EJERCICIOS PROPUESTOS CONCURRENCIA

(Soluciones)

(1) Diseñar mediante paso de mensajes, un proceso llamado “Controlador” que provoque que los dos primeros procesos que lo invoquen queden bloqueados y el tercero los despierte, y así cíclicamente.

```
program ejercicio
type
   buzon: mailbox of integer;
const
   numproc=10;
var
   pido_permiso:buzon
   permiso_concedido: array[1..numproc] of buzon;

process type Proceso(id:integer)
var
   mensaje:integer;
   mi_id=integer;
begin
   mi_id=id;
   repeat
      ...
      send(pido_permiso,mi_id);
      receive(permiso_concedido[mi_id], mensaje)
      ...
   forever
end;

process Controlador;
var
   pids: array[1..3] of integer;
   cont:integer;
begin
   cont:=0;
   repeat
      cont:=cont+1;
      receive(pido_permiso, pids[cont]);
      if cont==3 then
         begin
            for cont=1 to 3 do
               send(permiso_concedido[pids[cont]],cont);
            cont:=0;
         end;
   forever
end;

begin
   cobegin
      Controlador;
      for contador:=1 to numproc do
         Proceso(contador);
   coend
end;
```

(2) Diseñar mediante monitores, un proceso llamado "Controlador" que provoque que los dos primeros procesos que lo invoquen queden bloqueados y el tercero los despierte, y así cíclicamente.

```
monnitor x
  export: entrada;
  var n:integer;
      s:contition;

  procedure entrada
  begin
     n:=n+1;
     if n<3 then delay(s)
     else
        begin
           n:=0;
           resume(s);
           resume(s);
        end;
  end;

begin
  n:=0;
end;

x.entrada();        x.entrada()       x.entrada()
```

(2-b) Diseñar mediante semáforos, un proceso llamado "Controlador" que provoque que los dos primeros procesos que lo invoquen queden bloqueados y el tercero los despierte, y así cíclicamente.

```
semaphore llegada = 0;       //Controla la llegada de procesos
semaphore controlador = 0;   //
```

```
void Controlador (void) {
   i: integer;
   while (true) {
      for (i=1; i<=3; i++)
         wait(llegada);
      signal(controlador);
      signal(controlador);
      signal(controlador);
   }
}
```

```
void proceso(void) {
   ……
   signal(llega);
   wait(controlador);
   ….
}


main(){
   parbegin
      Controlador();
      for (i=0;i<num_procesos; i++)
         proceso(i);
   end
}
```

(3) Una serie de procesos envían un mensaje a un proceso controlador a través de un buzón llamado “mebloqueo”. Todos los procesos que envían un mensaje a dicho buzón son bloqueados hasta que otro proceso envía un mensaje a otro buzón llamado “desbloqueo”. Los procesos bloqueados hasta el momento deben ser liberados en el orden en que enviaron los mensajes. Cuando todos estos procesos han sido liberados, el proceso liberador podrá continuar. Desarrollar la solución mediante paso de mensajes.

```
program ejercicio
type
   item=integer;
   buzon: mailbox of item;
const
   numproc=10;
var
   mebloqueo:buzon;
   desbloqueo:buzon;
   continuo:array[0..numproc-1] of buzon;

   procesos: array[0..numproc-1] of Proceso;
   cont:integer;

process type Processo(id:integer)
var
   tipo: integer;
   menseje:item;
begin
   repeat
      tipo:=random(1);
      if tipo=1 then
         send(mebloqueo,id);
      else
         send(desbloqueo,id);
      receive(continuo[id],mensaje);
   forever;
end;
```

```
process Controlador
var
   id:item;
   cont:integer;
   bloqueados:array[0..numproc-1] of integer;
   orden:integer
begin
   orden:=0;
   repeat
      select
         receive(mebloqueo,id);
         bloqueados[orden]:=id;
         orden:=mod(orden+1,10);
      or
         receive(desbloqueo,id);
         bloqeados[orden]:=id;
         orden:=0;
         for cont:=0 to numproc-1
            begin
               id=bloqueados[cont];
               send(continuo[id],0);
            end;
      end
   forever
end;

begin
   cobegin
      Controlador;
      for cont:=0 to numproc-1
         Proceso[cont](cont);
   coend;
end;
```

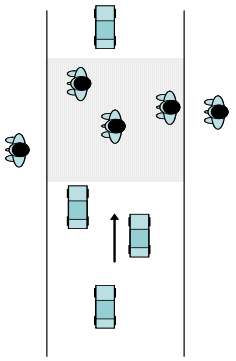
(4) Una serie de procesos envían un mensaje a un proceso controlador a través de un buzón llamado “mebloqueo”. Una parte del contenido del mensaje es un valor entero **n**. Todos los procesos que envían un mensaje a dicho buzón son bloqueados hasta que otro proceso envía un mensaje a otro buzón llamado “desbloqueo”. Los procesos bloqueados hasta el momento deben ser liberados en el orden establecido por el parámetro n. Cuando todos estos procesos han sido liberados, el proceso liberador podrá continuar.

Desarrollar la solución mediante paso de mensajes.

```
program ejercicio
type
   item=struct
          n:integer;
        end
   buzon: mailbox of item;
const
   numproc=10;
   maxn=10;
var
   mebloqueo:buzon;
   desbloqueo:buzon;
   continuo:array[1..numproc] of buzon;

   procesos: array[1..numproc] of Proceso;
   cont:integer;

process type Processo(id:integer)
var
   n, tipo: integer;
   mensaje:item;
begin
   n:=random(1..maxn);
   repeat
      tipo:=random(0..1);
      mensaje.n=n;
      if tipo=1 then
         send(mebloqueo,mensaje);
      else
         send(desbloqueo,mensaje);
      receive(continuo[id],mensaje);
   forever;
end;
```

```
process Controlador
var
   mensaje:item;
   cont,pos,id:integer;
   bloqueados:array[1..maxn,     1..numproc]     of
integer;
   contadores:array[1..maxn] of integer;
begin
   for cont:=1 to maxn
      contadores[cont]:=0;
   repeat
      select
         receive(mebloqueo,mensaje);
         contadores[mensaje.n]+=1;
         pos:==contadores[mensaje.n];
         bloqueados[mensaje.n,pos]:=id;
      or
         receive(desbloqueo,mensaje);
         contadores[mensaje.n]+=1;
         pos:=contadores[mensaje.n];
         bloqeados[mensaje.n,pos]:=id;
         for cont:=1 to maxn
            begin
               for pos:=1 to contadores[cont]
                  begin
                     id:=bloqueados[cont,pos];
                     send(continuo[id],mensaje);
                  end;
               contadores[cont]:=0;
            end;
      end
   forever
end;

begin
   cobegin
      Controlador;
      for cont:=1 to numproc
         Proceso[cont](cont);
   coend;
end;
```

(5) Sea una carretera en la que hay una zona de paso para peatones. La zona de cruce es suficientemente ancha para que quepan varios coches dentro, es decir, pueden estar sobre ella varios coches a la vez y por supuesto varios peatones a la vez.



Los peatones tienen preferencia, de forma que mientras haya peatones cruzando, los coches se pararan y esperaran a que el último peaton abandone la zona de cruce. Los coches podran pasar mientras no llegue ningún peaton, cuando llegue uno, se pararán y esperarán a que el peaton abandone la zona de cruce.

Modelar el problema utilizando:

- (a) Semáforos.
- (b) Monitores
- (c) Paso de Mensajes

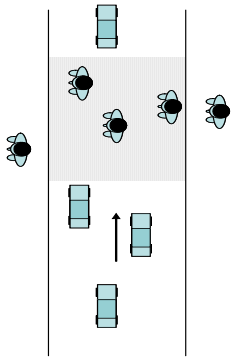
(5a) Solucion Semaforos

```
semaphore cri_coche = 1;     //Controla el acceso a la región crítica de los coches
semaphore cri_peaton = 1;    //Controla el acceso a la región crítica de los peatones
semaphore cruce = 1;         //Controla el acceso a la zona de cruce.
semaphore perm_coches = 1;   //Permite parar a los coches en el momento que un peatón desee cruzar.
int cont_coches = 0;         //Contador de coches en la zona de cruce
int cont_peatones = 0;       //Contador de peatones en la zona de cruce
```

```
void coche (void) {
   while (true) {
      wait(perm_coches);
         wait(cri_coche);
            cont_coches++;
            if (cont_coches ==1)
               wait(cruce);
         signal(cri_coche);
      signal(perm_coches)

      CruzarCoche();

      wait(cri_coche);
         cont_coches--;
         if (cont_coches ==0)
            signal(cruce);
      signal(cri_coche);
   }
}
```

```
void peaton(void) {
   while (true) {
      wait(cri_peaton);
         cont_peatones++
         if (cont_peatones == 1) {
            wait(perm_coches);
            wait(cruce);
            }
      signal(cri_peaton);

      CruzarPeaton();

      wait(cri_peaton);
         cont_peatones--;
         if (cont_peatones == 0) {
            signal(perm_coches);
            signal(cruce);
            }
      signal(cri_peaton);
   }
}
```

(5b) Solución Monitores

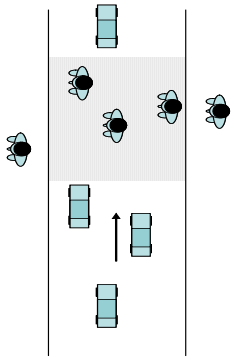


```
monitor paso_cebra {

export
   nuevo_coche();
   llega_coche();
   sale_coche();
   llega_peaton();
   sale_peaton();

condition
   peatones_esparando,
   coches_parados,
   paso_ocupado;

var
   int peatones_cruzando,
       coches_cruzando,
       num_coches;


procedure nuevo_choche(){
   num_coches++;
   }

procedure llega_coche(){
   if (!empty(peatones_esperando) ||
      (peatones_cruzando>0){
        resume(peatones_esperando);
        delay(coches_parados);
      }
   coches_cruzando++;
   num_coches--;
   resume(coches_parados);
}

procedure sale_coche(){
   coches_cruzando--;
   if (coches_cruzando==0)
      resume(paso_ocupado);
}

procedure llega_peaton(){
   if((peatones_cruzando==0)&&
       (num_coches>0))
       delay(peatones_esperando);
   peatones_cruzando++;
   if coches_cruzando>0{
      delay(paso_ocupado);
      }
   resume(paso_ocupado);
}

procedure sale_peaton(){
   peatones_cruzando--;
   if (!empty(coches_parados) &&
       peatones_cruzando==0)
       resume(coches_parados);
}

init(){
   peatones_cruzando=0;
   coches_cruzando=0;
   }
}

process type coche(){
   paso_cebra.nuevo_coche(); /*Acercarse al paso de cebra*/
   paso_cebra.llega_coche();
      /*Coche cruza paso cebra*/
   paso_cebra.sale_coche();
}

process type peaton(){
   paso_cebra.llega_peaton()
     /*Peaton cruza paso cebra*/
   paso_cebra.sale_peaton();
}
```

(5b) Otra Solucion Monitores

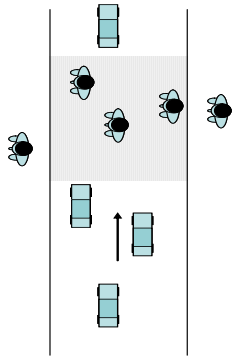
```
monitor paso_cebra {

export
   llega_coche();
   sale_coche();
   llega_peaton();
   sale_peaton();

condition
   coches_parados,
   paso_ocupado;

var
   int peatones_cruzando,
       coches_cruzando;
   bool bloquar_coches;


procedure llega_coche(){
   if (bloqueo_coches){
      delay(coches_parados);
      resume(coches_parados);
      }
   coches_cruzando++;
}

procedure sale_coche(){
   coches_cruzando--;
   if (coches_cruzando==0)
      resume(paso_ocupado);
}
```

```
procedure llega_peaton(){
   bloqueo_coches=true;
   peatones_cruzando++;
   if coches_cruzando>0{
      delay(paso_ocupado);
      resume(paso_ocupado);
      }
}

procedure sale_peaton(){
   peatones_cruzando--;
   if (peatones_cruzando==0){
       bloqueo_coches=false;
       resume(coches_parados);
       }
}

init(){
   peatones_cruzando=0;
   coches_cruzando=0;
   bloquear_coches=false;
   }
}

process type coche(){
   paso_cebra.llega_coche();
   /*Coche cruza paso cebra*/
   paso_cebre.sale_coche();
}

process type peaton(){
   paso_cebra.llega_peaton()
    /*Peaton cruza paso cebra*/
   paso_cebra.sale_peaton();
}
```



(5c) Solucion Mensajes

```
program ejercicio
type
   item=...
   buzon: mailbox of item;
var
   peaton_pide_paso:buzon;
   coche_pide_paso:buzon;
   peaton_concedido:buzon;
   coche_concedido:buzon;

process type Peaton
var
   mensaje:item;
begin
   mensaje.id=id;
   send(peaton_pide_paso,mensaje);
   receive(peaton_concedido,mensaje);
   /*Cruzando*/
   send(peaton_sale,mensaje);
end;

process type Coche
var
   mensaje:item;
begin
   mensaje.id=id;
   send(coche_pide_paso,mensaje);
   receive(coche_concedido,mensaje);
   /*Cruzando*/
   send(coche_sale,mensaje);
end;
```



```
process Controlador
var
   mensaje:item;
   coches_pasando,peatones_pasando integer;
   coches_parados,peatones_parados integer;
   bloqueo_coches boolean
begin
   coches_pasando=0; peatones_pasando=0;
   coches_parados=0; peatones_parados=0;
   bloqueo_coches=false;
   repeat
      select
         receive(peaton_pide_paso,mensaje);
         bloqueo_coches=true;
         if coches_pasando==0 begin
            send(peaton_concedido,mensaje);
            peatones_pasando++;
            end
         else
            peatones_parados++;
      or
         receive(coche_pide_paso,mensaje);
         if (not bloqueo_coches)begin
            send(coche_concedido,mensaje);
            coches_pasando++;
            end
         else
            coches_parados++;
      or
         receive(peaton_sale,mensaje);
         peatones_pasando--;
         if peatones_pasando==0 begin
            bloqueo_coches=false;
            for cont=1 to coches_parados
               begin
                  send(coche_concedido,mensaje);
                  coche_pasando++;
               end
             coches_parados=0;
            end
      or
         receive(coche_sale,mensaje);
         coches_pasando--;
         if coches_pasando==0 begin
            for cont=1 to peatones_parados
               begin
                  send(peaton_concedido,mensaje);
                  peatones_pasando++;
               end
            peatones_parados=0;
            end
      end
   forever
end;
```

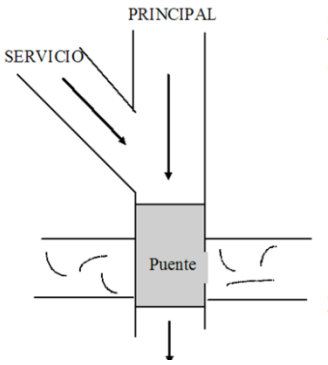
(6) Una serie de procesos envían un procedimiento de un monitor llamado “Bloquear(n)”, donde n es un valor entero entre 1 y 10. Todos los procesos que invocan dicho procedimiento son bloqueados hasta que otro proceso invoca al proceimiento “Liberar()”. Los procesos bloqueados hasta el momento deben ser liberados en el orden establecido por el parámetro n. Cuando todos estos procesos han sido liberados, el proceso liberador podrá continuar. Desarrollar la solución mediante monitores.

```
monitor bloq-lib
   export bloquear, liberar
   var bloqueados:array[1..10] of condition;

procedure bloquear(n:integer)
begin
   delay(bloqueados[n]);
end;

procedure liberar
var i:integer;
begin
   for i:=1 to 10
      while (not empty(bloqueados[i])
         resume(bloqueados[i]);
end

begin
end;
```

(6b) Una serie de procesos envían un procedimiento de un monitor llamado “Bloquear(n)”, donde n es un valor entero entre 1 y 10. Todos los procesos que invocan dicho procedimiento son bloqueados hasta que otro proceso invoca al procedimiento “Liberar()”. Además, como máximo puede haber 20 proceoss bloqueados, de forma que el vigesimoprimer proceso consecutivo que llama a bloquear, es decir, ya hay 20 bloqueados, procede a la liberación de los demás. Los procesos bloqueados hasta el momento deben ser liberados en el orden establecido por el parámetro n. Cuando todos estos procesos han sido liberados, el proceso liberador podrá continuar. Desarrollar la solución mediante monitores.

```
monitor bloq-lib
   export bloquear, liberar
   var bloqueados:array[1..20] of condition;
   var contador;

procedure bloquear(n:integer)
begin
   if contador==20
      liberar();
      contador:=0;
   else
     delay(bloqueados[n]);
    end
   contador++;
end;

procedure liberar
var i:integer;
begin
   for i:=1 to 20
      while (not empty(bloqueados[i])
         resume(bloqueados[i]);
   contador:=0;
end

begin
   contador:=0;
end;
```

(7) En la entrada a un antiguo puente de madera unidireccional confluyen dos carreteras, la carretera principal y la carretera de servicio. Los vehículos que vienen por ambas carreteras deben cruzar el puente pero tienen que tener en cuenta las siguientes consideraciones:



- El puente solo puede soportar el peso de 5 vehículos
- Cuando haya vehículos en la carretera principal que quieran cruzar el puente tendrán siempre prioridad frente a los vehículos de la carretera de servicio
- Los vehículos que estén en la carretera de servicio podrán empezar a cruzar el puente cuando no exista ningún vehiculo en la carretera principal que quiera cruzar el puente.

Implementa el código que gestiona la circulación de los vehículos mediante semáforos, explicando la implementación realizada.

```
semaforo np=0;            //contador de numero de coches en carretera principal
semaforo_bin mutex=0;     // para exclusión mutua contador
semaforo_bin cs=0;        // Corte de carretera de servicio
semaforo p=5;             // Entrada al puente
semaforo_bin servicio=1; //
```

```
void Principal(){
   wait(mutex);
      np++;
   signal(mutex);

   wait(p);

   /*Cruzar Puente*/

  //Sale del puente
   wait(mutex);
      np--;
      if np==0
         signal(cs);
   signal(mutex)

   signal(p);
}
```

```
void Servicio(){
   wait(servicio);
      wait(mutex);
      if np>0 {
         signal(mutex);
         wait(cs);
         }
      else
         signal(mutex);
   signal(servicio)

   wait(p)
   /*Cruzar Puente*/
   signal(p);
}
```

(8) En una peluquería hay 1 silla de barbero y n sillas para que los clientes esperen su turno sentados.

El barbero se echa a dormir si no hay clientes esperando (procedimiento sentarse_y_esperar()). Cuando llega un cliente a la barbería y el barbero está durmiendo lo despierta y se le corta el pelo.

Si llega un cliente mientras el barbero está ocupado cortando el pelo ocupa una silla de espera si hay sillas libres, si no, se van y vuelven más tarde.

Resolver el problema del barbero dormilón mediante:

a) Semáforos
b) Monitores
c) Paso de mensajes.

(8a) Semáforos

```
# define sillas  5      //Número de sillas de espera para clientes
semaphore clientes = 0;    //Número de clientes esperando ser atendidos
semaphore barberos = 0; //Número de peluqueros que están ociosos
semaphore mutex = 1;    //Para garantizar la exclusión mutua
int clientes_esperando = 0;   //Contador de número de clientes esperando
semaphore begincorte=endcorte=0;

void barbero (void) {
   while (true) {
      wait(clientes); //a dormir si no hay clientes
      wait(mutex);
         clientes_esperando--;
         signal(barberos); //ahora barbero listo
      signal(mutex);
      cortar_pelo( );
      }
}

void cliente (void) {
   wait(mutex);
   if (clientes_esperando < sillas) {
         clientes_esperando++;
         signal(clientes); //despierta al barbero
      signal(mutex);
      wait(barberos); //esperar si barbero ocupado
      sentarse_y_esperar( );
      }
   else
      signal(mutex);
}

void cortar_pelo(void) {
        wait(begincorte);
        cortarpelo();
        signal(fincorte);
      }

void sentarse_y_esperar(void) {
        sentarse();
        signal(begincorte);
        wait(fincorte);
      }
```

(8b) Monitores

```
monitor barbería
export entrar,clienteEsperar,
      llamarCliente,
      permitirCorte,
      terminarCorte,
      abrirBarberia;
const sillas=5;

var
   num_clientes:integer;
   clientes_esperando:integer;
   barbero_dormir:condition;
   sentado_sala:condition;
   comanzar_corte:condition;
   levantarse:condition;

function entrar:boolean
begin
if clientes_esperando<sillas
   begin
      clientes_esperando+=1;
      return true;
   end
else
   return false;
end;

procedure clienteEsperar
   resume(barbero_dormir);
   delay(sentado_sala);
end

procedure llamarCliente
begin
   resume(sentado_sala)
   clientes_esperando-=1;
   delay(comenzar_corte)
end
```

```
procedure permitirCorte
begin
   resume(comenzar_corte)
   delay(levantarse)
end

procedure terminarCorte
begin
   resume(levantarse);
   if clients_esperando==0
      delay(barbero_dormir);
end

procedure abrirBarberia
begin
   clientes_esperando=0;
   delay(barbero_dormir);
end;

begin
   num_clientes=N;
   //mantenemos cerrada la
   //barberia con:
   clientes_esperando=sillas;
   cobegin
      Barbero;
      for cont:=1 to num_clientes
         Cliente;
   coend;
end
```

```
process Barbero
begin
   barbería.abrirBarberia;
   repeat
      barbería.llamarCliente;
      ...
     /*Cortar Pelo*/
      ...
      barbería.terminarCorte
   forever
end

process type Cliente
begin
repeat
   if barbería.entrar then
      begin
         barbería.clienteEsperar
         ...
         /*Esperar barbero*/
         ...
         barbería.permitirCorte
         barbería.salir
      end
   else
      pasear;
forever
end
```

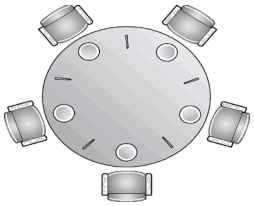
(8c) Paso de mensajes

```
program ejercicio
type
   item=...
   buzon: mailbox of item;
const
   NUMSILLAS=5;
var
   sala_espera:buzon;
   cliente:buzon;

process Barbero
var
   mensaje:item;
begin
   repeat
      /*barbero durmiendo esperando cliente*/
      receive(nuevo_cliente,mensaje);

      prepararse_para_corte();

      /*barbero preparado que pase cliente*/
      send(barbero,mensaje);

      /*espera que el cliente se haya sentado*/
      receive(comienza_corte,mensaje);

      cortar_pelo();

      /*indica al cliente que terminó*/
      send(termina_corte,mensaje);
   forever
end;

begin
   for cont=1 to NUMSILLAS
      send(sala_espera,mensaje);
end
```

```
process type Cliente
var
   mensaje:item;
begin
   /*Si hay sitio en la sala*/
   if (not empty(sitio_en_sala))
      begin
         receive(sitio_en_sala,mensaje);
         /*El cliente cogió sitio
           y despierta al barbero */

         send(nuevo_cliente,mensaje);

         /*espera que el barbero le de paso*/
         receive(barbero,mensaje);

         /*Deja su sitio libre*/
         send(sitio_en_sala,mensaje);

         sentarse_sillon();

         /* indica al barbero que puede empezar*/
         send(comienza_corte,mensaje);

         /*Esperando termine corte*/
         receive(termina_corte,mensaje);
      end
   /*Abandona barberia*/
end;
```

(9) Solucionar el problema de los filósofos mediante:

a) Semáforos
b) Monitores
c) Paso de Mensajes

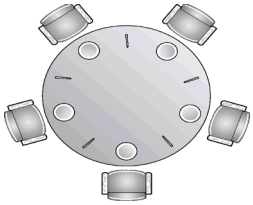


(9a) Filosofos con semáforos

```
#define N     5
#define IZQ   (i+N-1)%N
#define DCH   (i+1)%N
#define PENSANDO      0
#define HAMBRIENTO 1
#define COMIENDO      2
typedef int semaphore;
int estado[N];
semaphore mutex = 1;
semaphore s[N];

void filosofo(int i) {
   while(true){
      pensar();
      coger_tenedores(i);
      comer();
      dejar_tenedores(i);
      }
}
```

```
void coger_tenedores(i){
   wait(mutex);
   estado[i] = HAMBRIENTO;
   test(i);
   signal(mutex);
   wait(s[i]);
}

void dejar_tenedores(i){
   wait(mutex);
   estado[i]= PENSANDO;
   test(IZQ);
   test(DCH);
   signal(mutex);
}

void test(i){
   if (estado[i]==HAMBRIENTO &&
       estado[IZQ]!=COMIENDO &&
       estado[DCH]!=COMIENDO) {
           estado[i]=COMIENDO;
           signal(s[i]);
        }
}
```

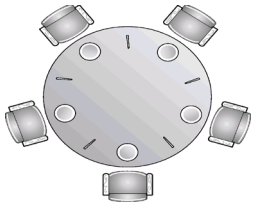
(9b) Filosofos con Monitores



```
monitor Filosofos
   const N=5;
   var estado: array[0..N-1] of (pensando,comiendo,hambriento)
   dormir: array[0..N-1] of contidion
   i:integer
   export coger_palillos, soltar_palillos

   procedure cober_palillos(i:integer)
   begin
      estado[i]:=hambriento;
      test(i);
      if estado[i]<>comiendo then delay(dormir[i])
   end;

   procedure soltar_palillos(i:integer)
   begin
      estado[i]:=pensando;
      test((i+4)mod N);
      test((i+1)mod N);
   end;

   procedure test(k:integer)
   begin
      if  (estado(k) == hambirento) and
          (estado[(k+N-1) mod  N] <> comiendo) and
          (estado[k+1] <> comiendo) then
         begin
            estado[k]:=comiendo;
            resume(dormir[k]);
         end;
   end;

begin
   for i:=0 to N-1 do
      estado[i]:=pensando;
end;
```

```
process type filosofo(i:integer)
begin
   repeat
      piensa;
      Filosofos.goger_palillos(i);
      come;
      Filosofos.soltar_palillos(i);
   forever
end;
```

(9c) Filósofos con Paso de mensajes



```
program Filosofos
type
   item=...
   buzon_filosofo=mailbox of item;
const
   N=5;
var
   pido_palillos: array[0..N-1] of buzon_filosofo;
   palillos_concedidos: array[0..N-1] of buzon_filosofo;
   suelto_palillos:array[0..N-1] of buzon_filosofo;
   palillos: array[0..N-1] of integer;
   cont:integer;
   filosofos: array[0..N-1] of Filosofo;

begin
   for cont:=0 to N-1
      palillos[cont]:=1;
   cobegin
      Controlador;
      for cont:=0 to N-1
         Filosofo(cont);
   coend
end;

process type Filosofo(id:integer)
var
   mensaje: item;
begin
   repeat
      ...
      /*filosofo pensando*/
      ...
      send(pido_palillos[id],mensaje);
      receive(palillos_concedidos[id],mensaje);
      ...
      /*filosofo comiendo*/
      ...
      send(suelto_palillos[id],mensaje);
      ...
   forever
end;

process Controlador;
var
   mensaje:item;
   cont:integer;
begin
   repeat
      ...
      select
         for cont=0 to N replicate
            when palillos[cont]=1 and
               palillos[(cont+1) mod N]=1 then
               receive(pido_palillos[cont], mensaje);
               palillos[cont]=0;
               palillos[(cont+1) mod N]=0;
               send(palillos_concedidos[cont],
                    mensaje);
      or
         for cont=0 to N replicate
            receive(suelto_palillos[cont], mensaje);
            palillos[cont]=1;
            palillos[(cont+1) mod N]=1;
      end
   forever
end;
```

(10) Para transmitir información confidencial se ha decidido cifrar los mensajes, dividirlos en tres partes (caracteres 0, 3, 6... en una; 1, 4, 7... en otra; y 2, 5, 8... en la tercera) y enviar cada parte por un canal de comunicación independiente.

Por lo tanto, para reconstruir un mensaje es necesario leer cada parte del canal correspondiente, reunir la información cifrada en un buffer y descifrarla.

Para llevar a cabo la reconstrucción del mensaje de forma eficiente, se ha decidido estructurar la aplicación utilizando 4 procesos.

Cada uno de los tres primeros se encarga de ir leyendo los caracteres de uno de los canales de comunicación a medida que van llegando (función *leer carácter(canal)*)— y de irlos depositando en el lugar correcto en un buffer de tamaño adecuado para contener el mensaje completo (por simplicidad, supóngase que todos los mensajes son de 3000 caracteres y que el buffer es un vector de 3000 caracteres).

El cuarto proceso espera a que esté el mensaje completo en el buffer y entonces lo descifra. (Naturalmente, los procesos que leen de los canales y que almacenan la información en el buffer deben esperar a que el cuarto proceso acabe de descifrar un mensaje antes de comenzar a almacenar el siguiente en el buffer).

Se pide especificar los algoritmos para los cuatro procesos utilizando semáforos y memoria compartida como herramienta de sincronización. Especificar claramente las variables compartidas por los procesos y su valor inicial, las variables privadas de cada proceso y su valor inicial, y los semáforos empleados y su valor inicial.

```
int num_canales=3;
semaforo sdescifrar=0;
semaforo selector[num_canales];

void lector_canal(i:integer){
   while (no terminar) {
      /* leer lo que nos toca */
      for (x=0; x<1000; ++x) {
         c = leer_caracter(canal i);
         buffer[3*x+i]=c;
         }

   signal(sdescifrar);

   wait(slector[i]);
}
```

```
void descifrador(){
int i;
while (no terminar) {
   /* esperar a que los tres lectores nos
avisen */
   for (i=0;i<num_canales;i++)
      wait(sdescifrar);

   descifrar();
      /* listo, avisar */

   for (i=0;i<num_canales;i++)
      signal(selector[i]);
}

main(){
   parbegin
      descifrador()
      for (i=0;i<num_canales;i++)
         lector_canal(i);
   end
}
```

(11) Una carretera de dos direcciones cruza un río a través de un puente colgante. El puente no puede resistir el peso de dos coches, por lo que solamente se debe permitir que haya un coche cruzando el puente en cada instante.

Utilizando semáforos como herramienta de sincronización, escribir dos algoritmos, uno para los procesos que actúen como coches que llegan a la orilla izquierda y otro para los procesos que simulen coches que lleguen a la orilla derecha. Diseñar los algoritmos de forma que siempre pasen coches de la orilla en la que la cola sea más larga. Si en ambos extremos hay los mismos coches tendrán preferencia los de la izquierda.

```
int nizq = 0, nder = 0;
int ocupado = FALSO;
semaforo sizq=0, sder=0;
semaforo mutex=1;
```

```
void coche_izquierda(){
wait(mutex);
if ( !ocupado ) {
   ocupado = CIERTO;
   signal(mutex);
   }
else {
   nizq++;
   signal(mutex);
   wait(sizq);
   wait(mutex);
   nizq--;
   signal(mutex);
   }

/*Cruzar el puente;*/

wait(mutex);
   if ( nder + nizq == 0 ) {
      ocupado = FALSO;
      }
   else if ( nder > nizq ) {
      signal(sder);
      }
   else {
      signal(sizq);
      }
signal(mutex);
}
```

```
void coche_derecha(){
wait(mutex);
   if ( !ocupado ) {
      ocupado = CIERTO;
      signal(mutex);
      }
   else {
      nder++;
      signal(mutex);
      wait(sder);
      wait(mutex);
      nder--;
      signal(mutex);
   }

/*Cruzar el puente;*/

wait(mutex);
   if ( nder + nizq == 0 ) {
      ocupado = FALSO;
      }
   else if ( nder > nizq ) {
      signal(sder);
      }
   else {
      signal(sizq);
      }
signal(mutex);
}
```

(12 a)(Semáforos) Un proceso, al que vamos a llamar el barquero, se encarga de transportar pasajeros de una parte a otra de un río. El resto de procesos, a los que llamaremos pasajeros, utilizan la barca para cruzar.

Escribir el algoritmo del barquero y los pasajeros utilizando semáforos como herramienta de sincronización para que se cumplan los siguientes requisitos:

- Los pasajeros deben esperar en la orilla izquierda hasta que llegue el barquero.
- Cuando el barquero llegue a la orilla izquierda, debe esperar a que haya al menos un pasajero preparado para subir.
- Cuando hayan subido 10 pasajeros o, si eran menos, todos los que hubiesen esperando, la barca debe iniciar la travesía.
- Al llegar a la orilla derecha el barquero avisará a los pasajeros, que bajarán de la barca.
- Cuando hayan bajado todos los pasajeros, el barquero iniciará el viaje de regreso a la orilla izquierda con nuevos pasajeros.

```
int npas = 0;              /* Num de pasajeros en la orilla*/
int npasbarca = 0;         /* Num de pasajeros en la barca */
semaforo sbarca=0;         /* Para esperar a la barca */
semaforo sbajarbarca=0;    /* Para esperar antes de bajar de la barca */
semaforo sbarq=0;          /* Para que espere el barquero */
semaforo mutex=1;          /* Excl. mutua en el acceso a npas */
```

```
void pasajero(){
   wait(mutex);
      npas++;
   signal(mutex);
   wait(sbarca);
   wait(mutex);
   /*Subir a la barca;*/
   npas--;
   npasbarca++;
   if (npas==0 || npasbarca==10) {
      signal(mutex);
      signal(sbarq);
      }
   else {
      signal(mutex),
      signal(sbarca);
   }

   wait(sbajarbarca);
      /*Bajar de la barca;*/
     wait(mutex);
      npasbarca--;
      if ( npasbarca == 0 )
         signal(sbarq);
      else
         signal(sbajarbarca);
      signal(mutex);
}
```

```
void barquero() {
   while true{
      signal(sbarca);
      wait(sbarq);
      /*Cruzar el rio*/
      signal(sbajarbarca);
      wait(sbarq);
   }
}
```

(12 b) (Monitores con notificación) Un proceso, al que vamos a llamar el barquero, se encarga de transportar pasajeros de una parte a otra de un río. El resto de procesos, a los que llamaremos pasajeros, utilizan la barca para cruzar.

Escribir el algoritmo del barquero y los pasajeros utilizando monitores como herramienta de sincronización para que se cumplan los siguientes requisitos:

- Los pasajeros deben esperar en la orilla izquierda hasta que llegue el barquero.
- Cuando el barquero llegue a la orilla izquierda, debe esperar a que haya al menos un pasajero preparado para subir.
- Cuando hayan subido 10 pasajeros o, si eran menos, todos los que hubiesen esperando, la barca debe iniciar la travesía.
- Al llegar a la orilla derecha el barquero avisará a los pasajeros, que bajarán de la barca.
- Cuando hayan bajado todos los pasajeros, el barquero iniciará el viaje de regreso a la orilla izquierda con nuevos pasajeros.

```
monitor Rio
   var
      numOrilla,numBarca:integer;

      cEsperaPasajeros,
      cColaDeEspera,
      cColaDeBajada,
      cZarpar,
      cBajar  :condition

   export
      PermitirSubir,PermitirBajar,
      EsperaEnCola, Subir, Bajar;

   procedure PermitirSubir()
   begin
      if(numOrilla==0)
           delay(cEsperaPasajeros)
      notify(cColaDeEspera);
      delay(cZarpar);
   end;

   procedure PermitirBajar()
   begin
      broadcast(cColaDeBajada);
      delay(cVolver);
   end;
```

```
   procedure EsperaEnCola()
   begin
      NumOrilla++;
      notify(cEsperaPasajeros)
      delay(cColaDeEspera);
   end;

   procedure Subir()
   begin
      numOrilla--;
      numBarca++;
      if (numBarca<=10 && numOrilla>0)
         notify(cColaDeEspera);
      else
         notify(cZarpar);
      endif
   end;

   procedure Bajar()
   begin
      delay(cColaDeBajada);
      numBarca--;
      if numBarca==0
         notify(cVolver);
   end;

begin
   numOrilla=0;
   numBarca=0;
end;
end monitor
```

```
process Barquero()
begin
   repeat
      Barco.PermitirSubir();
      /*Cruzar Rio*/
      Barco.PermitirBajar();
      /*Volver a por mas*/
   forever
end;
```

```
process type Pasajero()
begin
   Barco.EsperaEnCola();
   Barco.Subir();
   /*Cruzando Rio*/
   Barco.Bajar();
end;
```

(13) Se desea simular mediante procesos el comportamiento de los clientes en un supermercado.

Para ello se ha propuesto el siguiente modelo: El supermercado tiene capacidad para N procesos. Por lo tanto, si al llegar un cliente al supermercado éste está lleno, el cliente deberá esperar a que termine de comprar alguno de los que esté dentro.

Una vez dentro, los clientes van llenando el carro por sí mismos con los productos que necesitan. Sin embargo, por razones higiénicas, existe un charcutero y los clientes deben guardar cola para ser atendidos por él de uno en uno. Una vez ha llegado su turno, el cliente hace el pedido al charcutero y espera a que éste se lo prepare.

Finalmente, el cliente continúa con su compra y el charcutero espera al siguiente cliente.

Cuando el cliente acaba de comprar se dirige a la zona de cajas. El supermercado dispone de 3 cajas y el cliente selecciona una de ellas.

Una vez seleccionada, espera su turno en la cola correspondiente y, cuando le toca el turno, deposita la compra en la caja y espera a que el cajero le diga lo que debe. A continuación paga y espera a que el cajero le dé el recibo.

Finalmente, abandona la caja y sale del supermercado.

Se pide diseñar, utilizando semáforos como herramienta de sincronización, los algoritmos para los clientes, el charcutero y los cajeros, de forma que se coordinen adecuadamente para simular el comportamiento descrito.

```
semaforo ssuper=N;          /* Acceso al supermercado */
semaforo scolachar=1;       /* Acceso a la charcuteria */
semaforo sclichar=0;        /* Espera del cliente en la charcuteria */
semaforo schar=0;           /* Espera del charcutero */
semaforo scolacaja[3]=1;    /* Acceso a cada una de las tres cajas */
semaforo sclicaja[3]=0;     /* Espera del cliente en cada caja */
semaforo scaja[3]=0;        /* Espera del cajero de cada caja */
```

```
void cliente(){
wait(ssuper);
   /*Compra por su cuenta;*/
   wait(scolachar);
      /*Hace el pedido;*/
      signal(schar);
      wait(sclichar);
      /*Recoge el pedido;*/
   signal(scolachar);

   /*Sigue con su compra;*/
   /*Elige una caja (p.ej. la i);*/

   wait(scolacaja[i]);
      /*Deposita la compra en la caja;*/
      signal(scaja[i]);
      wait(sclicaja[i]);
      /*Paga;*/
      signal(scaja[i]);
      wait(sclicaja[i]);
      /*Recoge el recibo y la compra;*/
   signal(scolacaja[i]);
signal(ssuper);
}
```

```
void charcutero(){
while true{
   wait(schar);
      /*Prepara el pedido;*/
   signal(sclichar);
   }
}

void cajero(int i) {
while true{
   wait(scaja[i]);
      /*Calcula la cuenta;*/
   signal(sclicaja[i]);
   wait(scaja[i]);
      /*Cobrar;*/
   signal(sclicaja[i]);
   }
}
```

(14) Una carretera atraviesa un túnel construido a mediados del siglo pasado, de forma que su anchura no permite que se crucen en su interior dos vehículos que circulan en sentido contrario.

Para controlar el tráfico se han dispuesto unas barreras en la ambas entradas al túnel (entrada norte y entrada sur) controladas por un sistema informático.

Para controlar dichas barreras, se proporciona al programador una llamada *Levanta_Barrera(identificador)* que levanta una de las barreras de entrada al túnel, asegurándose además que pasa un único vehículo cada vez que se ejecuta dicha función.

Así pues: *Levanta_Barrera(B NORTE)*, levantará la barrera en el extremo norte del túnel, asegurando además que sólo pasa (en sentido de entrada) un vehículo. *Levanta_Barrera(B SUR)*, levantará la barrera en el extremo sur del túnel, asegurando además que sólo pasa (en sentido de entrada) un vehículo.

Además de las barreras se han colocado unos sensores en las entradas y salidas que detectan cuando llega un nuevo vehículo a una de los extremos y cuando sale uno que acaba de atravesar el túnel.

Cuando se detectan dichas situaciones, los sensores generan uno de los siguientes procesos:

Llega_Vehiculo_Norte cuando llega un nuevo vehículo al extremo Norte del túnel.
Llega_Vehiculo_Sur cuando llega un nuevo vehículo al extremo Sur del túnel.
Sale_Vehiculo_Norte cuando sale un vehículo por el extremo Norte del túnel.
Sale_Vehiculo_Sur cuando sale un vehículo por el extremo Sur del túnel.

Se desea controlar el paso por el túnel de forma que se cumplan los siguientes requisitos:

- Cuando no hay ningún vehículo atravesando el túnel se permite atravesarlo a cualquiera sea cual sea su sentido de circulación.
- Sólo pueden cruzar el túnel varios vehículos de forma simultánea si lo hacen en el mismo sentido.

Sabiendo que el sistema informático proporciona como herramienta de sincronización los monitores, se pide desarrollar el algoritmo de los cuatro procesos antes indicados (*Llega_Vehiculo_Norte*, *Llega_Vehiculo_Sur*, *Sale_Vehiculo_Norte* y *Sale_Vehiculo_Sur*) así como del monitor que utilicen para sincronizarse.

```
monitor tunel{
   int contaNorteSur, contaSurNorte;
   condition pasoNorte, pasoSur;

   function PeticionEntradaNorte {
      (* usada por los vehiculos que intentan *)
      (* atravesar en sentido Norte-Sur *)
      if (contaSurNorte != 0){
         pasoNorte.delay();
         pasoNorte.resume();
         }
      Levanta_Barrera(B_NORTE);
      contaNorteSur++;
      }

   function PeticionEntradaSur {
      (* usada los vehiculos que intentan *)
      (* atravesar en sentido Sur-Norte *)
      if (contaNorteSur != 0){
         pasoSur.delay();
         pasoSur.resume();
      }
      Levanta_Barrera(B_SUR);
      contaSurNorte++;
      }

   function SalidaPorElSur {
      (* usada por los vehiculos que ciculan *)
      (* en sentido Norte-Sur*)
      contaNorteSur--;
      if (contaNorteSur == 0)
          pasoSur.resume();
      }

   function SalidaPorElNorte {
      (* usada por los vehiculos que ciculan *)
      (* en sentido Sur-Norte*)
      contaSurNorte--;
      if (contaSurNorte == 0)
          pasoNorte.resume();
      }

   (* Inicializacion*)
   init{
      contaSurNorte = 0;
      contaNorteSur = 0;
      }
}

#external_monitor tunel
int Llega_Vehiculo_Norte{
   tunel.PeticionEntradaNorte();
}

#external_monitor tunel
int Llega_Vehiculo_Sur{
   tunel.PeticionEntradaSur();
}

#external_monitor tunel
int Sale_Vehiculo_Norte{
   tunel.salePorElSur();
}

#external_monitor tunel
int Sale_Vehiculo_Sur{
   tunel.salePorElNorte();
}
```

(15) Una persona tiene en us casa una jaula llena de canarios en la que hay un plato de alpiste y un columpio. Todos los canarios quieren primero comer del plato y luego columpiarse, sin embargo sólo tres de ellos pueden comer del plato al mismo tiempo y sólo uno de ellos puede columpiarse.

(15a) Desarrollar un monitor de nombre jaula que coordine la actividad de los canarios.

(15b) Desarrollar una solución con semáforos que coordine la actividad de los canarios.

```
#define N 3 /* Número de  puestos en el plato */
monitor jaula /* Definición del monitor */
    condición puesto_plato_disponible, columpio_disponible;
    int contadorP, contadorC;

    void obtener_puesto_en_plato()  /* Procedimiento del monitor */
    {
      if (contadorP == N) wait_mon(puesto_plato_disponible);
      contadorP=contadorP+1;
      comer();
    }

    void dejar_puesto_plato() /* Procedimiento del monitor */
    {
      contadorP = contadorP - 1;
      signal_mon(puesto_plato_disponible);
    }

    void obtener_columpio() /* Procedimiento del monitor */
    {
      if (contadorC == 1) wait_mon(columpio_disponible);
      contadorC=contadorC+1;
      columpiarse();
    }

    void dejar_columpio() /* Procedimiento del monitor */
    {
      contadorC = contadorC - 1;
      signal_mon(columpio_disponible);
    }

    { /* Inicialización del monitor */
      contadorP=0, contadorC=0;
    }
end monitor

void canario() /* Proceso canario */
{
    jaula.obtener_puesto_plato();
    jaula.dejar_puesto_plato();

    jaula.obtener_columpio();
    jaula.dejar_columpio();
}

main() /* Ejecución concurrente */
{
    ejecución_concurrente(canario,...,canario);
}
```