

Outils et modèles de synchronisation en Ada

Camille Bellissant
IUT 2 de Grenoble

Systemes d'exploitation

Cours, TD, TP

- **Programmation concurrente**
 - Processus
 - Exclusion mutuelle
 - Communication entre processus
- **Gestion de ressources**
 - Ordonnancement
 - Mémoire centrale
 - Mémoire secondaire et fichiers
 - Systemes répartis

Travaux Pratiques sous Unix en C, Ada et Java

- **Modèles de synchronisation**
 - Rendez-vous collectif
 - Allocation de ressources
 - Producteurs/Consommateurs
 - Lecteurs/Rédacteurs
- **Outils de synchronisation**
 - Verrous et sémaphores
 - Compteurs
 - Moniteurs
 - Objets protégés et rendez-vous Ada

Verrous [Dijkstra, 1965]

```
type état is (ouvert, fermé) ;
```

```
protected type verrou is
```

```
  entry franchir ;
```

```
  procedure ouvrir ;
```

```
  private
```

```
    loquet : état := ouvert ;
```

```
end verrou ;
```

```
protected body verrou is
```

```
  entry franchir when loquet = ouvert is
```

```
    begin
```

```
      loquet := fermé ;
```

```
    end franchir ;
```

```
  procedure ouvrir is
```

```
    begin
```

```
      loquet := ouvert ;
```

```
    end ouvrir ;
```

```
end verrou ;
```

```
V : verrou ;
```

```
...
```

```
task body P_i is      -- i = 1..n
```

```
begin
```

```
  ...
```

```
  V.franchir ;
```

```
  -- SECTION CRITIQUE
```

```
  V.ouvrir ;
```

```
  ...
```

```
end P_i ;
```

Sémaphores [Dijkstra, 1968]

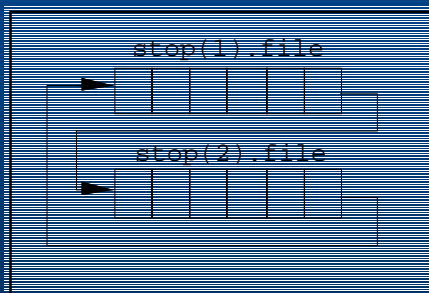
```
protected type sémaphore (initial : natural := 0) is
  entry P ;
  procedure V ;
private
  e : natural := initial ;
end sémaphore ;
```

```
protected body sémaphore is
entry P when i > 0 is
  begin
    e := e - 1 ;
  end P ;
procedure V is
  begin
    e := e + 1 ;
  end V ;
end sémaphore ;
```

© Burns & Wellings

Compteurs [Reed, 1979]

```
protected type compteur(init : integer := 0) is
  entry attendre(valeur : integer) ;
  procedure avancer ;
private
  e : integer := init ;
  file_bloquante : positive := 1 ;
  entry stop(1..2) (valeur : integer) ;
end compteur ;
```



```
protected body compteur is
  entry attendre(valeur : integer) when true is
  begin
    if e < valeur then requeue stop(file_bloquante) ;
  end if ;
  end attendre ;

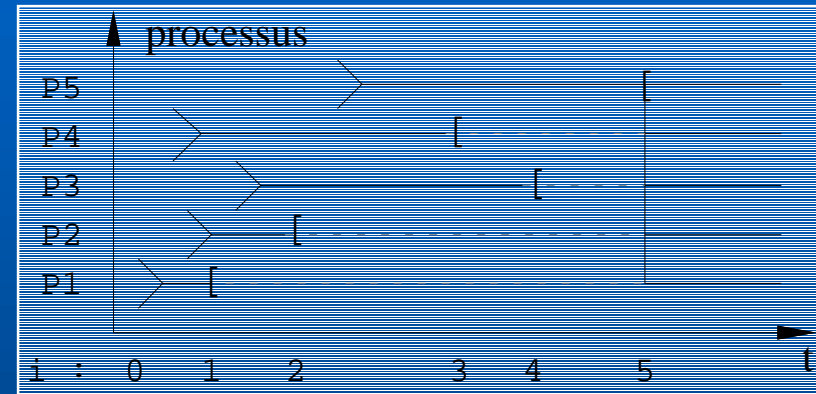
  procedure avancer is
  begin
    e := e + 1 ;
    file_bloquante := 3 - file_bloquante ;
  end avancer ;

  entry stop(for i in 1..2) (valeur : integer)
  when i /= file_bloquante is
  begin
    if e < valeur then requeue stop(file_bloquante) ;
  end if ;
  end stop ;

end compteur ;
```

Modèle du rendez-vous collectif avec un sémaphore

```
nb_présents : sémaphore(0) ;  
i : natural := 0 ; -- compteur  
  
procedure rendez_vous(n : positive) is  
begin  
  i := i + 1 ;  
  if i < n then nb_présents.P ; end if ;  
  i := i - 1 ;  
  if i > 0 then nb_présents.V ; end if ;  
end rendez_vous ;
```

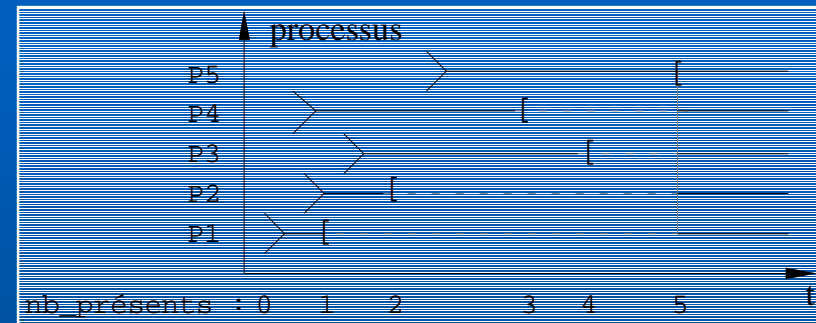


```
n : constant := 5 ;  
...  
task processus(k : 1..n) ;  
task body processus is  
begin  
  ...  
  rendez_vous(n) ;  
  -- attendre le dernier  
  ...  
end processus ;
```

Modèle du rendez-vous collectif avec un compteur

```
nb_présents : compteur(0) ;
```

```
procedure rendez_vous(n : positive) is  
begin  
  avancer(nb_présents) ;  
  attendre(nb_présents, n) ;  
end rendez_vous ;
```



```
n : constant := 5 ;  
...  
task processus(k : 1..n) ;  
task body processus is  
begin  
  ...  
  rendez_vous(n) ;  
  -- attendre le dernier  
  ...  
end processus ;
```

Moniteurs (1) [C.A.R. Hoare, 1974]

```
package moniteurs is
```

```
protected type alerte(init : natural := 0) is
```

```
  entry P ;
```

```
  procedure V ;
```

```
  function file return natural ;
```

```
private
```

```
  i : natural := init ;
```

```
end alerte ;
```

```
mutex : alerte(1) ;
```

```
suspens : alerte(0) ;
```

```
type signal is access all alerte ;
```

```
procedure SEND(s : signal) ;
```

```
procedure WAIT(s : signal) ;
```

```
procedure ENTREE_MONITEUR ;
```

```
procedure SORTIE_MONITEUR ;
```

```
end moniteurs ;
```

```
package body moniteurs is
```

```
protected body alerte is
```

```
-- sémaphore
```

```
  entry P when i > 0 is
```

```
  begin
```

```
    i := i - 1 ;
```

```
  end P ;
```

```
  procedure V is
```

```
  begin
```

```
    i := i + 1 ;
```

```
  end V ;
```

```
  function file return natural is
```

```
  begin
```

```
    return P'count ;
```

```
  end file ;
```

```
end alerte ;
```

Moniteurs 2 [C.A.R. Hoare, 1974]

```
suspens : sémaphore(0) ;
mutex : sémaphore(1) ;

type signal is access all sémaphore ;

procedure ENTREE_MONITEUR is
begin
  mutex.P ;
end ENTREE_MONITEUR ;

procedure SORTIE_MONITEUR is
begin
  if suspens.file > 0 then suspens.V ;
  else mutex.V ; end if ;
end SORTIE_MONITEUR ;
```

```
procedure SEND(s : signal) is
begin
  if s.file > 0 then
    s.V ;
    suspens.P ;
  end if ;
end SEND ;

procedure WAIT(s : signal) is
begin
  if suspens.file > 0 then suspens.V ;
  else mutex.V ; end if ;
  s.P ;
end WAIT ;
```

Modèle producteurs/consommateurs avec un moniteur

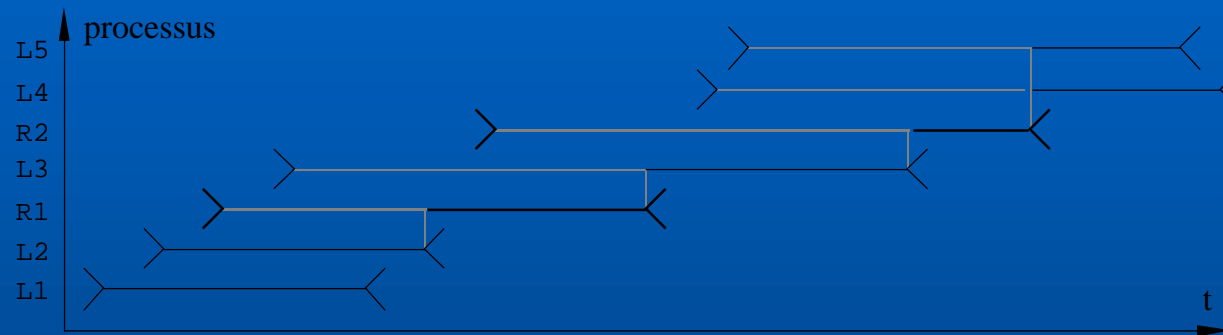
```
n : constant := ... ;
T : array(0..n-1) of message ;
  inpt, outpt : natural range 0..n-1 := 0 ;
  nb_mess : natural range 0..n := 0 ;
  non_plein : signal := new alerte ;
  non_vide : signal := new alerte ;
```

```
procedure déposer(m : message) is
begin
  ENTREE_MONITEUR ;
  if nb_mess = n then WAIT(non_plein) ;
  end if ;
  T(inpt) := m ;
  inpt := (inpt + 1) mod n ;
  nb_mess := nb_mess + 1 ;
  SEND(non_vide) ;
  SORTIE_MONITEUR ;
end déposer ;
```

```
-- taille du tampon
-- n cases dans le tampon
-- pointeurs dans le tampon
-- nombre de messages dans le tampon
-- signifie : nb_mess < n
-- signifie : nb_mess > 0

procedure retirer(m : out message) is
begin
  ENTREE_MONITEUR ;
  if nb_mess = 0 then WAIT(non_vide) ;
  end if ;
  m := tampon(outpt) ;
  outpt := (outpt + 1) mod n ;
  nb_mess := nb_mess - 1 ;
  SEND(non_plein) ;
  SORTIE_MONITEUR ;
end retirer ;
```

Modèle lecteurs/rédacteurs avec un rendez-vous (1)



```
type Accès is (Lecture, Ecriture) ;      -- © Grady Booch
task L_R is
  entry début(quel_accès : in Accès) ; -- une seule entrée pour deux accès
  entry fin_lecture ;
  entry fin_écriture ;
end L_R ;
```

```
task lecteur_i ;
task body lecteur_i is
begin
  LR.début(Lecture) ;
  LIRE ;
  LR.fin_lecture ;
end lecteur_i ;
```

```
task rédacteur_j ;
task body rédacteur_j is
begin
  LR.début(Ecriture) ;
  ECRIRE ;
  LR.fin_écriture ;
end rédacteur_j ;
```

Modèle lecteurs/rédacteurs avec un rendez-vous (2)

```
task body L_R is                                     -- © Grady Booch
nb_lec : natural := 0 ; -- nombre de lectures en cours
begin
  loop
    select
      accept début(quel_accès : in Accès)
      do
        case quel_accès is
          when Lecture => nb_lec := nb_lec + 1 ;
          when Ecriture => while nb_lec > 0
            loop
              accept fin_lecture ;
            end loop ;
        end case ;
      end début ;
      if nb_lec = 0 then accept fin_écriture ; end if ;
    or
      accept fin_lecture ;
      nb_lec := nb_lec - 1 ;
    end select ;
  end loop ;
end L_R ;
```

Modèle lecteurs/rédacteurs avec un objet protégé (1)

```
protected LR is
  entry début(opération : in accès) ;
  procedure fin_lecture ;
  procedure fin_écriture ;
private
  entry début_lecture ;
  entry début_écriture ;
  demande_écriture : boolean := false ;
end LR ;

protected body LR is
  entry début(opération : in accès) when not demande_écriture is
  begin
    if opération = lecture then requeue début_lecture ;
    else
      demande_écriture := true ;
      requeue début_écriture ;
    end if ;
  end début ;
```

Modèle lecteurs/rédacteurs avec un objet protégé (2)

```
entry début_lecture when nb_rédac = 0 is
begin
  nb_lec := nb_lec + 1 ;
end début_lecture ;
```

```
entry début_écriture when nb_lec = 0 and nb_rédac = 0 is
begin
  nb_rédac := 1 ;
end début_écriture ;
```

```
procedure fin_lecture is
begin
  nb_lec := nb_lec - 1 ;
end fin_lecture ;
```

```
procedure fin_écriture is
begin
  nb_rédac := 0 ;
  demande_écriture := false ;
end fin_écriture ;
```

```
end LR ;
```