Tema III. Monitores

28. Se consideran dos recursos denominados r1 y r2. Del recurso r1 existen N1 ejemplares y del recurso r2 existen N2 ejemplares. Escribir un monitor que gestione la asignación de los recursos a los procesos de usuario, suponiendo que cada proceso puede pedir:

- ·un ejemplar del recurso r1
- ·un ejemplar del recurso r2
- ·un ejemplar del recurso r1 y otro del recurso r2

La solución deberá satisfacer estas dos condiciones:

Un recurso no será asignado a un proceso que demande un ejemplar de r1 o un ejemplar de r2 hasta que al menos un ejemplar de dicho recurso quede libre

Se dará prioridad a los procesos que demanden un ejemplar de ambos recursos

Tema III. Monitores

```
28.
int n1 = N1, n2 = N2;
condicion r1, r2, ambos;
Pedir_recurso(int num_recurso){
    switch(num_recurso): {
        case 0: if(n1==0 || n2 == 0)
                    ambos.wait;
                n1--;
                n2--;
                break;
        case 1: if(n1==0)
                    r1.wait;
                n1--;
                break;
        case 2: if(n2==0)
                    r2.wait;
                n2--;
    }
}
```

```
Liberar_recurso(int num_recurso) {
    switch(num_recurso) {
        case 1: n1++;
                if(n2>0 &&
    ambos.queue)
                ambos.signal;
                else
                    r1.signal;
                break;
        case 2: n2++;
                if(n1>0 &&
    ambos.queue)
                ambos.signal;
                else
                    r2.signal;
    }
}
```

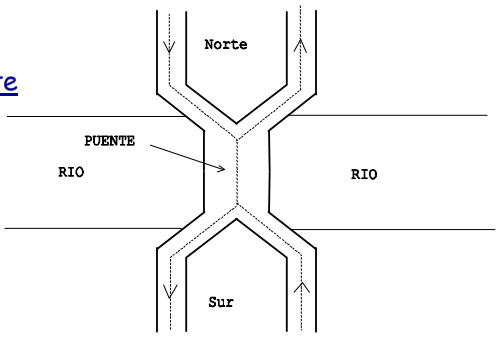
Tema III. Monitores

29. Escribir una solución al problema de "lectores-escritores" con monitores:
    - Con prioridad a los lectores.
    - Con prioridad a los escritores:
    - Con prioridades iguales

Tema III. Monitores

30. Coches que vienen del norte y del sur pretenden cruzar un puente sobre un río. Sólo existe un carril sobre dicho puente. Por lo tanto, en un momento dado, sólo puede ser cruzado por uno o más coches en la misma dirección (pero no en direcciones opuestas).
    a) Completar el código del siguiente monitor que resuelve el problema del acceso al puente suponiendo que llega un coche del norte (sur) y cruza el puente si no hay otro coche del sur (norte) cruzando el puente en ese momento.
    b) Mejorar el monitor anterior, de forma que la dirección del trafico a través del puente cambie cada vez que lo hayan cruzado 10 coches en una dirección, mientras 1 ó más coches estuviesen esperando cruzar el puente en dirección opuesta.

```
MONITOR GestionaTrafico
VAR        ..............
PROCEDIMIENTO EntrarCocheDelNorte
.....................
PROCEDIMIENTO SalirCocheDelNorte
.....................
PROCEDIMIENTO EntrarCocheDelSur
.....................
PROCEDIMIENTO SalirCocheDelSur
Inicialización
.....................
```



Tema III. Monitores

a) MONITOR GestionaTrafico

```
VAR       int N_cruzando,S_cruzando; condition N,S;

PROCEDIMIENTO EntrarCocheDelNorte
if (S_cruzando>0) N.wait; N_cruzando ++;N.signal;}
PROCEDIMIENTO SalirCocheDelNorte{
N_cruzando--; if (N_cruzando==0) S.signal; }
PROCEDIMIENTO EntrarCocheDelSur {
if (N_cruzando>0) S.wait; S_cruzando ++;S.signal;}
PROCEDIMIENTO SalirCocheDelSur{
S_cruzando--; if (S_cruzando==0) N.signal; }

Inicialización
{ N_cruzando=S_cruzando=0; }
```


Tema III. Monitores

b) MONITOR GestionaTrafico

```
VAR       int N_cruzando, S_cruzando, N_delante, S_delante; condition N,S;

PROC  EntrarCocheDelNorte{ if (S_cruzando>0 || N_delante>=10) N.wait();
N_cruzando ++; if (S.queue) N_delante++; if (N_delante<10) N.signal();}

PROC SalirCocheDelNorte{N_cruzando--; if (N_cruzando==0) {S_delante=0;
    S.signal();} }

PROC EntrarCocheDelSur { if (N_cruzando>0 || S_delante>=10) S.wait();
S_cruzando ++; if (N.queue) S_delante++; if (S_delante<10) S.signal();}

PROC SalirCocheDelSur{ S_cruzando--; if (S_cruzando==0) {N_delante=0;
    N.signal();} }

Inicialización
{ N_cruzando=S_cruzando=N_delante=S_delante=0; }
```

Tema III. Monitores

31. Una tribu de antropófagos comparte una olla en la que caben *M* misioneros. Cuando algún salvaje quiere comer, se sirve directamente de la olla, a no ser que ésta esté vacía. Si la olla está vacía, el salvaje despertará al cocinero y esperará a que éste haya rellenado la olla con otros *M* misioneros.

```
Proceso salvaje                 Proceso cocinero
Repetir                          Repetir
   Servirse_1_misionero;           Dormir;
   Comer;                          Rellenar_olla;
 Siempre;                        Siempre;
```

Implementar un monitor para la sincronización requerida, teniendo en cuenta que:

- La solución no debe producir interbloqueo.
- Los salvajes podrán comer siempre que haya comida en la olla.
- Solamente se despertará al cocinero cuando la olla esté vacía.

Tema III. Monitores

Ejercicio 31.

```
Monitor olla
var int num_misioneros; // numero de misioneros en la olla
    condition espera_cocinero, espera_salvaje;
PROC Servirse_1_Misionero()
{if ( num_misioneros == 0) { espera_cocinero.signal;
                             espera_salvaje.wait;}
num_misioneros --; if (num_misioneros > 0 && espera_salvaje.queue)
                                              espera_salvaje.signal;}

PROC Dormir() {if ( num_misioneros > 0 ) espera_cocinero.wait;}

PROC Rellenar_Olla() { num_misioneros = M; if (espera_salvaje.queue)
                                                  espera_salvaje.signal;}
Inicialización
{ num_misioneros=0}
```

Tema III. Monitores

32. Una cuenta de ahorros es compartida por varias personas (procesos). Cada persona puede depositar o retirar fondos de la cuenta. El saldo actual de la cuenta es la suma de todos los depósitos menos la suma de todos los reintegros. El saldo nunca puede ser negativo.

Programar un monitor para resolver el problema.

a) Todo proceso puede retirar fondos mientras la cantidad solicitada *c* sea menor o igual que el saldo disponible en la cuenta en ese momento. Si un proceso intenta retirar una cantidad *c* mayor que el saldo, debe quedar bloqueado hasta que el saldo se incremente lo suficiente (como consecuencia de que otros procesos depositen fondos en la cuenta) para que se pueda atender la petición. El monitor debe tener 2 procedimientos: *depositar(c)* y *retirar(c)*. Suponer que los argumentos de las 2 operaciones son siempre positivos.

b) Modificar la respuesta del apartado anterior, de tal forma que el reintegro de fondos a los clientes sea servido según un orden FIFO. Por ejemplo, suponer que el saldo es 200 unidades y un cliente está esperando un reintegro de 300 unidades. Si llega otro cliente debe esperarse, incluso si quiere retirar 200 unidades. Suponer que existe una función denominada *cantidad(cond)* que devuelve el valor de la cantidad (parámetro *c* de los procedimientos *retirar* y *depositar*) que espera retirar el primer proceso que se bloqueó (tras ejecutar WAIT en la señal **cond**).

Tema III. Monitores

Ejercicio 32 a) con señales no restrictivas

```
Monitor CuentaCorriente;
int saldo; condition cond;

Retirar(int cantidad) {  while (cantidad>saldo) {cond.signal; cond.wait;}
                         saldo-=cantidad;  if (cond.queue()) cond.signal;
                        }

Depositar (int cantidad) { saldo+=cantidad;
                           if (cond.queue()) cond.signal;
                          }
{saldo=… };
```

Tema III. Monitores

Ejercicio 32 a) con señales con prioridad

```
Monitor CuentaCorriente;
int saldo; condition cond;

Retirar(int cantidad) { while (cantidad>saldo) cond.wait (cantidad);
                        saldo-=cantidad;  if (cond.queue()) cond.signal;
                      }

Depositar (int cantidad) { saldo+=cantidad;
                           if (cond.queue()) cond.signal;
                         }
{saldo=… };
```


Tema III. Monitores

Ejercicio 32 b)

```
Monitor CuentaCorriente;
int saldo; condition cond;

Retirar(int cantidad) { if (cantidad>saldo || cond.queue( )) cond.wait;
                        saldo-=cantidad;
                        if ( cantidad(cond) <=saldo && cond.queue( ) ) cond.signal;
                      }

Depositar (int cantidad) { saldo+=cantidad;
                           if (cond.queue() && cantidad(cond) <=saldo )
                              cond.signal;
                         }
{saldo=… };
```

Tema III. Monitores

33. Los procesos P1, P2,...,Pn comparten un único recurso R, pero sólo un proceso puede utilizarlo cada vez. Un proceso Pi puede comenzar a utilizar R si está libre; en caso contrario, el proceso debe esperar a que el recurso sea liberado por otro proceso. Si hay varios procesos esperando a que quede libre R, se concederá al proceso que tenga mayor prioridad. La regla de prioridad de los procesos es la siguiente: el proceso Pi tiene prioridad i (1<=i<=n), donde los números menores implican mayor prioridad. Implementar los procedimientos para "pedir" y "liberar" el recurso.

Tema III. Monitores

Ejercicio 33.

Monitor Recurso

```
    VAR ocupado : boolean;
      recurso : condicion;

    Pedir (int i) {if (ocupado) recurso.wait (i);
                       ocupado = true;  // el recurso está ocupado
                  }
    Liberar () {ocupado = false;    // el recurso está libre
                recurso.signal (); }

{ocupado := false;} ;
```

Tema III. Monitores

34. En un sistema hay dos tipos de procesos: *A* y *B*. Queremos implementar un esquema de sincronización en el que los procesos se sincronizan por bloques de 1 proceso del tipo A y 10 procesos del tipo B. De acuerdo con este esquema:

- Si un proceso de tipo A llama a la operación de sincronización, y no hay (al menos) 10 procesos de tipo B bloqueados en la operación de sincronización, entonces el proceso de tipo A se bloquea.
- Si un proceso de tipo B llama a la operación de sincronización, y no hay (al menos) 1 proceso del tipo A y 9 procesos del tipo B (aparte de él mismo) bloqueados en la operación de sincronización, entonces el proceso de tipo B se bloquea.
- Si un proceso de tipo A llama a la operación de sincronización y hay (al menos) 10 procesos bloqueados en dicha operación, entonces el proceso de tipo A no se bloquea y además deberán desbloquearse exactamente 10 procesos de tipo B.
- Si un proceso de tipo B llama a la operación de sincronización y hay (al menos) 1 proceso de tipo A y 9 procesos de tipo B bloqueados en dicha operación, entonces el proceso de tipo B no se bloquea y además deberán desbloquearse exactamente 1 proceso del tipo A y 9 procesos del tipo B.

No se requiere que los procesos se desbloqueen en orden FIFO.

Tema III. Monitores

Ejercicio 34.

```
Monitor Sincronizacion
Int nA,nB;
Condition condA, CondB;

SincA () {nA++; if (nB<10) then condA.wait;
                  else for i=1 to 10 condB.signal;}
          nA--;}

SincB () {nB++; if (nA<1  ||  nB<10) then condB.wait;
                  else {condA.signal; for i=1 to 9 condB.signal;}
          nB--;}

{nA=nB=0;}
```