Monitors :

A monitor is a high-level construct like an OOP class that centralizes synchronization rules for shared objects, separating synchronization logic from process execution. Processes call the monitors only at synchronization points.

Implementation :

```
Var monitor_name : Monitor
  //declaration of shared variables
   var X, Y : integer
  // declaration of local variables
      i ; j : integer
  // declaration of variables of type condition
   Condition cond1, Cond2
  // Exported procedures
   Procedure entry proce1(...)
    Begin
     ---
    End;
   Procedure entry proce2(---)
    Begin
     ---
    End;
  // initialization of diffirent variables
    Begin
     ----
    End;
```

Rendez-vous de N process : (Réveil en cascade)

```
Rendez-vous_N : monitor
Var
        cpt : integer;
        Attendre_les_autres : condition;
Procedure entry rendez-vous
 Begin
    cpt--;
    if (cpt>0) {
      Attendre_les_autres.wait;
    }
    Attendre_les_autres.signal;
End;
// initialisation :
Begin
   cpt = N;
End;
```

Rendez-vous de N process : (sans réveil en cascade)

```
Rendez-vous_N : monitor
Var  cpt : integer
        waiting : integer
        Attendre_les_autres : condition
Procedure entry rendez-vous
Begin
    cpt--;
    if (cpt>0){
      waiting++;
      Attendre_les_autres.wait;
    } else {
      while (waiting>0){
          waiting--;
          Attendre_les_autres.signal;
      }
   }
End;
```

```
//initialization
Begin
    cpt = N;
    waiting = 0;
End;
```

Producteur / consommateur

```
Prod/cons : monitor
 var Buffer : Array[1..N] of element;
     nPlein : integer;
     in, out : integer;
     attenteProd, attenteCons : condition;

Procedure entry Producteur (resultat : elenet)
Begin
  if (nPlein == N) {
    attenteProd.wait;
  }
  nPlein ++;
  Buffer [in] = resultat;
  in = (in + 1) mod N;
  if (! attenteCons.Empty) {
    attenteCons.signal;
  }
End;

Procedure entry Consommateur (donnee : element)
Begin
    if (nPlein == 0) {
      attenteCons.wait;
    }
    nPlein --;
    donnee = Buffer [out];
    out = (out + 1) Mod N;
    if (! attenteProd.Empty) {
      attenteProd.signal;
    }
End;

//initialisation
Begin
    nPlein = in = out = 0;
End;
```

Lecteurs / Rédacteurs : priorité absolue aux lecteurs :

```
lects_reds : monitor
Var     nl : integer
        E : boolean
        LectCond, redCond : condition

Procedure entry DL
Begin
    if (E == true) {
        lectcond.wait;
    };
    nl++
End;

Procedure entry FL
Begin
    nl--;
    if (nl == 0) {
        redCond.signal;
    }
End;

// initialisation
Begin
    nl = 0;
    E = false;
End;
```

```
Procedure entry DE
Begin
    if (E == true || nl > 0) {
        redcond.wait
    }
    E = true;
End;

Procedure entry FE
Begin
    E = false;
    while (!lectcond.Empty) {
        lectcond.signal;
    }
End;
```

Simulation d'un sémaphore général :

```
semo : monitor
Var   s : integer;
      semcond : condition;

Procedure entry P(s)
Begin
   s--;
   if (s<0) {
    semcond.wait;
   }
End;

Procedure entry V(s)
Begin
   s++;
   if (s<=0) {
      semo-cond.signal;
   }
End;

// initialisation
Begin
   S= N;
End;
```