4$
- Condiciones lógicas de sincronización:
  $\forall i : 0 \leq i \leq 4 : C[i] : num\_tenedores\_ocupados \leq 4$
  $\wedge$ ($tenedor\_ocupado[i] == false \vee$
  $tenedor\_ocupado[i + 1 \ mod \ 5] == false$)
  $C_{previa} == filosofos\_con\_1er\_y\_sin\_2do\_tenedor < 4$

```
Monitor Filosofos;
var
   ten_ocupado: array[0..4] of boolean;
   tenedor: array[0..4] of condition;
   previa: condition;
   num_f12: integer //numero filosofos con 1 tenedor
procedure coge_tenedor( num_ten, num_proc : integer )
begin
  if num_ten == num_proc then begin
    if num_f12 == 4 then
        previa.wait()
        //C_previa == true
        //preparado para avanzar
    end
  if ten_ocup[num_ten] then
     tenedor[num_ten].wait() ;
     //C[num_proc]==true
     ten_ocup[num_ten] := true ;
  if num_ten == num_proc then
    num_f12= num_f12+1 ;
    else begin
      num_f12= num_f12-1 ;
      if num_f12 < 4 then
      //C_previa == true
        previa.signal() ;
    end
    {IM}
end
```

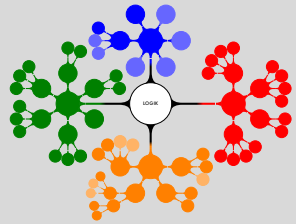



P60-c

```
procedure liberar_tenedor(num_ten:integer);
 begin
   ten_ocup[num_ten]= false;
   //C[num_ten] == true OR C[num_ten+1 mod5]==true
 //se activara num_fil==num_ten o num_fil==num_ten+1 mod 5
   tenedor[num_ten].signal();
   {IM}
 end

begin
  num_f12 = 0 ;
  // hay 0 filósofos con 1er y sin 2do tenedor
  for i = 0 to 4 do
    ten_ocup[i]= false ; // tenedores libres inicialmente
  {IM}
end
```

P56-a1

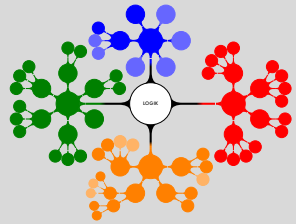



```
Monitor CuentaAhorrosFIFOsinPrioridad;
var saldo: integer;
    cola: condition;
```

```
 procedure retirar(cantidad
     : positive);
  begin
    while cantidad > saldo
        do begin
      cola.signal();
      cola.wait();
    end;
    saldo -= cantidad;
    cola.signal();
begin
```

```
procedure depositar(
    cantidad: positive)
begin
 saldo += cantidad;
 cola.signal();
end
```

P56-a2

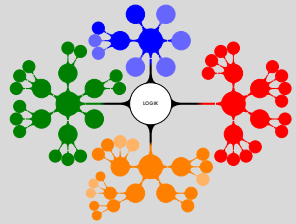



```
Monitor CuentaAhorrosColaConPrioridad;
var saldo: integer;
    cola: condition;
```

```
 procedure retirar(cantidad
     : positive);
  begin
    while cantidad > saldo
        do begin
      cola.wait(cantidad);
    end;
    saldo -= cantidad;
    cola.signal();
begin
```

```
procedure depositar(
    cantidad: positive)
begin
 saldo += cantidad;
 cola.signal();
end
```

P56-b1

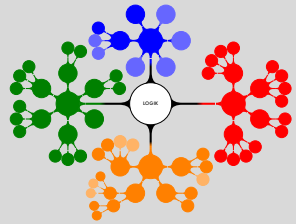



```
Monitor CuentaAhorrosOrdenLlegadaSinPrioridad;
var saldo: integer;
    ventanilla, resto: condition;
```

```
 procedure retirar(cantidad
     : positive);
  begin
    if (ventanilla.queue())
         then
       resto.wait();
    while cantidad > saldo
        do begin
      ventanilla.wait();
    end;
    saldo -= cantidad;
    resto.signal();
begin
```

```
procedure depositar(
    cantidad: positive)
begin
 saldo += cantidad;
 ventanilla.signal();
end
```

P56-b1

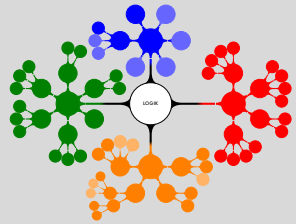



```
Monitor CuentaAhorrosOrdenLlegadaConPrioridad;
var saldo, contador: integer;
    cola: condition;
```

```
 procedure retirar(cantidad
     : positive);
  var ticket: integer;
  begin
    ticket= contador;
        contador ++ ;
    if (cola.queue()) then
       cola.wait(ticket);
    while cantidad > saldo
        do begin
      cola.wait(ticket);
    end;
    saldo -= cantidad;
    cola.signal();
 end;
```

```
procedure depositar(
    cantidad: positive)
begin
 saldo += cantidad;
 cola.signal();
end
begin
  saldo= CANTIDAD_INICIAL;
  contador= 0;
end;
```

P59




```
//Monitor Barrera2ConSemaforos;
var n: integer;
    s: Semaphore= 0;
    mutex: Semaphore= 1;
```

```
procedure entrada();
 begin
    sem_wait(mutex);
    n ++ ;
    if (n < 2) then begin
       sem_signal(mutex);
       sem_wait(s);
    end
```

```
  else begin
    n= 0;
    sem_signal(s);
    sem_signal(mutex);
  end
end;
```