USTHB

Faculté d'Informatique

Département d'Intelligence Artificielle et des Sciences de Données

SERIE D'EXERCICES N° 2

(Synchronisation par variable d'états et événements)

Exercice 1 : Le programme suivant (algorithme de Dekker) est proposé comme solution au problème de la section critique entre deux processus concurrents :

Initialement : Di := faux ; (i= 1,2) tour := 1 ou 2 ;

```
Processus Pi;
Début
 Répéter
        Di := vrai :
        Tant que  Dj faire Si tour = j Alors
              Début Di := faux ;  Tant que tour = j Faire <rien>Fait ; Di :=vrai   Fin ;
         Fait;
        <section critique i>
         tour : =j ;  Di : = faux ;
Jusqu'à faux ;
Fin.
```

- **Vérifier les conditions de la section critique.**

Initialement : *Di := faux ; (i= 1,2) tour := 1 ou 2 ;*

Processus *Pi;*

Début

Répéter

Di := vrai :

1 ***Tant que*** *Dj*

Faire

Si *tour = j* ***Alors***

Début

Di := faux ;

Retire sa demande

2 ***Tant que*** *tour = j* ***Faire*** *<rien>* ***Fait*** *;*

Di :=vrai

Fin

Fait;

<Section Critique i>

tour : =j ; Di : = faux ;

Jusqu'à *faux ;*

Fin.

Interprétation:

- Seul *Pi* modifie *Di.*
- *Pi* ne teste *Dj* qu' après avoir mis *Di* à *vrai.*
- Si *Pi* est loin de sa SC, Di=faux.
- *Di* ←*vrai* signifie désire d'accès en SC.
- *Di*←*faux* signifie retrait de la demande d'accès en SC .
- La modification de *tour* a lieu seulement à la sortie de la SC.
- Deux points d'attentes possibles pour *Pi*:
  - Au niveau de la boucle 1 :

    (*Di=vrai; tour=i; Dj=vrai*)
  - Au niveau de la boucle 2

    (*Di=faux; tour=j; Dj=vrai*)

Initialement : Di := faux ; (i= 1,2) tour := 1 ou 2 ;

Processus *Pi;*

Début

Répéter

1

Di := vrai :

Tant que *Dj*

Faire

Si *tour = j* ***Alors***

Début

Di := faux ;

Retire sa demande

2 ***Tant que*** *tour = j* ***Faire*** <rien> ***Fait*** ;

Di :=vrai

Fin

Fait;

<Section Critique i>

tour : =j ; Di : = faux ;

Jusqu'à *faux ;*

Fin.

- Exclusion mutuelle:

2 cas possibles:

- Un seul processus désire accéder en SC

Si *Pi* arrive seul, il met *Di* à *vrai*, il trouve *Dj* = *faux*, donc accède en SC quelque soit la valeur de *tour*.

En ce moment, *D=vrai; Dj=faux*; *tour* est quelconque.

Quand *Pj* arrive sachant que *Pi* est toujours en SC, il met *Dj* à *vrai* et attend (car *Di=vrai*):

- dans la boucle1 *si tour = j*

- dans la boucle 2 si *tour = i*, après avoir retiré sa demande (met *Dj* à *faux*).

- Les deux désirent y accéder (*Di=Dj=vrai*)

Si *tour=i* alors *Pi* accède à sa SC après une éventuelle attente temporaire dans la boucle 1 le temps que *Pj* retire sa demande en attendant au niveau de la boucle 2.

De manière générale, celui dont *tour* est égale à son identité accède en SC après une éventuelle attente temporaire dans la boucle 1 le temps que l'autre retire sa demande en attendant au niveau de la boucle 2.

D'autres cas existent, mais se ramènent à l'un des deux cas précédents.

Initialement : *Di := faux ; (i= 1,2) tour := 1 ou 2 ;*

Processus *Pi;*

Début

Répéter

Di := vrai :

1 ***Tant que*** *Dj*

Faire

Si *tour = j* ***Alors***

Début

Di := faux ;

Retire sa demande

2 ***Tant que*** *tour = j* ***Faire*** *<rien>* ***Fait*** *;*

Di :=vrai

Fin

Fait;

<Section Critique i>

tour : =j ; Di : = faux ;

Jusqu'à *faux ;*

Fin.

- Progression

- Si *Pj* est loin de sa SC → *Dj=faux*, *Pi* peut accéder à sa SC sans inconvénients.

Initialement : *Di := faux ; (i= 1,2) tour := 1 ou 2 ;*

Processus *Pi;*

Début

Répéter

1

Di := vrai :

Tant que *Dj*

Faire

Si *tour = j* ***Alors***

Début

Di := faux ;

Retire sa demande

2

Tant que *tour = j* ***Faire*** *<rien>* ***Fait*** *;*

Di :=vrai

Fin

Fait;

<Section Critique i>

tour : =j ; Di : = faux ;

Jusqu'à *faux ;*

Fin.

- Blocage mutuel

-Il est évité d'après le cas 2 de l'EM.

tour permet à un processus d'accéder à sa SC: Il s'agit de celui dont *tour* est égale à son identité.

Initialement : *Di := faux ; (i= 1,2) tour := 1 ou 2 ;*

Processus *Pi;*

Début

Répéter

1 *Di := vrai :*

Tant que *Dj*

Faire

Si *tour = j* ***Alors***

Début

Di := faux ;

Retire sa demande

2 ***Tant que*** *tour = j* ***Faire*** *<rien>* ***Fait*** *;*

Di :=vrai

Fin

Fait;

<Section Critique i>

tour : =j ; Di : = faux ;

Jusqu'à *faux ;*

Fin.

- Attente bornée

2 cas possibles:

- *Pj* est en SC et *Pi* est en attente au niveau 1:

(*Dj=vrai; Di=vrai; tour=i*)

Quand *Pj* sort de sa SC et **postule** une autre fois pour la SC, quelque soit sa vitesse relative d'exécution, c'est ***Pi* qui accédera** la prochaine fois en SC après une éventuelle attente au niveau 1, s'il ne constate pas la première modification de *Dj*, le temps que *Pj* retire sa demande en attendant au niveau 2 (car *Di=vrai* et *tour=i*).

- *Pj* est en SC et *Pi* est en attente au niveau 2:

(*Dj=vrai; Di=faux; tour=j*)

Quand *Pj* sort de sa SC, il met tour à i et postule une autre fois pour la SC. **Il pourra accéder un certain nombre de fois** en SC avant *Pi* si *Pi* ne progresse pas (car *Di=faux*). **Dès que *Pi* progresse** (*tour*=i), il met *Di* à *vrai*, la prochaine fois **c'est lui qui accédera** en SC quelque soit la vitesse relative d'exécution de *Pj*, après une éventuelle attente au niveau 1, s'il ne constate pas la première modification de *Dj*, le temps que *Pj* retire sa demande en attendant au niveau 2 (car *Di=vrai* et *tour=i*).

Exercice 2 : Le programme suivant (algorithme de Lamport) est proposé comme solution au problème de section critique entre deux processus concurrents ;

Choix : tableau [0.. 1] de booleen :=faux;
Numero : tableau [0.. 1] de entier :=0 ;

Notation : $(a, b) < (c, d) \Leftrightarrow (a < c)$ ou $(a = c$ et $b < d)$

Processus Pi ;
Début
Répéter
$Choix_i$= vrai ; $Numero_i$= max ($Numero_i$, $Numero_j$) +1 ; $Choix_i$= faux ;
Tant que $Choix_j$ faire rien fait ;
Tant que ($Numero_j \neq 0$) et (($Numero_j$, j) < ($Numero_i$, i)) faire <rien> fait;
<SC_i >
$Numero_i$ = 0 ;
<Section $restante_i$ > ;
Jusqu'à faux ;
Fin ;

a/ Vérifier les conditions de la section critique.
b/ Discuter le problème lié à la valeur numéro qui peut croître indéfiniment.
c/ Réécrire la solution pour *n* processus.

a/ Vérification des conditions de la SC

Choix : tableau [0.. 1] de booleen :=faux;
Numero : tableau [0.. 1] de entier :=0 ;

Processus *Pi ;*
Début
Répéter
$Choix_i$= *vrai ;* $Numero_i$= *max* ($Numero_i$, $Numero_j$) +1 ; $Choix_i$= *faux ;*
1 ***Tant que*** $Choix_j$ ***Faire*** *rien* ***Fait*** ;
2 ***Tant que*** ($Numero_j \neq 0$) *et* (($Numero_j$, j) < ($Numero_i$, i)) ***Faire*** <rien> ***Fait;***
<SC_i>
$Numero_i$ = 0 ;
<*Section restante*$_i$ > ;
Jusqu'à *faux*
Fin ;

Interprétation:

- La première boucle empêche *Pi* de tester la condition d'accès en SC pendant que *Pj* est en train de modifier son numéro.
- $Numéro_i$=0 signifie que *Pi* est loin de sa SC.
- La condition d'accès en SC est : ***[($Numero_i \neq 0$) et (($Numero_i$, j) < ($Numero_i$, i))]***

Choix : tableau [0.. 1] de booleen :=faux;
Numero : tableau [0.. 1] de entier :=0 ;

Processus *Pi ;*
Début
Répéter
$Choix_i$= *vrai ;* $Numero_i$= *max* ($Numero_i$, $Numero_j$) +1 ; $Choix_i$= *faux ;*
1 ***Tant que*** $Choix_j$ ***Faire*** *rien* ***Fait*** ;
2 ***Tant que*** ($Numero_j = 0$) *et* (($Numero_j$, *j*) < ($Numero_i$, *i*)) ***Faire*** <*rien*> ***Fait***;
<SC_i>
$Numero_i = 0$;
<$Section\ restante_i$> ;
Jusqu'à *faux*
Fin ;

- La première boucle **empêche** *Pi* de tester la condition d'accès en SC pendant que *Pj* est en train de modifier son numéro. Elle sert à **empêcher l'accès simultané** des deux processus en SC qui se produit comme suit:
Pi accède à la SC **grâce à** $Numéro_j=0$ et *Pj* accède en SC **grâce à sa propre identité** ($j<i$) avec $Numéro_i=Numéro_j$.

Exclusion mutuelle : Deux cas existent:
Cas 1 : Si *Pi* arrive **seul**, il met num_i à *1* et entre en SC_i en franchissant les deux boucles car $choix_j=faux$ et $num_j=0$. Quand *Pj* arrive, il met num_j à *2*, franchi la boucle 1 car $choix_i=faux$ et attend au niveau de la boucle 2 car $[(num_i<>0) et ((num_i,i)<(num_j,j))]$.

Cas 2 : Si les *Pi* et *Pj* arrivent en **même temps**, deux cas sont possibles :
a) S'ils **mettent à jour** num_i et num_j **en même temps** donc ($num_i=num_j=1$). Dans ce cas, le processus dont **l'identité est la plus petite accède** à la SC et l'autre se bloque au niveau de la boucle 2.

b) S'ils **mettent à jour séquentiellement** leurs num_i et num_j. Dans ce cas, celui qui a la plus petite valeur de *num* accède à la section critique et l'autre attend au niveau de la boucle 2.

```
Choix : tableau [0.. 1 ] de booleen :=faux;
Numero : tableau [0.. 1 ] de entier :=0 ;

Processus Pi ;
Début
Répéter
Choix_i= vrai ;  Numero_i= max (Numero_i , Numero_j) +1 ; Choix_i= faux ;
1  Tant que Choix_j Faire rien Fait ;
2  Tant que ( Numero_j ≠ 0) et ((Numero_j , j) < (Numero_i , i ) ) Faire <rien> Fait;
   <SC_i>
   Numero_i = 0 ;
<Section restante_i > ;
Jusqu'à faux
Fin ;
```

Progression: Elle est assurée car si un processus *Pi* est loin de la section critique, l'autre peut y accéder car $num_i=0$ (voir la première partie du cas 1 de l'exclusion mutuelle).

Blocage mutuel: Il est évité d'après le cas 2 de l'E.M.

Attente bornée : Elle est assurée.

Supposons que *Pj* est en SC_j , *Pi* en attente et $num_i = x$. Quand *Pj* sort de sa SC_i il met num_j à *0*, si *Pi* constate ce changement il accède en SC_i . Dans le cas contraire, si *Pj* est plus rapide en calculant num_j, sa valeur sera *(x+1)* donc $(num_i, i)<(num_i,j)$. Ceci mène *Pj* à se bloquer au niveau de la boucle 2 et *Pi* accède à la SC_i.

b/ Augmentation des valeurs de num_i et num_j

Choix : tableau [0.. 1] de booleen :=faux;
Numero : tableau [0.. 1] de entier :=0 ;

Processus *Pi ;*
Début
Répéter
$Choix_i$= vrai ; $Numero_i$= max ($Numero_i$, $Numero_j$) +1 ; $Choix_i$= faux ;
1 ***Tant que*** *$Choix_j$* ***Faire*** *rien* ***Fait*** *;*
2 ***Tant que*** *($Numero_j \neq 0$) et (($Numero_j$, j) < ($Numero_i$, i))* ***Faire*** *<rien>* ***Fait****;*
<SC_i>
$Numero_i$ = 0 ;
<Section $restante_i$ > ;
Jusqu'à *faux*
Fin *;*

Cette augmentation se produit quand à chaque fois qu'un processus sort de sa SC, il postule une autre fois pour la SC alors que l'autre est en attente de sa SC.

Dans ce cas, il faut réinitialiser les deux valeurs d'une manière à garder **l'ordre entre elles** et **sans conséquences sur le protocole.**

c/ Généralisation à n processus.

```
Const n=….;
Var Numero : tableau [0.. n-1] de entier :=0 ;
    Choix : tableau [0.. n-1 ] de booleen :=faux;
     j: entier;
Processus Pi (i),  i=0, …n-1
Début
Répéter
Choix_i= vrai ;
Numero_i= max (Numero_j , j∈[0..n-1] ) +1 ;
Choix_i= faux ;
j:=0;
Répéter
Tant que Choix_j Faire rien  Fait ;
Tant que ( Numero_i = 0) et ((Numero_j, j) < (Numero_i, i) )
     Faire <rien>  Fait;
j:=j+1;
Jusqu'à (j=n);
  <SC_i>
 Numero_i = 0 ;
<Section restante_i > ;
Jusqu'à faux
Fin ;
```

Idées de base:

- On garde au mieux identique la forme générale d'un processus.

- Chaque processus doit avoir un **numéro calculé par généralisation.**

- Le processus de plus **petit numéro** (<> 0) ou **de plus petite identité** dans le cas d'égalité de numéros **accède en SC** parmi ceux qui désirent accéder à la SC.

- Un processus doit pouvoir **accéder en SC**, si les autres sont **loin** de leurs SCs.

- Pendant qu'un processus *Pj* **modifie son numéro, aucun autre ne doit tester le numéro de *Pj*.**

→ Chaque processus doit avoir une variable ***choix***.

Exercice 3 :

On propose la solution suivante comme protocole d'exclusion mutuelle pour deux processus Pi et Pj.

Initialement Di= Dj= faux ; tour= i ou j ;

Processus Pi;
Début
Répéter
Di := vrai;
Tant que tour = j
Faire
Tant que Dj = vrai **Faire** <rien>**Fait** ;
tour :=i
Fait ;
<Section Critique i>
Di : = faux ;
Jusqu'à faux
Fin.

1/ Donner la condition d'entrée en SC.
2/ Vérifier si cette solution peut être retenue comme protocole d'exclusion mutuelle.

3/ On modifie la solution comme suit :
Répéter
Di := vrai :
Tant que (tour = j) ou (Dj=vrai)
Faire
Tant que Dj = vrai **Faire** <rien>**Fait** ; tour :=i
Fait ;
<Section Critique i>
Di : = faux ;
Jusqu'à faux
a- Donner la condition d'entrée en SC.
b- Vérifier les 4 conditions de la SC.

Initialement *Di= Dj= faux ; tour= i ou j ;*

Processus *Pi;*
Début
Répéter
Di : = vrai;
1 ***Tant que*** *tour = j*
Faire
2 ***Tant que*** *Dj = vrai* ***Faire*** *<rien>* ***Fait*** *;*
tour : =i
Fait *;*
<Section Critique i>
Di : = faux ;
Jusqu'à *faux*
Fin.

Interprétation

- Si *Pi* est en SC, *Di=vrai*.
- Si *Pi* est loin de la SC, *Di=faux*.
- La boucle 2 est le point d'attente d'entrée en SC.

1- Condition d'accès en SC: *(tour <> j)* [*et Dj=faux*]

2- Eligibilité de la solution:

Ce protocole ne peut pas être utilisé comme solution au problème d'EM car l'exclusion mutuelle n'est pas assurée.

Supposons *tour=j* et *Pi* arrive seul. Il accède à la boucle externe pour exécuter la boucle interne. Puisque *Dj=faux*, il sort de cette boucle, admettons qu'il s'arrête à ce niveau et *Pj* arrive. *Pj* entre directement en SC car *tour<>i*. Ensuite *Pi* progresse en mettant *tour* à *i* et entre aussi en SC.

3/ Nouvelle solution

Répéter

Di := vrai :

1 ***Tant que*** *(tour = j)* ***ou (Dj=vrai)***

Faire

2 ***Tant que*** *Dj = vrai* ***Faire*** *<rien>* ***Fait*** *; tour :=i*

Fait *;*

<Section Critique i>

Di : = faux ;

Jusqu'à *faux*

a- Donner la condition d'entrée en SC.

b- Vérifier les 4 conditions de la SC.

Interprétation

- Si *Pi* est en SC, *Di= vrai*.
- Si *Pi* est loin de sa SC, *Di= faux*.
- La boucle 2 est le point d'attente d'entrée en SC.

a/ Condition d'accès en SC

(tour<>j) et (Dj=faux)

b- Vérification des conditions de la SC

Exclusion mutuelle: 2 cas possibles:

Cas 1: Si *Pi* arrive seul, il met *Di* à *vrai*, ensuite entre en SC directement si *tour=i* ou après l'avoir modifié à i dans le cas contraire car (*Dj=faux*). En ce moment : *(tour=i, Di=vrai, Dj=faux)*. Quand *Pj* arrive, sachant que *Pi* est toujours en SC, ***Pj* attend au niveau de la boucle interne** car au moins *Di=vrai*.

Cas2: Si *Pi* et *Pj* arrivent en même temps (*Di=Dj=vrai*), **aucun des deux** processus ne peut entrer en SC quelque soient leurs vitesses relatives d'exécution. Ils se bloquent au niveau de la boucle interne.

→ l'EM est assurée.

3/ Nouvelle solution

Répéter

Di := vrai :

Tant que *(tour = j) ou (Dj=vrai)*

Faire

Tant que *Dj = vrai* ***Faire*** *<rien>* ***Fait*** *; tour :=i*

Fait *;*

<Section Critique i>

Di : = faux ;

Jusqu'à *faux*

Progression: Assurée.

Si *Pj* est loin de sa SC (*Dj=faux*). Quand *Pi* arrive, il met *Di* à *vrai* ensuite entre en SC directement si *tour<>i* ou après l'avoir modifié à *i* dans le cas contraire car (*Dj=faux*).

Blocage mutuel: Non évité.

Voir cas 2 de l'EM.

Attente bornée:

Elle **n'est pas assurée** dans le cas suivant : Si *Pj* est en SC (*Dj=vrai*) et *Pi* en attente dans la boucle interne (*Dj=vrai*). Quelque soit la valeur de *tour*, quand *Pj* sort de sa SC et postule une autre fois, si *Pi* ne constate pas la modification de *Dj*, les deux processus se bloquent mutuellement et donc *Pi* est privé de la SC.

Exercice 4 On suppose un système constitué uniquement de deux événements mémorisés e1, e2.
Ecrire une implémentation à l'aide des sémaphores des deux primitives suivantes :
Attendre (e1 et e2) qui permet d'attendre que les événements e1 et e2 se produisent.
Déclencher (ei) qui permet de réveiller tous les processus en attente de cet événement si leurs conditions sont satisfaites et l'événement est acquitté. Si aucun processus ne l'attend, l'événement est mémorisé.

- Représentation des états des deux événements:
  → *e:tableau [1..2] de entier :=0 ;*
- Nécessité d'attendre si le prédicat n'est pas satisfait
  → *S: sémaphore:=0;*
- Plusieurs processus peuvent être en attente
  → *cpt: entier:=0;*
- Protection des variables de synchronisation
  → *mutex : sémaphore:=1 ;*

Structure de données communes

e:Tableau [1..2] de entier :=0 ;
S : sémaphore :=0 ;
mutex : sémaphore :=0 ;
cpt :entier :=0 ;

Les primitives

```
Primitive Attendre (e1 et e2) ;
Début
P(mutex);
Si((e1<>1) ou (e2<>1)) Alors
  cpt :=cpt+1 ;
  V(mutex) ;
   P(S) ;
Sinon
   e1:=0; e2:=0;
   V(mutex)
Fsi
Fin ;
```

```
Primitive Déclencher (e[j]) ;j=1,2.
Début
  P(mutex) ;
  Si e[j]<>1 alors
     e[j]:=1;
     Si (((j=1) et (e[2]=1)) ou ((j=2)et (e[1]=1))) Alors
          Si (cpt)<>0) Alors
               Tantque (cpt<>0) Faire
                    V(S) ; cpt :=cpt-1 ;
               Fait ;
               e[1] :=0 ; e[2] :=0 ;
          Fsi
       Fsi
   Fsi;
V(mutex)
Fin ;
```

Exercice 5 On s'intéresse aux événements mémorisés. On définit les primitives suivantes :

Wait(k,e1,…en) : permet d'attendre k événements sur n déclenchés par des processus quelconques.

Signal(e) : déclenche l'arrivée de l'événement *e*. Tous les processus en attente de cet événement sont activés si leurs conditions sont satisfaites et l'événement est acquitté. Si aucun processus ne l'attend ou les conditions des processus en attente de l'événement ne sont pas satisfaites, l'événement est mémorisé.

1. Utiliser cet outil pour programmer l'application suivante dans deux cas différents :

Soient 3 processus ayant un point de rendez-vous. Un processus arrivant au point de rendez-vous continue son exécution :

a/ si les 2 autres sont arrivés à ce point.

b/ si au moins 1 autre est arrivé.

2- Proposer une implémentation de ces primitives.

1/a Rendez-vous si les deux autres arrivent:

e1, e2, e3 : événement:=0

-Première tentative

- Déroulement

→ **Solution incorrecte.**

e1=1;
e2=1;
e3=1;
e1=0; e2=0; e3=0.

Processus *P1;*
-
Signal (e1);
Wait (2, e2, e3);
-

Processus *P2;*
-
Signal (e2);
Wait (2, e1, e3);
-

Processus *P3;*
-
Signal (e3);
Wait (2, e1, e2);
-

Blocage →

On doit signaler l'arrivée par un événement différent pour chaque processus → 6 événements.

Exercice 5

--

1- Utiliser cet outil pour programmer l'application suivante dans deux cas différents :
Soient 3 processus ayant un point de rendez-vous. Un processus arrivant au point de rendez-vous continue son exécution :

a/ si les 2 autres sont arrivés à ce point.

b/ si au moins 1 autre est arrivé.

2- Proposer une implémentation de ces primitives

1/a Rendez-vous si les deux autres arrivent:

- Deuxième tentative
  Déroulement

```
e12=1;
e13=1;
e21=1;
e23=1;
e31=1;
e21=0;
e31=0;
e32=1;
e12=0;
e32=0;
e13=0;
e23=0;
```

e12, e13, e21,e23, e31, e32: événement:=0

***Processus** P1;*
-
Signal (e12);
Signal (e13);
Wait (2, e21, e31);
-

***Processus** P2;*
-
Signal (e21);
Signal (e23);
Wait (2, e12, e32);
-

***Processus** P3;*
-
Signal (e31);
Signal (e32);
Wait (2, e13, e23);
-

1/a Rendez-vous si les deux autres arrivent:

e12, e13, e21,e23, e31, e32: événement:=0

- Deuxième tentative
  Déroulement
  → **Solution correcte**

***Processus** P1;*
-
Signal (e12);
Signal (e13);
Wait (2, e21, e31);
-

***Processus** P2;*
-
Signal (e21);
Signal (e23);
Wait (2, e12, e32);
-

***Processus** P3;*
-
Signal (e31);
Signal (e32);
Wait (2, e13, e23);
-

1/b Rendez-vous au moins un des deux arrive:

Il suffit de mettre le premier paramètre de wait () **à 1** au lieu de 2.

Exercice 5

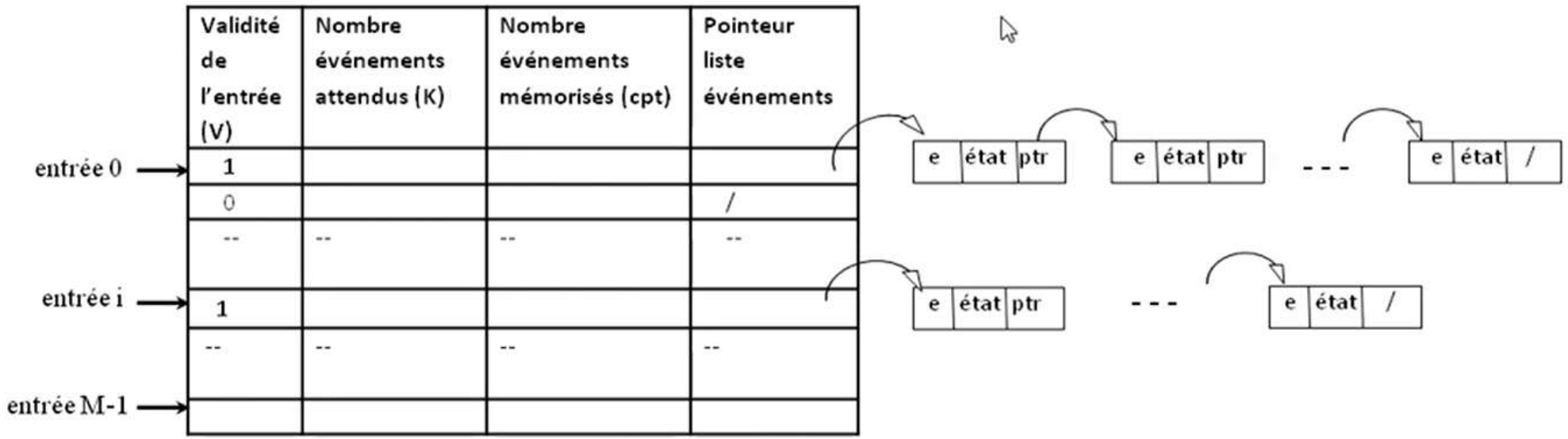
--

2- Proposer une implémentation de ces primitives.

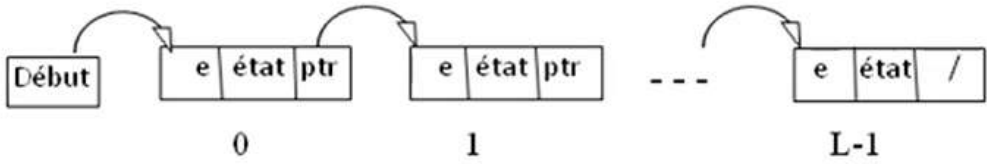
On suppose une application constituée d'un ensemble de *M* processus *{P0, P1,..PM-1}* qui se **synchronisent** à l'aide d'un ensemble de *L* **événements** *{e1,e2...eL}*. Chaque processus utilise un sous-ensemble de ces événements.

Structures de données communes

- La table des processus (TP)



- La liste de tous les événements avec leurs états (LE)



- Sémaphores

S: tableau [0..M-1] de sémaphore:=0;
mutex: sémaphore:=1;

Exercice 5

--

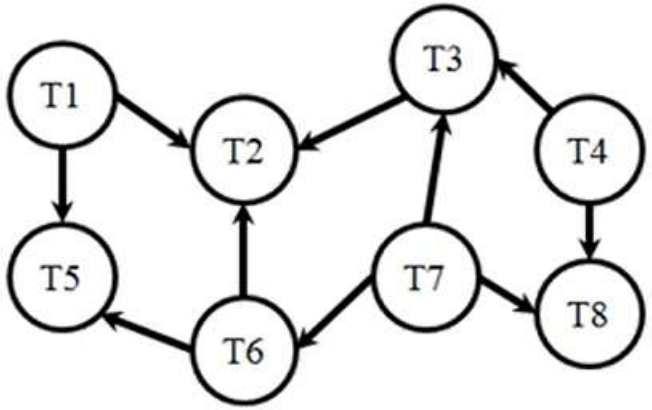
2- Proposer une implémentation de ces primitives

2- Implémentation des deux primitives

```
Primitive Wait (k, e1,…en);
cpt: entier:=0;
Début
P(mutex);
Pour chaque événement e de la liste e1,.. en Faire
 Cherche e dans la liste d'événements LE: soit LE.e ;
 Si (LE.e.etat=1) Alors
    cpt:=cpt+1; <garder trace de LE.e dans une liste temporaire>
    Si (cpt=k) Alors break Fsi
 Fsi;
Fait;
Si (cpt=k) Alors
       <Acquitter les événements appartenant à la liste temporaire
              dans LE>;
        V(mutex);
Sinon
       <Insérer dans la table de processus (i, 1, cpt, {e1, ...en}
                                  avec leurs états actuels>
// i est l'identité du processus qui exécute cette primitive
V(mutex);
P(S[i])
Fsi
Fin;
```

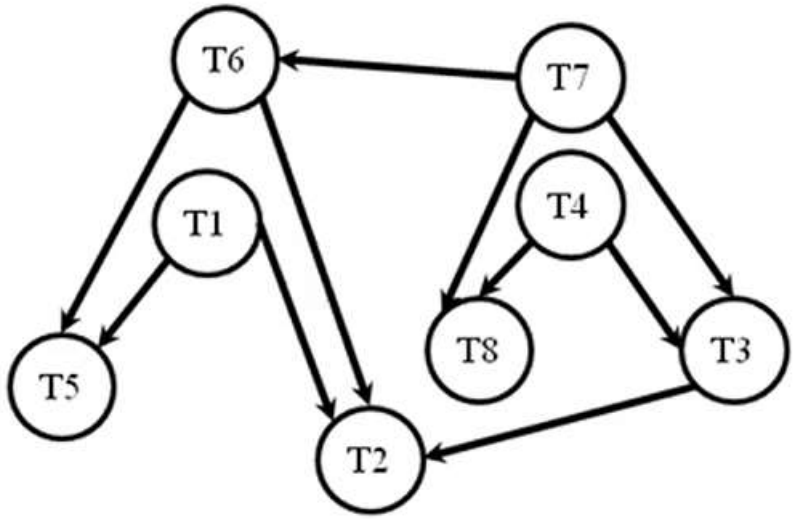
```
Primitive Signal(e);
j: entier;
Début
P(mutex);
 Chercher e dans la liste des événements LE:
       Soit LE.e cet événement:
Si (LE.e.etat =0) Alors
  LE.e.etat:=1;
 Pour j:=0 à M-1 Faire
      Si (TP[j].V=1) Alors
        //processus j bloqué.
          Si (e ∈ à la liste attendue par le processus j):
                Soit TP[j].e cet événement Alors
               Si (TP[j].e.etat=0) Alors
                    TP[j].e.etat:=1;
                    TP[j].cpt:= TP[j].cpt+1;
                     Si (TP[j].cpt=TP[j].K) Alors
                          <Mettre j dans une liste temporaire>
                     Fsi
                Fsi
            Fsi
     Fsi
Fait;
Pour tout j ∈ à la liste temporaire créée: Faire
   <Acquitter chaque événement de TP[j] telle que
          TP[j].e.etat=1 dans la liste des événements LE >;
   < Libérer l'entrée i dans la table des processus>
V(S[j];
Fait
Fsi
V(mutex)
Fin;
```

Exercice 6: Soit le graphe suivant :



1/ Est-il proprement lié ?

2/ Exprimer le graphe à l'aide de Parbegin/Parend et éventuellement avec les sémaphores.



1/ Type du graphe:

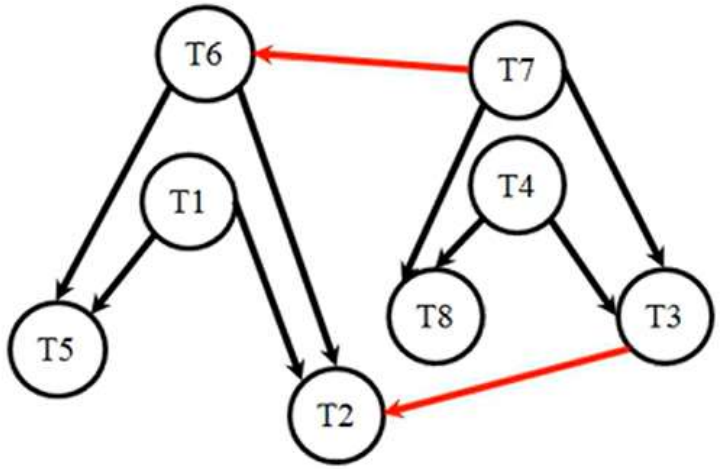
Le graphe n'est pas décomposable dès de départ

→ Il n'est pas proprement lié.

Exercice 6:

--

2/ Exprimer le graphe à l'aide de *Parbegin/Parend* et éventuellement avec les sémaphores.



Etapes

- Rendre le graphe proprement lié.
  On supprime les arcs *(T7, T6)* et *(T3, T2)*.
- Expression du graphe obtenu à l'aide de *Parbegin/ Parend*.
- Reprise des arcs supprimés et ajout des sémaphores.
- Introduire *P()* et *V()* aux bons endroits.
  Chaque arc nécessite un sémaphore privé.

```
Programme P ;
Début
S76, S32: Sémaphore:=0;
Parbegin
  Début
    Parbegin
      Début
        P(S76) ;
        T6
      Fin ;
      T1
    Parend ;
    Parbegin
      T5 ;
      Début
        P(S32) ;
        T2
      Fin
    Parend ;
  Fin ;
  Début
    Parbegin
      Début
        T7 ;
        V(S76)
      Fin ;
      T4
    Parend ;
    Parbegin
      T8;
      Fin
        T3;
        V(S32)
      Fin
    Parend
  Fin
Parend
End.
```