Exercice 1 : Soit un système composé de trois processus fumeurs et d'un processus agent.
Chaque fumeur roule continuellement une cigarette et la fume. Pour pouvoir rouler une cigarette, trois ingrédients sont nécessaires : le tabac, le papier et les allumettes.
Un des processus fumeurs détient le papier, le deuxième processus le tabac et le troisième les allumettes.
Le processus agent a un approvisionnement infini des trois ingrédients. L'agent place deux ingrédients distincts sur la table ainsi le fumeur qui possède le troisième ingrédient peut rouler une cigarette, la fumer puis signale à l'agent la fin de son opération. L'agent remet sur la table deux autres ingrédients et ainsi le cycle se répète.
A- Synchroniser les processus à l'aide des sémaphores.
B- Modifier la solution pour permettre à plus d'un fumeur d'opérer en même temps.
C/ Généraliser la solution à plusieurs fumeurs.

```
S: tableau [1..3] de semaphore:=0;
ag: semaphore:=0;
```

```
Processus Agent ();
j: entier;
Debut
Repeter
j:=Choisir();
<Déposer ingrédients j>;
V(S[j]);
P(ag);
Jusqu'à faux
Fin.
```

```
Processus Fumeur (i: [1..3]);
Debut
Repeter
P(S[i]);
<Prendre ingrédients>;
< rouler et fumer cigarette>;
V(ag)
Jusqu'à faux
Fin.
```

Idées de base:

- *Trois* processus fumeurs et *un* processus agent.
- Les processus *sont* tous cycliques.
- Pas de relation entre les processus *fumeurs*.
- Quand l'*agent* dépose deux ingrédients, un seul *fumeur* est concerné; c'est celui qui dispose du troisième ingrédient.

Raisonnement:

Codifions les processus : *1*: détient Papier (P), *2*: détient Tabac (T); *3*: détient Allumettes (A).

- Soit une fonction *Choisir()* qui retourne une valeur aléatoire *j* entre *1* et *3* correspondant au processus *fumeur j*.
- *j* indique, par correspondance, le code des ingrédients déposés par l'agent: *1: T+A; 2: A+P; 3: P+T.*
- Chaque *fumeur* ne peut opérer que si l'*agent* dépose les ingrédients le concernant → ***S: tableau de [1..3] de semaphore:=0;***
- L'*agent* doit attendre la fin de l'opération du *fumeur* en cours → sémaphore ***ag:=0***.

B- Modifier la solution pour permettre à plus d'un fumeur d'opérer en même temps.
C/ Généraliser la solution à plusieurs fumeurs.

```
S: tableau [1..3] de semaphore:=0;
ag: semaphore:=0;
```

```
Processus Agent();
j: entier;
Debut
Repeter
j:=Choisir();
<Déposer ingrédients j>;
V(S[j]);
P(ag);
Jusqu'à faux
Fin.
```

```
Processus Fumeur (i: [1..3]);
Debut
Repeter
P(S[i]);
<Prendre ingrédients>;
< rouler et fumer cigarette>;
V(ag)
Jusqu'à
Fin.
```

B/ - Si *ag=1* Alors *deux* fumeurs au maximum opèrent simultanément.
- Si *ag=2* Alors *trois* fumeurs au maximum opèrent simultanément.

C/ Généralisation

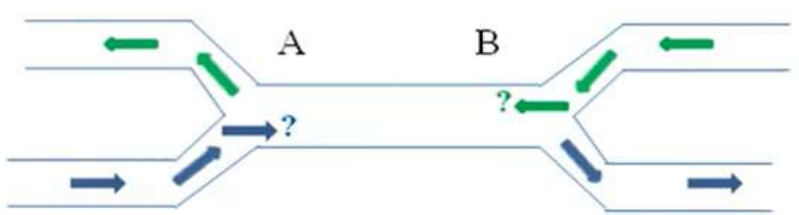
- Nombre de processus fumeurs >3.
  - Répartir les fumeurs en 3 classes
    - → La solution reste valable.
    - → Tour de rôle au sein de chaque classe.
- Permettre à *N* fumeurs d'opérer simultanément
  - → *ag=N-1.*
- Généraliser à un problème à *M* éléments
  - → *M* classes au lieu de 3 actuelles.
  - → *S: tableau[1..M] de semaphore;*

Exercice 2 : Des voitures venant du nord et du sud doivent traverser un pont. Sur ce pont ne peut passer qu'une seule file de voitures à la fois et dans un même sens.

A- Ecrire un algorithme qui permet aux voitures de passer du nord vers le sud ou du sud vers le nord en se synchronisant à l'aide des sémaphores au niveau du pont.

B- Discutez votre solution et proposez d'éventuelles modifications pour diverses situations.

```
Processus SensAB ();
Debut
P(mutexAB);
nAB:=nAB+1;
Si nAB=1 Alors P(S) Fsi;
V(mutexAB);
<Traverser AB>
P(mutexAB);
nAB:=nAB-1;
Si nAB=0 alors V(S) Fsi;
V(mutexAB);
Fin.
```




```
S                       : semaphore:=1;
mutexAB, mutexBA        : semaphore:=1;
nAB, nBA                : entier:=0 ;
```

```
Processus SensBA ();
Debut
P(mutexBA);
nBA:=nBA+1;
Si nBA=1 Alors P(S) Fsi;
V(mutexBA);
<Traverser BA>;
P(mutexBA);
nBA:=nBA-1;
Si nBA=0 alors V(S) Fsi;
V(mutexBA)
Fin.
```

Idées de base:

- La première voiture de chaque sens arrivant au point doit vérifier l'occupation du pont par l'autre sens.
  → compter le nombre de voitures en cours pour chaque sens → *nAB, nBA : entier:=0 ;*
- A la sortie du pont, seule la dernière voiture en cours doit libérer le pont.
  → le pont est en EM entre sens → *S: sémaphore:=1.*
- Protection des variables de synchronisation.

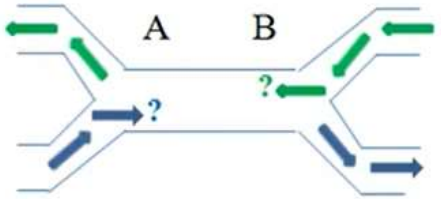
Exercice 2 : –

A- Ecrire un algorithme qui permet aux voitures de passer du nord vers le sud ou du sud vers le nord en se synchronisant à l'aide des sémaphores au niveau du pont.

--

Processus *Sens AB ();*
Debut
P(mutexAB);
nAB:=nAB+1;
Si *nAB=1* ***Alors*** *P(S)* ***Fsi;***
V(mutexAB);
<Traverser AB>
P(mutexAB);
nAB:=nAB-1:
Si *nAB=0* ***alors*** *V(S)* ***Fsi;***
V(mutexAB);
Fin.

Processus *SensBA ();*
Debut
P(mutexBA);
nBA:=nBA+1;
Si *nBA=1* ***Alors*** *P(S)* ***Fsi;***
V(mutexBA);
<Traverser BA>;
P(mutexBA);
nBA:=nBA-1;
Si *nBA=0* ***alors*** *V(S)* ***Fsi;***
V(mutexBA)
Fin.



f(mutexAB)

f(mutexBA)

f(S)

1AB: nAB=1; S=0
2AB: nAB=2
3AB: nAB=3
1BA: mutexBA=0; nBA=1; S=-1
2BA: mutexBA=-1
1AB: nAB=2
2AB: nAB=1
4AB: nAB=2

3BA: mutexBA=-2
3AB: nAB=1
4AB: nAB=0; S=0
1BA: mutexBA=-1
2BA: nBA=2; mutexBA=0
3BA: nBA=3; mutexBA=1
5AB: nAB=1; mutexAB=0;S=-1
6AB: mutexAB=-1
1BA: nBA=2
2BA: nBA=1;
3BA: nBA=0; S=0
5AB: mutexAB=0
6AB: nAB=2; mutexAB=1
5AB: nAB=1
6AB: nAB=0; S=1

Critique:

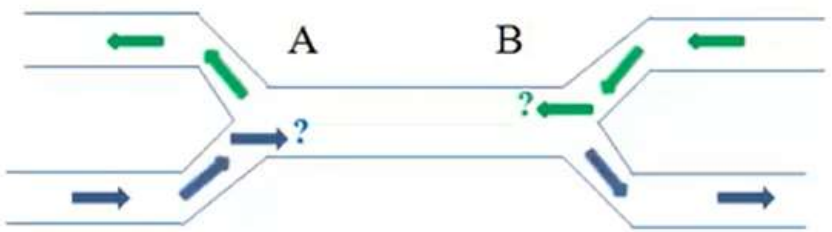
- Capacité du pont illimitée. Si capacité limitée, encadrer <Traverser> par *P(CP)* et *V(CP)*, *avec CP=capacité.*
- Problème de famine. En effet, si le trafic du sens en cours est continu, le sens en attente est **privé** de la traversée.

Exercice 2 :

--

B- Discutez votre solution et proposez d'éventuelles modifications pour diverses situations.

Processus *SensAB ();*
Debut
P(r);
P(mutexAB);
nAB:=nAB+1;
Si *nAB=1* ***Alors*** *P(S)* ***Fsi;***
V(mutexAB);
V(r);
<Traverser AB>
P(mutexAB);
nAB:=nAB-1:
Si *nAB=0* ***alors*** *V(S)* ***Fsi;***
V(mutexAB);
Fin.



S : sémaphore:=1;
r : sémaphore:=1;
mutexAB, mutexBA : sémaphore:=1;
nAB, nBA : entier:=0 ;

Processus *SensBA ();*
Debut
P(r);
P(mutexBA);
nBA:=nBA+1;
Si *nBA=1* ***Alors*** *P(S)* ***Fsi;***
V(mutexBA);
V(r);
<Traverser BA>;
P(mutexBA);
nBA:=nBA-1;
Si *nBA=0* ***alors*** *V(S)* ***Fsi;***
V(mutexBA)
Fin.

Idées de base:

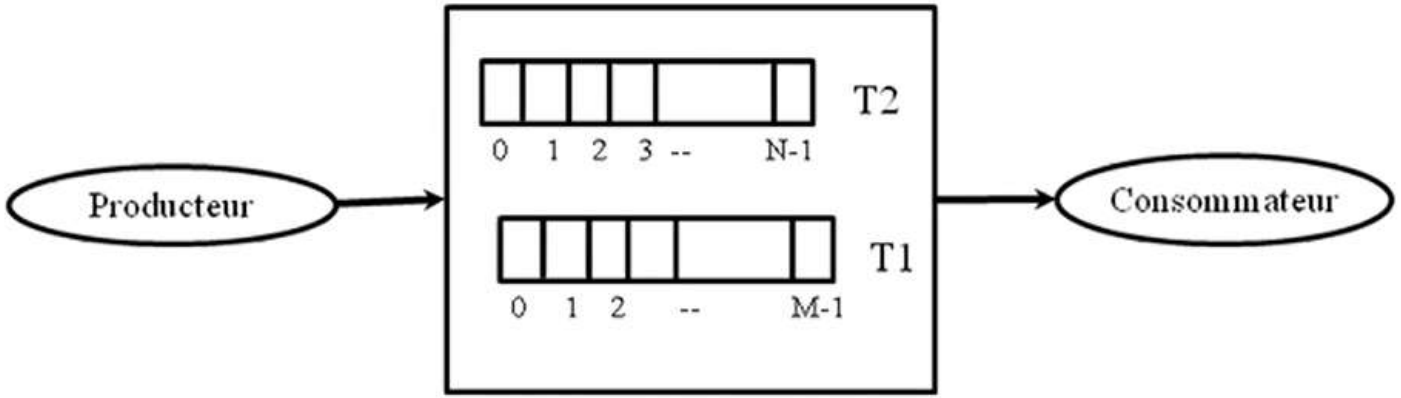
- Essayons d'augmenter la solution existante.
- Fifo → faire attendre les voitures sur une même file
  → *r: sémaphore:=1.*

Exercice 3 : Soit un système constitué d'un processus producteur et un processus consommateur qui se partagent deux tampons T1 et T2 de tailles respectives M et N, comme suit :

- Le producteur, à tout moment, ne peut déposer dans T2 que si le tampon T1 est plein.
- Le consommateur, à tout moment, ne peut prélever du tampon T2 que si le tampon T1 est vide.

A- Synchroniser les deux processus en utilisant les sémaphores.

B- Généraliser cette solution au cas de plusieurs producteurs et plusieurs consommateurs.



- Si on traite **séparément** les deux tampons
  → 4 sémaphores *mv1, mv2, np1, np2*.
  → Si tampon 1 est plein/vide, le producteur/consommateur se **bloque** alors que peut être l'autre tampon n'est pas plein/vide.
- Deux cas de figure : soit que le producteur **arrive à déposer**, soit **non** (ie. se bloque)
  → Un tampon de taille (*M+N*) qui **regroupe** *T1* et *T2*.
  → Uniquement **2** sémaphores *mv* et *np*.
  → **deux niveaux** d'opérations:
    - Vérifier globalement la **disponibilité** de la case à utiliser (en dépôt ou en prélèvement),
    - **Sélectionner** le tampon où opérer.

Exercice 3 : Soit un système constitué d'un processus producteur et un processus consommateur qui se partagent deux tampons T1 et T2 de tailles respectives M et N, comme suit :

- Le producteur, à tout moment, ne peut déposer dans T2 que si le tampon T1 est plein.
- Le consommateur, à tout moment, ne peut prélever du tampon T2 que si le tampon T1 est vide.

A- Synchroniser les deux processus en utilisant les sémaphores.

B- Généraliser cette solution au cas de plusieurs producteurs et plusieurs consommateurs.

```
nv      : semaphore:=M+N ;
np      : semaphore:=0 ;
mutex :semaphore =1;
cpt     : entier:=0 ;
 // nombre de cases pleines dans T1
```

```
Processus Prod () ;
Debut
Repeter
Produire(art) ;
P(nv) ;
P(mutex);
Si (cpt<M)  Alors
                    V(mutex) ;
                     Deposer(T1,art);
                     P(mutex) ;
                    cpt :=cpt+1;
                    V(mutex)
             Sinon
                  V(mutex) ;
                  Deposer(T2,art)
Fsi
V(np) ;
Jusqu'à (faux)
Fin.
```

```
Processus Cons ();
Debut
Repeter
P(np) ;
P(mutex);
Si (cpt>0) Alors
                    V(mutex) ;
                     Prelever(T1, art);
                     P(mutex) ;
                     cpt :=cpt-1 ;
                     V(mutex)
             Sinon
                     V(mutex) ;
                     Prelever(T2,Art);
Fsi ;
V(nv);
Consommer(art)
Jusqu'à (faux)
Fin.
```

--

B- Généraliser cette solution au cas de plusieurs producteurs et plusieurs consommateurs.

```
mv      : semaphore:=M+N ;
np      : semaphore:=0 ;
mutex :semaphore =1;
cpt     : entier:=0 ;
 // nombre de cases pleines dans T1
mutexp: semaphore:=1;
mutexc: semaphore:=1;
```

- Considérer les tampons comme **une seule entité en EM** au sein de chaque famille.

```
Processus Prod () ;
Debut
Repeter
Produire(art) ;
P(mv) ;
P(mutexp);
P(mutex);
Si (cpt<M)  Alors
                V(mutex) ;
                 Deposer(T1,art);
                 P(mutex) ;
                cpt :=cpt+1;
                V(mutex)
          Sinon
              V(mutex) ;
              Deposer(T2,art)
Fsi;
V(mutexp);
V(np)
Jusqu'à (faux)
Fin.
```

```
Processus Cons () ;
Debut
Repeter
P(np) ;
P(mutexc);
P(mutex);
Si (cpt>0) Alors
                V(mutex) ;
                 Prelever(T1, art);
                 P(mutex) ;
                 cpt :=cpt-1
          Sinon
                 V(mutex) ;
                 Prelever(T2,Art)
Fsi ;
V(mutexc);
V(mv);
Consommer(art)
Jusqu'à (faux)
Fin.
```

Exercice 4 : Soient 2 processus producteurs P_1, P_2 et deux processus consommateurs C_1, C_2 qui partagent un tampon de taille fixe. On définit les contraintes suivantes d'accès au tampon :

- P_i constitue une classe CL_i,
- C_i ne peut consommer que les messages produits par le processus de la classe CL_i,
- A tout instant, le tampon ne contient que les messages produits par le processus d'une seule classe.

A/ A l'aide des sémaphores, synchroniser l'ensemble de ces processus parallèles pour l'accès au tampon.

B/ Quelles seront les modifications nécessaires à apporter à la solution,

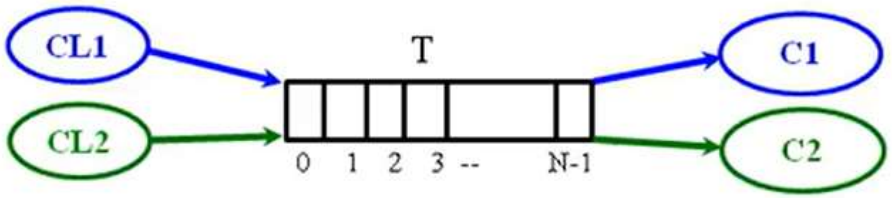
a- si on ajoute des processus producteurs à chaque classe CLi ?

b- et de plus, si on ajoute une classe CL_3 et un consommateur C_3 ?

Exercice 4 : --

A/ A l'aide des sémaphores, synchroniser l'ensemble de ces processus parallèles pour l'accès au tampon.

--



Processus *CL(i:[1..2]);*
Debut
Repeter
Produire(Art);
P(mutex[i]);
cpt[i]:=cpt[i]+1;
Si *cpt[i]=1* ***Alors*** *P(S)* ***Fsi;***
V(mutex[i]);
P(nv);
Deposer(art);
V(np[i])
Jusqu'à *(faux)*
Fin.

- Colorons les messages:
ceux de *CL[1]: en bleu* – de *CL[2]: en vert.*
- Le tampon est soit vide, soit contient les messages d'une même couleur
→Tampon en EM entre les deux couleurs
→ *S: semaphore:=1.*
→Similarité avec le problème du pont.
→Compter les messages des deux couleurs présents dans le tampon
→ *cpt: tableau[1..2] de entier:=0;*
mutex: tableau [1..2] de semaphore:=1.
- Le tampon est verrouillé au dépôt du *premier* message et déverrouillé au prélèvement du *dernier* message.
- Le tampon est géré par les sémaphores : *np1:=0, np2:=0, nv:=N.*
- *nv* est suffisant étant donné qu'une seule classe est active à la fois. L'autre classe ne pourra accéder au tampon que si celle active le libère, donc pas de conflit d'accès sur *nv*.
- par contre, *np1* et *np2* sont nécessaires.

Processus *C(i: [1..2]);*
Debut
Repeter
P(np[i]);
Prelever(art) ;
V(nv) ;
P(mutex[i]) ;
cpt[i] :=cpt[i]-1 ;
Si *(cpt[i]=0)* ***Alors*** *V(S) Fsi ;*
V(mutex[i]);
Consommer(art)
Jusqu'à faux
Fin.

Exercice 4 : --

B/ Quelles seront les modifications nécessaires à apporter à la solution,
a- si on ajoute des processus producteurs à chaque classe CLi ?
a- et de plus, si on ajoute une classe CL_3 et un consommateur C_3 ?

a/ Plusieurs producteurs de chaque classe

Les dépôts se font en séquentiel → *On doit:*
- ajouter un sémaphore mutexp:=1 commun aux deux classes et encadrer Déposer() par P(mutexp) et V(mutexp)
- et instancier CL[i] au nombre de processus de la classe.

b/ Ajout de CL3 et C3

Il suffit d'instancier CL[i] et C[i] pour une troisième classe (i=3).
→ Ajouter np3, mutex3, cpt3.

Exercice 5 : Des processus « utilisation» et des processus « systèmes » se partagent n imprimantes.
Les processus systèmes ont la priorité pour l'acquisition d'une imprimante.
- Décrire le comportement de ces deux classes de processus pour leur synchronisation en utilisant les moniteurs dans deux cas de demandes :
A/ Une imprimante à la fois.
B/ k imprimantes à la fois.

```
imp: Moniteur ;
const n=...;
var cu, cs : condition ;
    nbr :entier ;
```

- *3 entrées: dem-s(), dem-u() et* ***lib ()***
- *Deux conditions.*

```
entree procedure dem_s () ;
Debut
Si (nbr=0) Alors cs.wait() Fsi;
  nbr :=nbr-1
Fin;
entree procedure dem_u () ;
Debut
Si (nbr=0) Alors cu.wait() Fsi;
  nbr :=nbr-1
Fin;
entree procedure lib () ;
Debut nbr:=nbr+1;
Si non cs.empty() Alors cs.signal()
                  Sinon cu.signal()
Fsi
Fin;
```

```
inititialisation
Debut
   nbr :=n
Fin
FinMoniteur.
```

```
Processus systeme;
-
imp.dem_s();
<utiliser imprimante>
imp.lib()
-
```

```
Processus utilisateur;
-
imp.dem_u();
<utiliser imprimante>
imp.lib()
-
```

Exercice 5 : --

B/ k imprimantes à la fois.

```
k_imp: Moniteur;
const n=...;
var cu, cs : condition ;
    nbr :entier ;
<déclaration des files explicites fu et fs d'entiers
+ leurs procédures et fonctions d'accès>

entree procedure dem_s (k: entier) ;
Debut
Si
     [non cs.empty ou (nbr<k)]
  Alors
     inserer(fs, k);
     cs.wait();
     extraire(fs)
Fsi;
 nbr :=nbr-k
 Fin;
entree procedure dem_u (k: entier) ;
Debut
Si
   [non cs.empty ou non cu.empty() ou (nbr<k)]
  Alors
       inserer(fu, k);
       cu.wait();
       extraire(fu)
Fsi;
 nbr :=nbr-k
 Fin;
```

- Mêmes conditions, même variable,
- Ajouter un paramètres **k** aux entrées.
- Pour satisfaire les processus bloqués, on doit connaître leurs besoins

→entretient de files explicites: ***fs*** et ***fu***.

```
entree procedure lib (k: entier) ;
Début
nbr:=nbr+k;
Tantque
      [ non cs.empty()
                    et (premier(fs)<=nbr)]
 Faire
     cs.signal()
 Fait;
Si cs.empty() Alors
 Tantque
      [non cu.empty()
                   et (premier(fu)<=nbr)]
  Faire
     cu.signal()
  Fait
 Fsi
Fin;
```

```
Processus systeme;
-
imp.dem_s(k);
<utiliser imprimante>
imp.lib(k)
-

Processus utilisateur;
-
imp.dem_u(k);
<utiliser imprimante>
imp.lib(k)
-
```

```
inititialisation
Debut
   nbr :=n
Fin
Fin Moniteur.
```

Exercice 6 : On considère 2 ressources appelées R1, R2. La ressource R1 existe en N1 exemplaires et la ressource R2 en N2 exemplaires.

On supposant que chaque processus peut demander :

- Soit 1 exemplaire de la ressource R1,
- Soit 1 exemplaire de la ressource R2,
- Soit 1 exemplaire de R1 et 1 exemplaire de R2

A/ Ecrire un moniteur gérant l'accès à ces ressources en donnant la priorité à celui qui exprime le troisième type de demande.

B/ Modifier la solution pour permettre l'accès FIFO.

Exercice 6 : –

A/ Ecrire un moniteur gérant l'accès à ces ressources en donnant la priorité à celui qui exprime le troisième type de demande.

```
alloc_prio: Moniteur ;
const n1=... ; n2=... ;
var nr1, nr2: entier;
nc3 :entier ;
 c1, c 2,c3 :condition ;

entree procedure dem_r1 ();
Debut
Si
       (nr1=0)
              ou (nr1<=nc3)
  Alors c1.wait ()
Fsi ;
nr1 :=nr1-1 ;
Fin ;
entree procedure dem_r2 ();
Debut
Si
      (nr2=0)  ou (nr2<=nc3)
   Alors c2.wait () Fsi ;
nr2 :=nr2-1 ;
Fin ;

 Intialisation
  Debut
   nr1:=n1;nr2:=n2 ;
    nc3:=0
  Fin
 FinMoniteur.
```

- 6 entrées du moniteurs
- 3 conditions
- Tenir compte des demandes des processus en attente demandant *R1* et *R2*

→Réserver les ressources demandées

→ Compteur *nc3*.

```
entry procedure dem_r12 () ;
Debut
Si
   (nr1=0) ou (nr2=0)
 Alors
       nc3 :=nc3+1 ;
        c3.wait () ;
        nc3 :=nc3-1
Fsi ;
nr1 :=nr1-1 ; nr2 :=nr2-1 ;
Fin ;

entree procedure lib_r12 () ;
Debut
nr1 :=nr1+1 ; nr2 :=nr2+1 ;
Si (nc3<>0)
  Alors c3.signal () ;
      Si (nr1>nc3) Alors  c1.signal () Fsi;
      Si (nr2>nc3) Alors  c2.signal () Fsi
  Sinon
      c1.signal (); c2.signal ()
Fsi
Fin ;
```

```
entree procedure lib_r1 ();
Debut
nr1 :=nr1+1 ;
Si
     (nc3<>0) et (nr2<>0)
  Alors c3.signal ()
  Sinon
     Si
        (nr1>nc3)
    Alors c1.signal ()  Fsi
Fsi
Fin ;

entree procedure lib_r2 ();
Debut
nr2 :=nr2+1 ;
Si (nc3<>0) et (nr1<>0)
       Alors c3.signal()
       Sinon
         Si  (nr2>nc3) Alors c2.Signal  Fsi ;
Fsi
Fin ;
```

nr1=0+1;nr2= 2+1; nc3=2

Exercice 6 : –

B/ Modifier la solution pour permettre l'accès FIFO.

```
alloc_fifo : Moniteur ;
const n1=...; n2=...;
var c1, c2,c3 :condition ;
    nr1, nr2: entier;
    nc3 :entier ;
< Déclaration de la file f et
ses procédures d'accès >

entree procedure dem_r1 () ;
Debut
Si
    (nr1=0)
          ou (nr1<=nc3)
  Alors
       inserer(f, 1) ;
        c1.wait () ;
       extraire(f, 1)
Fsi ;
nr1 :=nr1-1
Fin ;
```

Mêmes variables et conditions

Fifo→ utiliser une file commune.

→ file explicite *f* de **numéros de types de demandes**

- Réservation de ressources aux processus bloqués.

```
initialisation
Debut
  nr1:=n1;
  nr2 :=n2
nc3:=0
Fin ;
```

```
entree procedure dem_r2 ();
Debut
Si
     (nr2=0) ou (nr2<=nc3)
        Alors
              inserer(f, 2) ;
              c2.wait () ;
              extraire(f, 2)
Fsi ;
nr2 :=nr2-1
Fin ;

entry procedure dem_r12 ();
Debut
Si
     (nr1=0)
           ou (nr2=0)
      Alors
              inserer(f, 3) ;
              nc3 :=nc3+1 ;
              c3.wait () ;
              extraire(f, 3) ;
              nc3 :=nc3-1
Fsi ;
nr1 :=nr1-1 ; nr2 :=nr2-1
Fin ;
```

Exercice 6 : –

B/ Modifier la solution pour permettre l'accès FIFO.

```
entry procedure lib_r1 () ;
Debut
nr1 :=nr1+1 ;
Si
    (non c1.empty ())
        et (nb_avant(3,1)<nr1))
 Alors c1.signal ()
 Sinon
    Si
       (non c3.empty ())
           et (nr2<>0)
       Alors c3.signal ()
    Fsi
Fsi
Fin ;
```

```
entree procedure lib_r2 () ;
Debut
nr2 :=nr2+1 ;
Si (non c2.empty ()) et (nb_avant(3,2)<nr2)
  Alors c2.signal ()
  Sinon Si non c3.empty ()et (nr1<>0)
           Alors c3.signal ()
        Fsi
Fsi
Fin ;
```

```
entree procedure lib_r12 () ;
Debut
nr1 :=nr1+1 ; nr2+1 ;
Si non vide(f)
 Alors
    Si
       (premier(f)=3)
     Alors c3.signal ();
          Si (nb_avant(3,1)<nr1) Alors c1.signal () Fsi;
          Si (nb_avant(3,2)<nr2) Alors c2.signal () Fsi
     Sinon
       Si
          (premier(f)=1)
        Alors c1.signal () ;
             Si (nb_avant(3,2)<nr2) Alors c2.signal () Fsi
        Sinon
           Si
              (premier(f)=2)
              Alors c2.signal () ;
               Si (nb_avant(3,1)<nr1) Alors c1.signal () Fsi
           Fsi
       Fsi
    Fsi
Fsi
Fin ;
```

Exercice 7 : Etant donné un système contenant N processus évoluant de manière parallèle et dont le fonctionnement de chacun est donné comme suit :

```
Processus P(i : entier) ;   /* i : identité du processus pouvant prendre une valeur de 1 à N */
Var j : entier ; T : Tableau [1..N] de Booleen :=Faux ;
Debut -
Pour j :=1 à N Faire
        Envoyer (B, 'présent', i)
Fait ;
Repeter
        Recevoir (B , m, k) ;
        Si T[k]= Faux Alors T[k] := Vrai  Sinon Envoyer (B, m, k) Fsi ;
        j :=1 ;  Tantque (j<=N) et (T[j]= Vrai Faire j :=j+1 Fait
Jusqu'à (j>N)
Fin.
```

Où B est une boite aux lettres commune (peut être utilisée par chaque processus en émission et en réception) supposée de capacité infinie.

• Envoyer (B, m, i) : permet d'envoyer un message m accompagné d'un entier i vers la boite aux lettres B.

• Recevoir (B, m, k) : permet de recevoir un message m accompagné d'un entier k dans la boite aux lettres B. Cette primitive bloque le processus appelant jusqu'à la réception du message.

A/ Discuter cette solution.

- Si on suppose que chaque processus P(i) i=1, N possède sa propre boite aux lettres B(i) (donc utilisée uniquement pour la réception de messages),

B- Réécrire la solution précédente avec cette nouvelle considération.

C- Comparer les deux solutions.

Exercice 7 : --

A/ Discuter cette solution.

--

```
Processus P (i : entier) ;
Var j : entier ; T : Tableau [1..N] de Booleen :=faux ;
Debut -
Pour j :=1 à N Faire
   envoyer (B, 'présent', i)
Fait ;
Repeter
  recevoir (B , m, k) ;
   Si T[k]= faux Alors T[k] := vrai
                 Sinon envoyer (B, m, k)
   Fsi ;
j :=1;  Tantque (j<=N) et (T[j]= vrai) Faire j :=j+1 Fait
Jusqu'à (j>N)
Fin.
```

- Deux étapes:
  - envoi de messages dans B
    → annonce de l'arrivée
  - parcours du tableau pour vérification du contenu
    → vérification de l'arrivée des autres processus

→ **RDV de N processus.**

- Il y a suffisamment de messages pour tous les processus
→pas de blocage, si tous les processus arrivent au point de RDV.
- **Utilisation de la communication pour réaliser de la synchronisation.**

Exercice 7 -

Si on suppose que chaque processus P(i) i=1, N possède sa propre boite aux lettres B(i) (donc utilisée uniquement pour la réception de messages),

B- Réécrire la solution précédente avec cette nouvelle considération.

C- Comparer les deux solutions.

B- Récriture de la solution

```
Processus P (i : entier) ;
Var j : entier ;
Debut
-
Pour j :=1 à N Faire
  Si (j<>i)
        Alors envoyer (B[j], 'présent', i)
  Fsi
Fait ;
Pour j :=1 à N-1 Faire
    recevoir(B[i], m, k)
Fait
Fin.
```

- Deux étapes:
  - Annonce de l'arrivée
  - Vérification de l'arrivée des autres processus

C- Comparaison

	Taille mémoire	Type d'attente	Nombre de messages (émission /réception)
Solution 1	N*N messages +N*N bits	Active	>> (N*N / >>N*N
Solution 2	N*(N-1) messages	Passive	N* (N-1) / N*(N-1)

Exercice 8 Dans un système d'exploitation, on dispose des primitives de communication par boites aux lettres utilisant les primitives suivantes :

Send (B, message) : Dépôt de message dans la boite aux lettres B.

Receive (B, message) : Attente et Retrait de message de la boite aux lettres B.

Soit le processus suivant:

```
Processus P ;
Struct m, mess : ….. ;
Debut
   -
   Send( B1, ''vide'') ; Send (B2,''vide'') ;
   Repeter
      Receive (B1, m) ;
      Si (m<>''vide'')  Alors mess ← m
                        Sinon  Send (B1,''vide''); Receive (B2, m) ;
                               Si (m<>''vide'') Alors mess ←m
                                                Sinon Send (B2,''vide'')
                               Fsi
      Fsi
    Jusqu'à (mess<>''vide'')
Fin.
```

B1 et B2 étant deux boites aux lettres communes à P et à d'autres processus.

1/ Expliquer le fonctionnement de la solution, déduire sa fonction.

- Remplaçons la primitive Receive par la fonction Read (B) qui retourne le premier message de la boite aux lettres B s'il existe sinon vide,

2/ Réécrire le processus P.

Exercice 8 --

1/ **Expliquer le fonctionnement de la solution, déduire sa fonction.**

--

```
Processus P ;
Struct m, mess : ..... ;
Debut
   -
   send( B1, ''vide") ; send (B2,"vide") ;
   Repeter
      receive (B1, m) ;
      Si (m<>"vide")
            Alors mess ← m
            Sinon  send (B1,"vide"); receive (B2, m) ;
                   Si (m<>"vide")
                         Alors mess ← m
                         Sinon send (B2,"vide")
                   Fsi
      Fsi
    Jusqu'à (mess<>"vide") ;
Fin.
```

- Envoi de messages vides
- Si le message reçu dans une boite aux lettres est ''vide'', cela veut dire qu'aucun message n'est reçu dans cette boite aux lettres.
- Le renvoi de message est fait si la boite est vide.

Fonction:

Attente de réception d'un message <u>unique</u> qui sera reçu exclusivement dans l'une des deux boites aux lettres B1 ou B2.

- L'envoi de messages vides permet d'éviter au processus de se bloquer dans la réception au niveau d'une boite aux lettres alors que le message attendu est envoyé sur l'autre boite aux lettres.

Exercice 8 --

- Remplaçons la primitive Receive par la fonction Read (B) qui retourne le premier message de la boite aux lettres B s'il existe sinon vide,

2/ Réécrire le processus P.

```
Processus P () ;
Struct m : ..... ;
Debut
   -
  Repeter
    m:= read(B1) ;
     Si (m=null) Alors m:=read (B2)  Fsi
  Jusqu'à (m<>null);
-
Fin.
```