Série n°3 : Les moniteurs

Corrigé de l'exercice 1

1) 2 ressources indépendantes ==> 2 moniteurs

```
Processus Avion_Voyageurs
Début
|
| Gestion_Accès_Piste_PS1.Demande_accès();
|
|      <Atterrir sur PS1>;
|
| Gestion_Accès_Piste_PS1.Libérer_accès();
|
Fin.
```

```
Processus Avion_Marchandises
Début
|
| Gestion_Accès_Piste_PS2.Demande_accès();
|
|      <Atterrir sur PS1>;
|
| Gestion_Accès_Piste_PS2.Libérer_accès();
|
Fin.
```

```
Gestion_Accès_Piste_PSi=1ou2 : moniteur ;
   PSi : Condition ;
   occupée_PSi: booléen ;

Procédure Demande_accès()
Début
|     Si occupée_PSi = vrai Alors PSi.wait; Fsi ;
|     occupée_PSi : = vrai ;
Fin ;

Procédure Libérer_accès()
Début
|     occupée_PSi := faux ;
|     PSi.signal ;
Fin ;

Initialisation
Début
|     occupée_PSi := faux ;
Fin ;
```

2) Favoriser l'avion qui possède **la plus faible réserve** en carburant.

```
Processus Avion_Voyageurs
Début
|
|Gestion_Accès_Piste_PS1.Demande_accès(R);
|
|      <Atterrir sur PS1>;
|
| Gestion_Accès_Piste_PS1.Libérer_accès();
|
Fin.
```

```
Processus Avion_Marchandises
Début
|
|Gestion_Accès_Piste_PS2.Demande_accès(R);
|
|      <Atterrir sur PS1>;
|
| Gestion_Accès_Piste_PS2.Libérer_accès();
|
Fin.
```

```
Gestion_Accès_Piste_PSi=1ou2 : moniteur ;
var
   PSi : Condition ;
   occupée_PSi: booléen ;

Procédure Demande_accès(R)
Début
|     Si occupée_PSi = vrai Alors PSi.wait(R); Fsi ;
|     occupée_PSi : = vrai ;
Fin ;

Procédure Libérer_accès()
Début
|     occupée_PSi := faux ;
|     PSi.signal ;
Fin ;

Initialisation()
Début
|     occupée_PSi := faux ;
Fin ;
```

Corrigé de l'exercice 2

2 ressources R1 et R2 avec 3 types de demande et le 3eme type est plus prioritaire ===> 1 seul moniteur qui gère les 3 types de demande.

```
Processus Pi
Début
|
| Allocation_Ressources.Demande_R1();
|
|   <Utiliser l'exemplaire de R1>;
|
| Allocation_Ressources.Libérer_R1();
|
Fin.
```

```
Processus Pi
Début
|
| Allocation_Ressources.Demande_R2();
|
|   <Utiliser l'exemplaire de R2>;
|
| Allocation_Ressources.Libérer_R2();
|
Fin.
```

```
Processus Pi
Début
|
| Allocation_Ressources.Demande_R1R2();
|
|   <Utiliser les exemplaires de R1 et R2>;
|
| Allocation_Ressources.Libérer_R1R2();
|
Fin.
```

```
Allocation_Ressources: moniteur;
var
    Dispo1; Dispo2, NB3 : entier;
    C1, C2, C3 : condition;
    // NB3 permet de compter le nombre de processus bloqués qui
       exprime le 3ième type de demande

Procédure Demande_R1();
Début
| Si (Dispo1 <= NB3)  Alors C1.wait;  fsi;
| Dispo1:= Dispo1 - 1;
fin;

Procédure Demande_R2();
Début
| Si (Dispo2 <= NB3)  Alors C2.wait;  fsi;
| Dispo2:= Dispo2 - 1;
fin;

Procédure Demande_R1R2();
Début
| Si (Dispo1 = 0) ou (Dispo2 = 0) Alors NB3 ++;
| |                                    C3.wait;
| |                                    NB3 --;
| Fsi;
| Dispo1:= Dispo1 - 1;
| Dispo2:= Dispo2 - 1;
fin;

Procédure Libérer_R1();
Début
| Dispo1:= Dispo1 + 1;
| Si (C3.empty = faux) et (Dispo2 != 0)
| |              Alors  C3.signal;
| |              Sinon  Si (Dispo1 > NB3)  Alors C1.signal; fsi;
| Fsi;
fin;

Procédure Libérer_R2();
Début
| Dispo2:= Dispo2 + 1;
| Si (C3.empty = faux) et (Dispo1 != 0)
| |              Alors  C3.signal;
| |              Sinon  Si (Dispo2 > NB3)  Alors C2.signal; fsi;
| Fsi;
fin;

Procédure Libérer_R1R2();
Début
| Dispo1:= Dispo1 +1;
| Dispo2:= Dispo2 +1;
| Si (C3.empty = faux) Alors  C3.signal;
| |                           Si (Dispo1 > NB3)  Alors C1.signal; fsi;
| |                           Si (Dispo2 > NB3)  Alors C2.signal; fsi;
| |                    Sinon  C1.signal; C2.signal;
| Fsi;
fin;

Initialisation()
Début
|  Dispo1:=N1;
|  Dispo2:=N2;
|  NB3:=0;
Fin;
```

Corrigé de l'exercice 3

Nous avons :

- Deux classes de processus système et utilisateur tel que les processus système sont plus prioritaire que les processus utilisateur.
- N imprimantes partagées entre ces deux classes.
- L'ordre FIFO est appliqué au sein de la même classe.

Cas 1: demande d'une seule imprimante à la fois

```
Processus sys
Début
|
| Allocation_Ressources.Demande_Imp_Sys();
|
|   <Utiliser l'imprimante>;
|
| Allocation_Ressources.Libérer_Imp();
|
Fin.
```

```
Processus user
Début
|
| Allocation_Ressources.Demande_Imp_Util();
|
|   <Utiliser l'imprimante >;
|
| Allocation_Ressources.Libérer_Imp();
|
Fin.
```

```
Allocation_Ressources: moniteur;
var
    Dispo : entier;
    sys, util : condition;

Procédure Demande_Imp_Sys();
Début
| Si (Dispo = 0)  Alors sys.wait;  fsi;
| Dispo:= Dispo - 1;
fin;

Procédure Demande_Imp_util();
Début
| Si (Dispo = 0)  Alors util.wait;  fsi;
| Dispo:= Dispo - 1;
fin;

Procédure Libérer_Imp();
Début
| Dispo:= Dispo + 1;
| Si (sys.empty = faux) Alors  sys.signal;
| |                    Sinon  util.signal;
| Fsi;
fin;

Initialisation()
Début
|  Dispo:=N;
Fin;
```

Cas 2: demande de K imprimantes à la fois

Le fait que les processus demandent un nombre d'exemplaires quelconque, nous avons besoin de sauvegarder le nombre d'exemplaires demandé par chaque processus pour vérifier si sa demande pourra être satisfaite avant de le réveiller.

==> deux listes chainées LSys, LUtil sont utilisées pour ce but.

```
Processus sys
Début
|
| Allocation_Ressources.Demande_Imp_Sys(k);
|
|   <Utiliser l'imprimante>;
|
| Allocation_Ressources.Libérer_Imp(k);
|
Fin.
```

```
Processus user
Début
|
| Allocation_Ressources.Demande_Imp_Util(k);
|
|   <Utiliser l'imprimante >;
|
| Allocation_Ressources.Libérer_Imp(k);
|
Fin.
```

```
Allocation_Ressources: moniteur;
var
    Dispo : entier;
    sys, util : condition;
    enrg { n: entier; //correspond au nombre d'exemplaires demandé
           svt :^enrg;}
    LSys, LUtil : file d'attente de type eneg;

Procédure Demande_Imp_Sys(k);
Début
| Si (Dispo < k) ou (sys.empty = faux)
| |              Alors Enfiler(k,LSys); sys.wait;
| |              Sinon Dispo:= Dispo - k;
| Fsi;
Fin;

Procédure Demande_Imp_util(k);
Début
| Si (Dispo < k) ou (sys.empty = faux) ou (util.empty = faux)
| |              Alors Enfiler(k,LUtil); util.wait;
| |              sinon Dispo:= Dispo - k;
| Fsi;|
Fin;

Procédure Libérer_Imp(k);
Début
| Dispo:= Dispo + k;
| Tant que (sys.empty = faux) et Dispo >= Tête(LSys).n
| Faire
| |   Défiler (x, LSys);
| |   dispo:= Dispo - x;
| |   sys.signal;
| Fait;
| Si (sys.empty = vrai)
| |            Alors Tant que (util.empty = faux) et Dispo >= Tête(LUtil).n
| |                   Faire
| |                   |  Défiler (x, LUtil);
| |                   |  Dispo:= Dispo - x;
| |                   |  util.signal;
| |                   Fait;
| Fsi;
Fin;

Initialisation()
Début
|  Dispo:=N;
|  Tête(LSys):=Nil;
|  Tête(LUtil):=Nil;
Fin;
```

Exercice 4

```
Processus Voiture
   Début
   |
   |     Accès_Pont.DemandeV() ;
   |       <Traverser le pont>
   |     Accès_Pont.LibérerV() ;
   |
   Fin.
```

```
Processus Camion
   Début
   |
   |     Accès_Pont.DemandeC() ;
   |       <Traverser le pont>
   |     Accès_Pont.LibérerC() ;
   |
   Fin.
```

```
Accès_Pont : Moniteur ;
Var  Cap : entier ;
     V, C: condition ;

Procédure DemandeV() ;
Début
|  Si (Cap = 0)  ou (⌉ C.empty) Alors V.wait Fsi ;
|  Cap := Cap - 2 ;
Fin;

Procédure DemandeC() ;
Début
|  Si (Cap < 8)   Alors C.wait Fsi ;
|  Cap := Cap - 8 ;
Fin;

Procédure LibérerV() ;
Début
|  Cap := Cap + 2 ;
|  Si (⌉ C.empty) et (Cap = 8) Alors  C.signal;
|  |                            Sinon V.signal;
|  Fsi;
Fin;

Procédure LibérerC() ;

Début
|  Cap := Cap + 8 ;
|  Si (⌉ C.empty) Alors  C.signal; Fsi;
|  Si ( C.empty) Alors  Tant que (⌉ V.empty) et (Cap >= 2)
|  |                   Faire
|  |                   |  V.signal;
|  |                   Fait;
|  Fsi;
Fin;

Initialisation()
Début
|  Cap:=10;
Fin;
```