Chapitre 2 : Synchronisation entre processus (**Partie 01**)

1 Processus Parallèles

Il existe trois façon pour l'exécution de 'N' processus : séquentielle, pseudo-parallèle (temps partagé), parallèle réel.

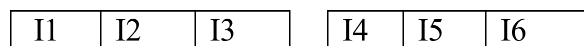
Voici un exemple de deux processus :

P1 est composée de l'exécution séquentielle des instructions I1, I2 et I3, on écrit P1= I1;I2; I3

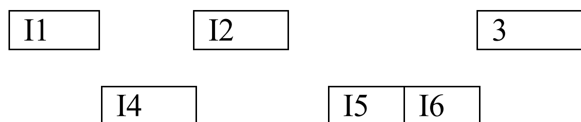
P2 est composée de l'exécution séquentielle des instructions I4, I5 et I6, on note P2= I4;I5; I6

Les trois façons d'exécution sont :

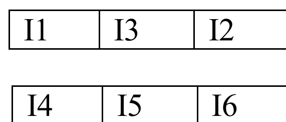
- Séquentielle : P1 puis P2 (sur un processeur)



- Pseudo-parallèle: P1 exécuté en temps processeur partagé avec P2 (sur un processeur)



- Parallèle réel : sur deux processeurs distinct.



2 Notion de ressources /section critique

2.1 Définition : Une ressource désigne toute entité dont a besoin un processus (PR) pour s'exécute. Elle est soit une :

- Ressource matérielle (Processeur, Périphérique)
- Ressource logicielle (variable, fichier).

En plus, la ressource est caractérisée par :
- Un état : libre/ occupée.
-Son nombre de points d'accès **(nombre de processus pouvant utiliser la ressource en même temps).**

2.2 L'utilisation d'une ressource : L'utilisation d'une ressource par un processus se fait de la manière suivante (trois étapes):

1 - Allocation
2- Utilisation
3- Restitution

Les phases d'allocation et de restitution doivent assurer que la ressource est utilisée conformément à son nombre de points d'accès.

Exemple : (ressource matérielle : imprimante)

Allocation : Occupée (1 point d'accès) ***impression*** ***Restitution*** : Libre (1 point d'accès).

2.3 Définitions **:**

- Une ressource avec un seul point d'accès appelée une ressource critique.

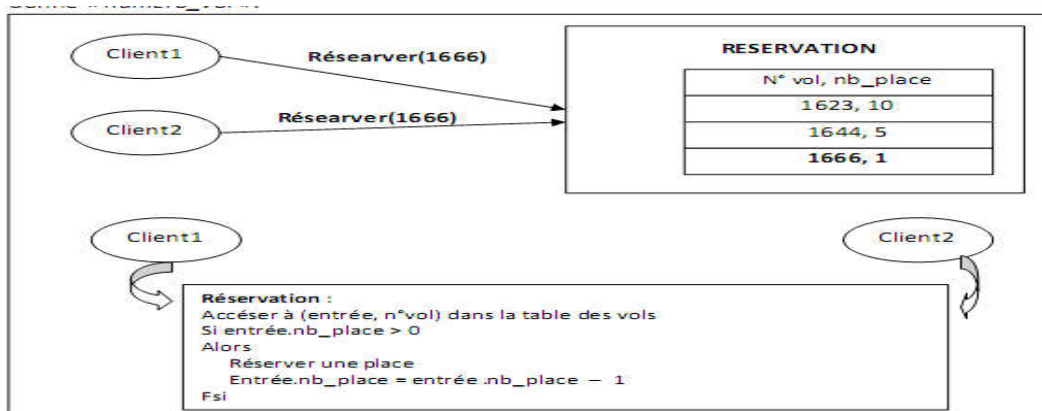
- Une section critique (**S.C**) est la partie du code dans laquelle le processus utilise la **R.C**

- Une section critique (**S.C**) est un ensemble de suites d'instructions qui peuvent produire des résultats imprévisibles lorsqu'elles sont exécutées simultanément par des processus différents.

2. Problème de l'exclusion mutuelle (E.M)

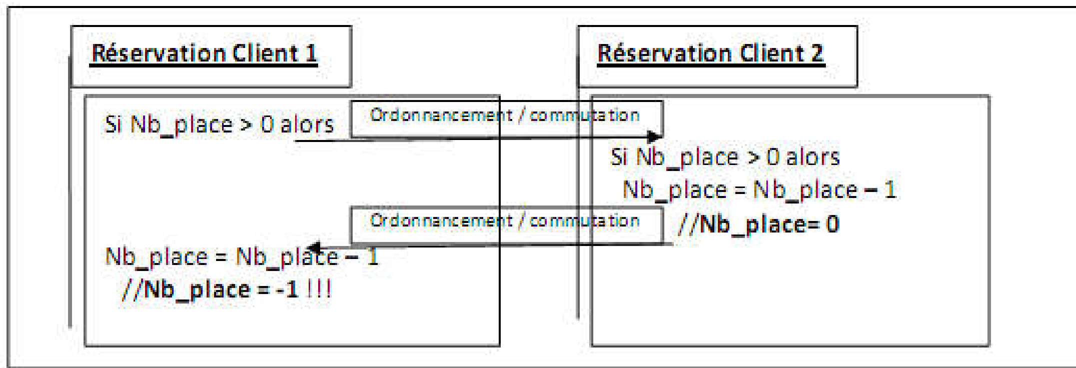
2.1 Position du problème de E.M

On considère un système permettant à des clients de réserver une place dans un avion donné « numéro_vol ».



On considère la situation où deux clients demandent simultanément la réservation d'une place sur le vol **1666** pour lequel reste une seule place disponible.

NB : les deux processus Réservation s'exécutent en concurrence ; le S.E ordonnance les de ux processus via un algorithme en temps partagé.



- **Probleme** : nb_place est **une ressource critique** , Que faut-il faire afin qu'un seul processus à la fois accède à nb_place (nb_place est **une ressource critique).**

- **Solution** : La SC doivent être exécutés en **Exclusion Mutuelle**. i.e. Avant d'exécuter une SC, un processus doit s'assurer qu'aucun autre processus n'est en train d'exécuter cette **S. C**.

2.2 Protocoles d'E.M

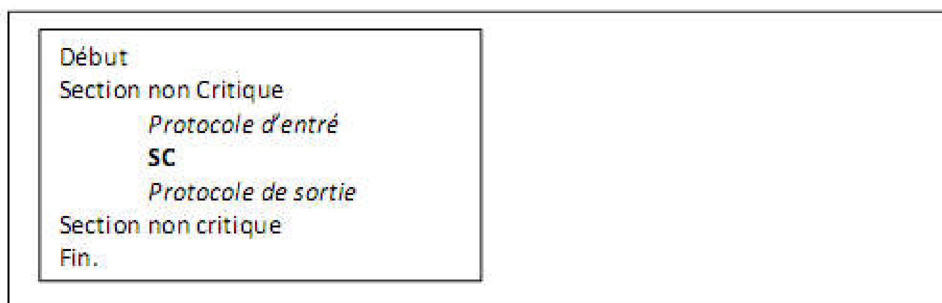
Pour éviter toute utilisation incorrecte d`une ressource critique, les suites d`instructions qui la manipulent (section critique) dans les différents processus ne doivent jamais s`exécuter simultanément. Les sections critiques de chaque processus doivent s`exécuter en Exclusion mutuelle.

L'exécution en Exclusion Mutuelle nécessite de définir un protocole d'entré en S.C et un protocole de sortie de S.C :

- **Protocole d'entré en S.C**: ensemble d'instructions qui permettent de vérifier que l`accès du processus à sa section critique est possible et aussi interdire l`accès des autres processus a leurs sections critiques.

- **Protocole de sortie de S.C:** ensemble d'instructions qui permet à un processus ayant terminé sa S.C d'avertir d'autres processus en attente que la voie est libre.

En fin, la structure des processus devient : (**modifier le : repeat….until**)



3 Propriétés de l'exclusion Mutuelle (Les quatre conditions de Dijkstra)

Il existe quatre conditions pour réaliser correctement une exclusion mutuelle :

Conditions 1. A tout instant, un seul processus peut avoir accès à sa section critique.

Conditions 2. Si aucun processus n`est dans sa section critique, un processus en attente doit pouvoir accéder à sa section critique au bout d`un temps fini. En d`autre terme, la section critique est toujours atteignable.

Conditions 3. Si un processus est bloqué en dehors d`une section critique, ce blocage ne doit pas empêcher l`entrée d`un autre processus en sa section critique.

Conditions 4. La solution doit être la même pour tous les processus. En d`autre terme, Aucun processus ne doit jouer de rôle privilégié.

4. Solution matérielle au problème d`exclusion mutuelle

Dans cette catégorie on distingue :

4.1 Masquage des IT (Monoprocesseurs): Il permet au processeur d'exécuter le code de la section critique jusqu'à sa fin sans être 'interrompu par un autre processus.
<Entré en SC> : Masquer les IT.

<S.C>

<Sortie de SC> : Démasquer les IT.

Inconvénients :

- Cette solution ne peut être utilisée dans les systèmes multiprocesseurs car le faite de masquer les interruptions d'un processeur n'entraîne pas le masquage des autres processeurs.

- Temps passé en S.C est non contrôlable.

4.2 L`instruction 'Test and Set' (TS ou TAS) (Multiprocesseurs):

• L'instruction Test and Set Lock « int TSL (int b) » exécute de manière indivisible (elle ne peut pas interrompu, car elle exécuté dans un seul cycle mémoire) les opérations suivantes :

– récupère la valeur de b,
– affecte la valeur 1 à b et
– retourne l'ancienne valeur de b.

```
int  TAS (int *b)
{
  Int a=b;
  b=1 ;
  return a;
}
```

Lorsqu'un processeur exécute l'instruction TAS, il verrouille le bus de données pour empêcher les autres processeurs d'accéder à la mémoire pendant la durée de l'exécution.

Exemple d'utilisation :

Soit **verrou** une variable commune aux processus et elle est concernée par la section critique.

```
Int   Verrou =0 ;

Process Pi ;

{

…….

While TAS(verrou) do;

  <Section critique;>

verrou:= false;

………. ;

}
```

Inconvénient: - Attente active.

Remarque : il y a aussi L'instruction **SWAP** utilisé pour résoudre le problème de E.M , elle permute les valeurs de deux variables de manière atomique swap (a, b);

5 Solution algorithmique au problème d`exclusion mutuelle

Hypothèse. Nous considérons P0, P1 qui rentre en compétition pour une ressource critique R. Ces deux processus sont définis par le programme suivant :

```
Algoritme  gestion_processus ;
{Déclaration des variables communes à P0 et P1}
Processus P0 ;
    {Déclaration des variables locales à P0}
debut
…….. ;
Section critique/R ;
…….. ;
fin ;

Processus P1;
    {Déclaration des variables locales à P1}
debut
…….. ;
Section critique/R ;
```

```
…….. ;
fin ;
debut
 {initialisation des variables communes}
{Lancer l'exécution parallèle de deux processus }

fin ;
```

5.1 Algorithme 1 : Solution de 2 processus avec une Variable Commune

Utiliser une variable commune aux deux processus qui mémorise le tour du processus qui doit entrer en section critique. Cette variable est appelée **Tour**, selon sa valeur 0 ou 1, le processus P0 ou P1 accède à sa section critique. **Tour** est initialisée à 0.

Voici les codes de P0 et P1 exprimé en Langage C :

Processus P0	Processus P1

while (1)
{ /*attente active*/
while (tour !=0) ;
section_critique_P0() ;
tour = 1 ;
….
}
	while (1)
	{ /*attente active*/
	while (tour !=1) ;
	section_critique_P1() ;
	tour = 0 ;
	….
	}

Note. On note une attente active dans la boucle while où le processus refait les mêmes actions sans résultats.

CRITIQUE: Si si le Po est suspendu hors section crtique, le processus P1 reste bloqué malgré que un processus p0 qui n'est pas en section critique.

Conclusion. – Attente active consomme du temps CPU. La 2ème et le 3ème propriété de Dijkstra n`sont pas vérifiées → Solution Fausse.

5.2 Algorithme 2 : Solution de 2 processus avec un Tableau de booléen Commune

Voici les codes de P0 et P1 exprimé en pseudo code du langage PASCAL :

Process P0 ;	Process P1 ;

BEGIN
…….. ;
drapeau [0]:= true;
While Drapeau[1] do ;
<Section critique>
drapeau [o] := false ;
…….. ;
End ;
	BEGIN
	…….. ;
	drapeau [1]:= true;
	While Drapeau[0] do;
	<Section critique>
	drapeau [1]:= false;
	…….. ;
	End ;

Dans la deuxième solution, les processus P0 et P1 peuvent consulter le tableau d`indicateurs simultanément. Pour remédier à cela, on intervertit les 2 instructions (test et affectation). Dans ce cas, le tableau d`indicateurs n`a plus le même sens.

CRITIQUE. La deuxième propriété de Dijkstra n`est pas vérifiée.

Si les deux processus exécutent simultanément la première affectation, ils se bloqueront ensuite mutuellement, ce qui rend la section critique inatteignable.

Conclusion. Solution Fausse.

5.3 Algo 4 : Solution de Dekker

Solution de 2 processus avec une variable commune et un Tableau de booléen Commune. Dans cette solution, la première et la troisième proposition sont combinées. Les variables communes sont tour et drapeau.

Var Drapeau: array [0..1] of boolean ;

Tour: (0,1)

Initialement: for i:=0 to 1 do drapeau [i]:= false;

Tour: =0; ou 1

Process P0 ;	Process P1 ;
BEGIN	BEGIN
…….. ;	…….. ;
drapeau [0]:= true;	drapeau [1]:= true;
tour:= 1;	tour:= 0;
While (Drapeau[1]) and (tour=1) do;	While (Drapeau[0]) and (tour=0) do;
<Section critique>	<Section critique>
drapeau [o]:= false;	drapeau [1]:= false;
…….. ;	…….. ;
End ;	End ;

Conclusion. Solution correcte car les quatre propriétés de Dijkstra sont satisfaites.

6. Notion de sémaphore

Les solutions proposées pour le problème d'exclusion mutuelle ne peuvent pas être utilisées lorsqu'il s'agit de problèmes plus complexes. Ainsi, les **sémaphores** permettent de résoudre les différents types de problème de l'EM.

6.1 Principe. C'est de contrôler la synchronisation par l'utilisation d'un type de données abstrait appelé Sémaphore, proposé par Dijkstra (1965).

6.2 Définition.

• Un sémaphore est une variable, entière, positive, globale et protégé i.e. on peut y accéder qu'au moyen des trois procédures :

- Init(S, x) : initialiser le sémaphore S à une certaine valeur X ;
- P(S) ou down(S) : Peut-on passer ? Peut-on continuer ?
- V(S) ou up(S): Libérer ? vas y ?

Explications :

• la primitive (operation) P(S) est utilisé pour déterminer si un processus peut ou non continuer son exécution.

• Les processus qui ne peuvent pas continuer leur exécution sont mis dans une file associée au sémaphore et passent à l'état bloqué.

• la primitive V(S) est utilisé pour libérer un processus de la file de sémaphore en lui permettant de continuer son exécution.

• Ces deux opérations sont P et V et qui s'exécutent en exclusion mutuelle

6.3 Déclaration d'un sémaphore

Un sémaphore est une structure à deux champs :

- Une variable entière positive, ou valeur du sémaphore.
- Une file d'attente (une structure de liste) de processus.

```
struct {

    int  n; // nombre de processus pouvant utilise le sémaphore sans se bloquer

    List_of_process  en_attente ; //processus bloqué derrière le sémaphore

  }
```

1ère Réalisations logicielles :

- Init(S, x) {S.n = x ;}

- **P(S)** { S.n = S.n - 1 ;

  Si S.n <0 alors bloquer le processus en fin de S.en_attente ;}

- **V(S)**{ S.n = S.n +1 ;

  Si S.n <= 0 alors débloquer le processus en tête de S.en_attente ;}

NB : L'initialisation dépend du nombre de processus pouvant effectuer en même temps « sect ion critique »

- Exemple : m, si on a m imprimantes identiques.

- Cette implémentation donne à chaque fois dans S.n le nombre de ressources libres ;

- Lorsque S.n est négative, sa valeur absolue donne le nombre de processus dans la file d'attente S.en_attente.

2ème Réalisations logicielles :

- P(S){ Si S.n > 0 alors S.n = S.n - 1 ;

  Sinon bloquer le processus en fin de S.en_attente ;}

- V(S) {Si S.en_attente non-vide alors débloquer le processus en tête de S.en_attente ;

  Sinon S.n :=S.n +1 ;}

6.4 Propriétés des sémaphores

La définition d'un sémaphore et des primitives P et V a les conséquences suivantes:

- Un sémaphore ne peut pas être initialisé à une valeur négative, mais il peut devenir négatif après un certain nombre d'opérations P.
- Si plusieurs processus tentent d'entrer dans leurs sections critiques, ce n'est qu'un seul d'entre eux qui réussira à exécuter l'opération P(S)
- Un processus qui invoque la primitive V sur un sémaphore, réveillera un autre processus bloqué derrière ce sémaphore, si sa valeur est inférieure ou égale à 0.
- L'invocation de la primitive P sur un sémaphore par un processus peut avoir l'un des effets suivants:
  - Le processus sera bloqué et mis dans la liste associée au sémaphore; lorsque la valeur du sémaphore est inférieure à zéro.
  - Lorsque la valeur du sémaphore est supérieure ou égale à zéro; le processus continue son exécution normalement.

- La valeur d'un sémaphore dénote :

  - Soit le nombre de processus bloqués derrière ce sémaphore (valeur <0),

  - soit le nombre de processus qui peuvent exécuter la primitive P sans être bloqués (valeur>=0).

•Un sémaphore est dit binaire si sa valeur ne peut être que 0 ou 1, générale (de comptage) sinon.

L'utilisation correcte des sémaphores et des primitives P et V permet de résoudre divers problèmes de synchronisation. *Nous allons illustrer ceci à travers plusieurs exemples classiques d'utilisation des sémaphores.*

7. l'utilisation des sémaphores pour l'accès à une section critique

On considère deux processus P0 et P1 , en compétition pour l'entrée à une section critique.

Solution: L'exclusion mutuelle peut être garantie par un sémaphore initialisé à 1 (souvent **Mutex** est le nom symbolique donné à ce sémaphore).

Init(Mutex, 1) : initialiser le sémaphore Mutex à 1

<Entré en SC> : P(Mutex);

<S.C>

<Sortie de SC> : V(Mutex);

7.1 Exemples 1

Program Exclusionmutuelle;

Init (Mutex ,1);

```
 Process P1;
debut
 ………….;
        P(Mutex);
        Section critique;
         V(mutex);
         ……..;
fin

Process P0 ;

debut

 ………….;
```

P(Mutex);

Section critique;

V(mutex);

……..;

fin

7.2 Solution de problème de réservation par les sémaphores :

```
Mutex : sémaphore
Init(Mutex, 1)
Réservation :
P(Mutex)
 Si nb_place > 0 alors // réserver une place
    Nb_place = nb_place  - 1
 Fsi
V(Mutex)
```