Exo__1 :

Comme montre le tableau ci-dessous, il existe quatres (04) possibilités différentes d'écriture (affichage) après une exécution en pseudo-parallèle (en concurrence) de P1 et P2.

Bienvenu Aurevoir Mohamed Amir	Bienvenu Aurevoir Amir Mohamed
Aurevoir Bienvenu Amir Mohamed	Aurevoir Bienvenu Mohamed Amir

Exo__2 :

	Valeur finale de Nb_place	état logique du résultat
parallèle réal	0	Faux car réserver la même pale deux fois
en pseudo-parallèle	-1	Faux car réserver la même placee deux fois.

Exo__3:

Sin , Sout : **semaphore** // on déclarer 02 variables de type sémaphore.
Init(Sin,1); // ou Sin.n ←1 , initialiser "Sin" par 1.
Init(Sout,1); // ou Sout.n ←1 , initialiser "Sout" par 1.

```
Processus P1
{
P(Sout);
 out=out+1 ;
V(Sout);

P(Sin);
 in=in-1 ;
V(Sin);
```

```
Processus P2
{
  P(Sout);
   out=out-1;
  V(Sout);

}
```

```
Processus P3
{
  P(Sin);
   in=in+1 ;
  V(Sin);

}
```

Remarques :

La variable "in" peut être manipulé par P1 et P3 en même temps (lors d'une exécution en concurrence) alors on est besoin d'un sémaphore, soit "Sin", pour assurer l'accès en exclusion mutuelle (**E.M**) à la variable "in". Le semaphore "Sin" devra initialiser par "1" pour permettre à un (01) seul processus d'agir sur la variable "in" à la fois.

La variable "out" peut être manipulé par P1 et P2 en même alors on est besoin d'un sémaphore, soit "Sout", pour assurer l'accès en E.M à la variable "out". Le semaphore "Sout" devra initialiser par "1" pour permettre à un (01) seul processus d'agir sur la variable "out" à la fois.

- On peut déclarer et initialiser directement les sémaphores comme ceci : sempahore Sin←1, Sout←1;

Exo__4:

- Cette proposition n'est pas correcte car il elle ne respecte pas la 3ème condition de E.M de Dijkstra (Si un processus est bloqué en dehors de sa section critique (S.C) alors ce blocage ne doit pas empêcher un autre processus d'entrer dans S.C). Dans cette proposition, lors d'une exécution en parallèle tous les processus seront bloqué sur des sémaphores au niveau de leurs deuxième Section critiques : P1 se bloque sur P (MUTEX2), P2 se bloque sur P(MUTEX3), P3 se bloque sur P(MUTEX3).

- Proposition d'une solution correcte respectant les conditions de Dijkstra

Int a=b=c=0;
Sémaphore MUTEX1=1, MUTEX2=1, MUTEX3=1;

PROCESSUS P1	**PROCESSUS P2**	**PROCESSUS P3**
P(MUTEX1) ; a←a+1; V(Mutex1) ; P(MUTEX2) ; b=b+1 ; V(MUTEX2) ;	P(MUTEX2) ; b←b-1; V(MUTEX2) ; P(MUTEX3) ; c←c-1 ; V(MUTEX3) ;	P(MUTEX3) ; c←c+1; V(MUTEX3) ; P(MUTEX1) ; a←a-1 ; V(MUTEX1) ;

Exo__5:

Soit Resultat = (a+ b) * (c + d) - (e/f). Le processus P2 calcul R2=c+d; le processus P3 calcul R3=e/f, alors P1 calcul d'abord R1 = (a+b) puis Resultat =R1*R2-R3; il faut synchroniser entre P1 et P2 & P3.

Voici la solution (déclaration et initialisation des variables globale et codes des processus):

SR2 : semaphore // pour synchroniser entre P1 et P2
SR3 : semaphore // pour synchroniser entre P1 et P3
Init (SR2,0); // initialiser SR2 par 0;
Init (SR3,0); // initialiser SR3 par 0;
int RESULTAT, R1,R2,R3;

```
Processus P1()
{
 R1=a+b;
 P(SR2);
 P(SR3);
Resultat=R1+R2-R3
}
```

```
Processus P2()
{
    R2=c+d;
    V(SR2);

}
```

```
Processus P3()
{
      R3=e/f ;
       V(SR3);

}
```

Exo__6:

sempahore S1,S2 : sempahore;
Init(S1, 0); Init(S2,1);

Processus P1
Debut
Répéter Tant que « Vrai »
P(S1);
I1;
V(S2);
Fin Tant que
Fin

Processus P2
Debut
Répéter Tant que « Vrai »
P(S2);
I2;
V(S1);
Fin Tant que
Fin

Remarque :

➔ S1 devra initialiser par "0" car P1 à l'état initial devra être bloqué jusque ce que P2 terminer l'exécution de "I2."
➔ S2 devra initialiser par "1" car P2 commence en premier.

Exo__7:

Les programmes des trois processus : P1 et P2 sont des processus producteurs et P3 est un processus consommateur (voir solution de problème un producteur/ consommateur dans chapitre 2, partie2). Voici la solution:

B1vide, B1plien : semaphore // pour l'accès au Bac B1;
Init(B1vide,1); Init(B1plien,0);
B2vide, B2plien : semaphore // pour l'accès au Bac B2;
Init(B2vide,1); Init(B2plien,0);

```
Processus P1()
{
 A=Fabriquer();
 P(B1vide);
    Deposer(A,B1);
 V(B1plien);
}
```

```
Processus P2()
{
 B=Fabriquer();
 P(B2vide);
    Deposer(B,B2);
 V(B2plien);
}
```

```
Processus P3()
{
 P(B1plien);
     Retirer(A,B1);
 V(B1vide);

 P(B2plien);
    Retirer(B,B2);
 V(B2vide);

  X= fabriquer(A,B);
}
```

Exo__8:

int Nb_AB=0; // Compter le nombre de train circulant de A vers B.
int Nb_BA=0; // Compter le nombre de train circulant de B vers A.
Semaphore Svoie←1; // sémaphore d' E.M pour l'accès à la voie.
Semaphore S1←1; //sémaphore de E.M pour éviter l'accès simultané à la variable Nb_AB;
Semaphore S2←1;// sémaphore de E.M pour éviter l'accès simultané à la variable Nb_BA;

```
Processus Train_AB()
{
P(S1);
    Nb_AB= Nb_AB+1;
    Si Nb_AB=1 alors P(Svoie);
V(S1);
  <Circulation de A vers B;>
P(S1);
    Nb_AB= Nb_AB-1;
    Si Nb_AB=0 alors V(Svoie);
V(S1);
}
```

```
Processus Train_AB()
{
  P(S2);
    Nb_BA= Nb_BA+1;
    Si Nb_BA=1 alors P(Svoie);
  V(S2);
     <Circulation de A vers B;>
P(S2);
    Nb_BA= Nb_BA-1;
    Si Nb_BA=0 alors V(Svoie);
V(S2);
   }
```

Remarque: la solution de cet exercice se ressemble à la solution de processus lecteurs de problème Lecteurs/rédacteurs (voir le cours de chapitre 2, partie 2) mais avec deux catégories de lecteurs qui correspond deux catégories de trains : train(A→B) et Train(B→A).

Exo__9:

En utilisant des sémaphores, écrire les programmes de rendez-vous.

cas 1 : Programmes de rendez-vous de 2 Processus :

semaphore S1←0:, S2←0;

```
Processus P1()
{
  V(S2);
  P(S1);
 < rendez-vous >
}
```

```
Processus P2()
{
   V(S1);
   P(S2);
   < rendez-vous >
}
```

cas 2 : Programmes de rendez-vous de 3 Processus :

semaphore S1←0:, S2←0; S3←0;

```
Processus P1()
{
  V(S2);
  V(S3):
  P(S1);
 < rendez-vous >
}
```

```
Processus P2()
{
    V(S1);
    V(S3):
    P(S2);
    < rendez-vous >
}
```

```
Processus P3()
{
     V(S1);
     V(S2):
     P(S3);
   < rendez-vous >
}
```

Cas 3 : Programmes de rendez-vous de N (N>3) Processus :

Idée de la solution (Faites la solution vous-même**)** : utiliser une variable (soit cpt initialisé à zéro) pour calculer le nombre processus qui viennent d'arriver au <rendez-vous> . chaque processus doit faire cpt←cpt+1; puis tester si cpt<N ce processus doit se bloquer sinon (le processus dans lequel cpt sera égale à N) il va réveiller tous les autres processus.