OBJECTIFS DU COURS :

Introduire la problématique du parallélisme dans les systèmes d'exploitation et étudier la mise en œuvre des mécanismes d'exclusion mutuelle, de synchronisation, de communication dans un environnement centralisé.

Inculquer à l'étudiant les concepts et les outils de base des systèmes répartis.

PLAN DE COURS :

Chapitre I : Notions de parallélisme, de coopération et de compétition (10%)

- Parallélisme
- Outils de parallélisme
- Problématique

Chapitre II : Gestion de parallélisme : outils de synchronisation et de communication (70%)

- L'exclusion mutuelle (Définition, Propriétés, Méthodes, …)
- La synchronisation (Evènements, Verrous, Sémaphores, Moniteurs, …)
- La communication interprocessus (Partage de variables, Boites aux lettres, …)
- L'interblocage (Conditions, Préventions, Détection et reprise, …)

CHAPITRE I : NOTIONS DE PARALLÉLISME, DE COOPÉRATION ET DE COMPÉTITION

I/ Introduction :

Les systèmes d'exploitation sont passés par plusieurs générations depuis l'apparence de l'informatique jusqu'aujourd'hui, où parmi ces générations on trouve les systèmes qui concrétisent la notion de parallélisme citons : les systèmes multiprogrammation, les systèmes temps partagés, les systèmes répartis, les systèmes temps réel, …

L'objectif du parallélisme est d'avoir des machines performantes soit en temps d'exécution soit en complexité de programmes.

II/ Définitions :

a/ Simultanéité ou parallélisme :

On appelle simultanéité l'activation de plusieurs processus en même temps.

- Si le nombre de processeur est supérieur ou égal au nombre de processus on parle de la simultanéité totale ou parallélisme réel.
- Sinon, on parle du pseudo-simultanéité ou parallélisme apparent.

b/ Parallélisme apparent :

C'est l'exécution de plusieurs processus sur un seul processeur où on utilise une commutation temporelle pour basculer l'exécution entre ces processus.

Exemple : dans les systèmes temps partagé, on utilise l'ordonnancement par tourniquet pour réaliser ce parallélisme :

- Conceptuellement, chaque processus a son propre processeur virtuel et son compteur ordinal.
- Concrètement, un seul processeur et un seul compteur ordinal.

b/ Parallélisme réel :

C'est l'utilisation d'un ensemble de processeurs (non forcément identiques), la communication entre ces processeurs est faite via des lignes spécialisées.

Exemple : les systèmes répartis dont l'objectif est de masquer l'aspect répartition aux utilisateurs.

- L'utilisateur accède aux ressources éloignées de la même façon que les ressources locales.

III/ Outils de parallélisme :

a/ Décomposition en tâches :

1/ Définition :

On appelle une tâche une unité élémentaire de traitement ayant une cohérence logique. L'exécution du processus P est constituée de l'exécution séquentielle des tâches : T_1, T_2, …, T_n.

$$P = T_1 T_2 \ldots T_n$$

A chaque tâche T_i, on associe sa date de début ou d'initialisation d_i et sa date de terminaison ou de fin f_i.

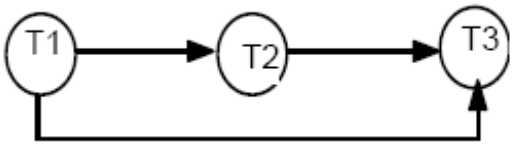
2/ Relation de précédence :

Une relation de précédence, notée **<**, sur un ensemble **E** est une relation vérifiant :

- ∀ T ∈ E, on n'a pas T < T
- ∀ T ∈ E et ∀ T' ∈ E, on n'a pas simultanément T < T' et T' < T
- La relation < est transitive

La relation $T_i < T_j$ entre tâches signifie que $\mathbf{f_i}$ inférieur à $\mathbf{d_j}$ entre dates. Si on n'a ni $T_i < T_j$, ni $T_j < T_i$, alors on dit que T_i et T_j sont exécutables en parallèle.

Une relation de précédence peut être représentée par un graphe orienté. Par exemple, la chaîne de tâches S = ((T_1, T_2, T_3), ($T_i < T_j$ pour i inférieur à j)) a pour graphe :

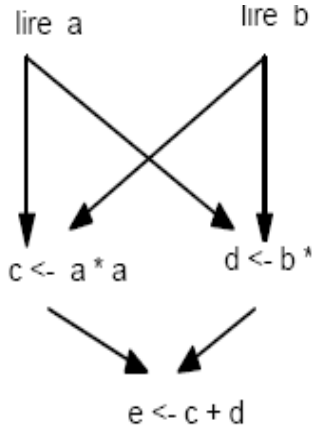


3/ Parallélisme de tâches :

La mise en parallèle s'effectue par la structure algorithmique :

Parbegin
….
Parend

Exemple :




```
debut
        parbegin
                lire a
                lire b
        parend
        parbegin
                c ← a * a
                d ← b * b
        parend
        e ← c + d
fin
```

4/ Le multitâches :

Plusieurs tâches peuvent être présentées et simultanément actives à un moment donné dans la MC. Ces tâches sont totalement indépendantes les uns des autres.

Le SE permet de partager le temps CPU entre plusieurs programmes qui semblent s'exécuter simultanément.

Exemple :

- Windows Me → multitâches coopératif puisque les temps libres de tâches principales sont utilisés pour traiter des tâches d'arrière-plan.
- Windows NT4, 2000, XP, UNIX → consiste à installer un programme ordonnanceur (Scheduler) qui alloue selon des critères de priorité le processeur à un programme.

5/ Langages parallèles :

Citons : CSP, Occam, ADA, Java, C, Fortron, …

b/ Le multithreading :

1/ Définition :

Un **processus léger** (en anglais, *thread*), également appelé **fil d'exécution** (autres appellations connues: unité de traitement, unité d'exécution, fil d'instruction, processus allégé), est similaire à un processus car tous deux représentent l'exécution d'un ensemble d'instructions du langage machine d'un processeur. Du point de vue de l'utilisateur, ces exécutions semblent se dérouler en parallèle. Toutefois, là où chaque processus possède sa propre mémoire virtuelle, les processus légers appartenant au même processus père se partagent sa mémoire virtuelle, sa section de code, sa section de données et d'autres ressources système. Par contre, tous les processus légers possèdent leur propre pile système.

Il comprend :

- Un identificateur,
- Un compteur de programme,
- Un ensemble de registres,
- Et une pile

Il partage avec son créateur :

- Les variables globales,
- Les descripteurs de fichiers,
- Les fonctions de gestion de fichiers.

2/ Utilisation :

Les processus légers sont typiquement utilisés avec l'interface graphique (GUI) d'un programme. En effet, les interactions de l'utilisateur avec le processus, par l'intermédiaire des périphériques d'entrée, sont gérées par un processus léger, tandis que les calculs lourds (en termes de temps de calcul) sont gérés par un ou plusieurs autres processus légers. Cette technique de conception de logiciel est avantageuse dans ce cas, car l'utilisateur peut continuer d'interagir avec le programme même lorsque celui-ci est en train d'exécuter une tâche. Une application pratique se retrouve dans les traitements de texte où la correction orthographique est exécutée tout en permettant à l'utilisateur de continuer à entrer son texte.

L'utilisation des processus légers permet donc de rendre l'utilisation d'une application plus fluide, car il n'y a plus de blocage durant les phases de traitements intenses.

Exemple : un traitement de texte peut avoir :

- Un thread qui affiche les graphiques.

- Un thread qui lit les touches tapées.
- Un thread qui effectue des corrections orthographiques.

3/ Les avantages du multithreading :

- Permet à une application de continuer l'exécution même si l'une de ses parties est bloquée ou effectue une opération de longue durée.
- Partage de ressources entre les threads d'un même processus.
- L'économie d'allocation mémoire et le temps d'exécution → plus de temps pour créer un processus que pour créer un thread.
- Bonne méthode dans les architectures parallèles.

IV/ Problématique :

On peut considérer deux processus **User** et **Client** logiquement parallèle, toute fois, ils ne se déroulent pas simultanément toujours. Lorsqu'**User** utilise le processeur, l'exécution de **Client** soit suspendue.

Ce conflit est dû à une insuffisance de ressources ainsi que d'autres interactions comme :

- **Client** ne doit pas pouvoir accéder à une zone mémoire qu'**User** est en train d'écrire.
- Lorsqu'une zone mémoire contient une valeur, **User** doit réveiller **Client** s'il est inactif.

Cet exemple met en évidence l'existence de différents types d'interaction entre les processus qui se coopèrent pour accomplir une tâche spécifique :

- Conflit pour l'accès simultané à une ressource qui ne peut être utilisée que par un seul processus à la fois.
- Action directe d'un processus sur un autre → mise en attente ou réveil.

Alors, on peut considérer un système d'exploitation comme un ensemble de processus parallèles pouvant interagir ; et pour mettre en œuvre ces processus tout en réalisant ces interactions, on a deux problèmes à résoudre :

1/ Ecrire des programmes décrivant chacun son activité individuelle.

2/ Concevoir des mécanismes d'interaction permettant les différents moyens de coopération et de compétition entre ces processus : Exclusion Mutuelle, Synchronisation et communication de l'information.

CHAPITRE II : GESTION DE PARALLÉLISME : OUTILS DE SYNCHRONISATION ET DE COMPÉTITION

I/ Introduction :

La coopération des processus pour accomplir une tâche commune nécessite l'existence d'un mécanisme qui permet l'échange d'information entre eux ainsi des outils de synchronisation et d'exclusion mutuelle pour contrôler leur ordre d'exécution.

II/ Exclusion mutuelle :

1/ **Exemple introductif :**

Considérons deux processus (peuvent être lourd ou léger) :

Entier X = 2000	
Processus 1 ------ Instructions ------- X := X+1000 ------ Instructions -------	**Processus 2** ------ Instructions ------- X := X-2000 ------ Instructions -------

On peut avoir plusieurs cas d'exécution :

1er cas : exécution séquentielle

Entier X = 2000	
Processus 1 ------ Instructions ------- X := X+1000 ------ Instructions -------	**Processus 2** ------ Instructions ------- X := X-2000 ------ Instructions -------
Charger (Registre A, X) Charger (Registre B, 1000) Additionner (Registre A, Registre B) Sauvegarder (X, Registre A)	
	Charger (Registre A, X) Charger (Registre B, 2000) Soustraire (Registre A, Registre B) Sauvegarder (X, Registre A)

⇨ Exécution des deux processus soit P1<P2 ou P2<P1 donne le même et le bon résultat.

2ème cas : Exécution parallèle (dans un système temps partagé)

Entier X = 2000	
Processus 1 ------ Instructions ------- X := X+1000 ------ Instructions -------	**Processus 2** ------ Instructions ------- X := X-2000 ------ Instructions -------
Charger (Registre A, X) Charger (Registre B, 1000)	
	Charger (Registre A, X) Charger (Registre B, 2000)

	Soustraire (Registre A, Registre B) Sauvegarder (X, Registre A)
Additionner (Registre A, Registre B) Sauvegarder (X, Registre A)	

⇨ Dans ce cas on peut imaginer plusieurs scénarios d'exécution où on aboutit à des résultats différents à chaque exécution ⇨ **incohérence de résultat**.

⇨ Le problème due à l'utilisation de la même variable X → une ressource critique.

2/ Définitions :

1/ Ressource : est toute objet dont un processus a besoin pour progresser dans son exécution (Mémoire centrale, CPU, Signaux, …)

2/ Ressource partageable : elle peut être allouée en même temps à plusieurs processus → elle est partageable à n points d'accès.

3/ Ressource critique : partageable à un seul point d'accès → elle ne peut être allouée qu'à un seul processus à la fois.

4/ Section critique : une section critique d'un processus est une séquence d'instructions qui utilisent une ressource critique.

3/ Définition de l'exclusion mutuelle :

L'exclusion mutuelle est un protocole qui protège une ressource critique contre les accès simultanés de plusieurs processus en coopération ou en compétition. Elle permet de restreindre l'accès à un seul processus à la fois.

D'une autre façon les processus s'excluent mutuellement pour accéder à une ressource critique.

4/ Les propriétés du protocole exclusion mutuelle :

Chaque solution utilisée pour réaliser l'exclusion mutuelle doit respecter :

- Définition : à tout instant un seul processus au plus peut se trouver en section critique.
- Atteignabilité : Si plusieurs processus sont bloqués en attente de la section critique et aucun processus n'est en section critique, alors l'un d'eux doit y accéder en un temps fini.
- Progression : un processus en attente à la section critique en un temps fini.
- Binalisation de la solution : Tout les processus doivent utiliser la même solution et aucun des processus ne joue de rôle privilégié.
- Un processus hors de sa section critique ou du protocole d'entrée ne doit pas influer sur le protocole de l'exclusion mutuelle (sur l'un des autres processus).
- Aucune hypothèse de doit être faite sur les vitesses des processus.

5/ Solutions pour l'exclusion mutuelle :

Pour écrire l'exclusion mutuelle entre les processus, on utilise le plus souvent :

- Attente active : variables de verrou, test & set, l'alternance)
- Attente passive : (sémaphores, moniteurs)
- Ainsi des solutions hardware (mémoire commune, masquage des interruptions)

5.a) Masquage des interruptions :

Chaque processus masque les interruptions avant d'entrer dans sa section critique, alors l'interruption horloge ne va pas avoir lieu et le processeur ne pourrait plus être alloué à un autre processus.

Inconvénient :

1/ Cette approche n'est pas intéressante, c'est dangereux de permettre aux processus utilisateurs de masquer les interruptions → peur de l'oubli de démasquer les interruptions → serait la fin du système.

2/ Si le système possède plusieurs processeurs à mémoire partagée, le masquage des interruptions aura lieu seulement dans le processeur d'origine, les autres continuent leurs accès vers la mémoire partagée.

5.b) Exclusion mutuelle par attente active :

A/ Les variables de verrou :

Considérons une variable verrou unique qui a initialement la valeur 0, le processus doit tester ce verrou avant d'accéder à la section critique :

```
Si verrou = 0 alors
        Verrou ← 1
        Processus entre dans sa SC
Sinon si verrou = 1 alors
        Attente jusqu'à ce que le
        processus en SC le rendre 0
        après la fin de sa SC
```

Entier X = 2000, entier verrou = 0	
Processus 1	**Processus 2**
------ Instructions ------- Si verrou = 0 alors Verrou← 1 X := X+1000 Verrou ← 0 Sinon Attente (tester la valeur de verrou) ------ Instructions -------	------ Instructions ------- Si verrou = 0 alors Verrou← 1 X := X-2000 Verrou ← 0 Sinon Attente (tester la valeur de verrou) ------ Instructions -------

Inconvénient : test simultané de verrou en même temps par deux processus (par exemple) et verrou vaut 0 → les deux changent la valeur à 1 et entrent dans leurs SCs.

B/ L'alternance :

Considérons la variable entière **Tour** prenant de valeur 0 et 1 en alternance entre deux processus :

Entier X = 2000, entier Tour = 0	
Processus 1 ------ Instructions ------- TQ (Tour <> 0) faire Attente (tester la valeur de Tour) X := X+1000 Tour ← 1 ------ Instructions -------	Processus 2 ------ Instructions ------- TQ (Tour <> 1) faire Attente (tester la valeur de Tour) X := X-2000 Tour ← 0 ------ Instructions -------

Inconvénient : Solution non valable s'il y a une grande différence de vitesse entre les processus.

C/ L'instruction Test & Set :

La plupart des processeurs ont une instruction (mécanisme élémentaire câblé) TAS qui permet à la fois de tester et changer la valeur d'un mot mémoire d'une manière exclusive, algorithmiquement on peut écrire TAS comme suit :

```
Debut :
   Bloquer l'accès au mot mémoire M
   Si (M = 0) alors
      M ← 1
      Libérer l'accès au mot mémoire M
      CO ← CO + 2
   Fin si
   Si (M = 1) alors
      Libérer l'accès au mot mémoire M
      CO ← CO + 1
   Fin si
Fin
```

NB : c'est une instruction qui s'exécute d'une manière indivisible

Entier X = 2000, entier P = 0	
Processus 1 ------ Instructions ------- E : TAS(P) Aller à E X := X+1000 P ← 0 ------ Instructions -------	**Processus 2** ------ Instructions ------- E : TAS(P) Aller à E X := X-2000 P ← 0 ------ Instructions -------

5.c) Exclusion mutuelle par attente passive :

A/ Les sémaphores :

Définition :

Un sémaphore est une variable entière constitue la solution utilisée couramment pour restreindre l'accès à des ressources partagées. Le sémaphore a été inventé par Edsger Dijkstra en 1965.

Un sémaphore S est constitué :

- D'un entier e(s).

- D'une file d'attente f(s).
- Deux primitives P(s) et V(s)
  - P = Proberen (puis-je)
  - V = Verhogen (vas-y)

Tel que :

```
P(s)
    e(s) ← e(s) – 1
    si (e(s) < 0) alors
        état(p) ← bloqué
        entrer(p, f(s))
fin si
fin
```

```
V(s)
    e(s) ← e(s) + 1
    si (e(s) <= 0) alors
        sortir (f(s), q)
        état(q, prêt)
        entrer(q, f(p.prêt))
fin si
fin
```

Utilisation pour l'exclusion mutuelle :

Le sémaphore utilisé pour l'exclusion mutuelle est initialisé à la valeur 1, et tous les processus qui utilisent ce sémaphore doivent respecter :

```
Processus
P(s)
    < SC >
V(s)
fin
```

- A tout instant un processus au plus se trouve dans sa section critique.
- Lorsqu'aucun processus ne se trouve dans sa section critique, l'entrée en SC se fait au bout d'un temps fini.
- Les deux primitives d'exécutent d'une manière indivisible.

Exemple :

Entier X = 2000, sémaphore s = 1	
Processus 1	**Processus 2**
------ Instructions -------	------ Instructions -------
P(s)	P(s)
X := X+1000	X := X-2000
V(s)	V(s)
------ Instructions -------	------ Instructions -------

Remarque :

1/ Le choix d'un processus de la file d'attente dépend de la manière de gestion des files dans le système, la description de V n'indique pas comment se fait le choix d'un processus.

2/ P et V sont implémentées comme des primitives systèmes.

3/ Un sémaphore ne peut être initialisé à une valeur négative mais peut devenir après un certain nombre d'exécution de P.

B/ Les moniteurs :

Définition :

Le moniteur est un concept programmé qui permet l'implémentation des SE, il offre des facilités pour assurer l'exclusion mutuelle et la synchronisation entre les processus.

Les moniteurs sont proposés par Hoare et Brinch Hensen.

Caractéristiques :

- Un moniteur est un ensemble de :
  - Variables d'états,
  - Procédures internes,
  - Procédures externes (point d'entrée),
  - Conditions,
  - Primitives de synchronisation.

Les variables d'états sont manipulables par les procédures externes seulement.

- Un seul processus peut être actif dans le moniteur à un instant donné.
- L'exclusion mutuelle est assurée au niveau du moniteur par le compilateur.
- Les instructions qui manipulent les ressources critiques sont mises dans des procédures internes du moniteur.
- Pour bloquer les processus, il faut utiliser des conditions avec deux primitives associées Wait et Signal.

```
Wait(C)
    Etat(p) ← bloqué/    placer le processus dans la file d'attente associée à la condition C
    Entrer(p, f(C)
Fin
```

```
Signal (C)
    Si (f(C) != null) alors
        Sortir(f(C), q)
        Etat(q) ← prêt /  sortir le processus suivant de la file d'attente associée à C
        Entrer(q, f(processus prêt))
    Fin si
Fin
```

- Plusieurs processus peuvent être en attente d'un signal où l'ordonnanceur choisit un.

Exemple :

```
Entier X = 2000,
Moniteur Exemple
    X-libre : condition
    Test : boolean
Procédure ProtégerX()
```

```
    Si Test alors
        Wait(X-libre)
    Fin si
    Test ←vrai
Fin
Procédure LibérerX()
    Test ← faux
    Signal(X-libre)
Fin
Début
    Test ← faux
Fin
```

Processus 1	**Processus 2**
------ Instructions ------- Exemple.ProtégerX() X := X+1000 Exemple.LibérerX() ------ Instructions -------	------ Instructions ------- Exemple.ProtégerX() X := X-2000 Exemple.LibérerX() ------ Instructions -------

Remarque :

1/ Les moniteurs sont un concept programmé, le compilateur doit les connaître, les langages C et Pascal n'ont pas de moniteurs, ils sont prédéfinis dans quelques rares langages comme le Concurrent Euclid et Java.

2/ Brinch Hensen et Hoare définit chacun une approche de moniteurs :

Brinch Hensen	**Hoare**
Quand le processus P1 réveille le processus P2, il sort du moniteur et il termine son exécution en dehors du moniteur en parallèle.	Quand le processus P1 réveille le processus P2, P2 entre dans le moniteur. P1 se met directement dans la file d'attente des processus activateur et attend que le moniteur soit libre pour entrer et continuer son exécution.

III/ Synchronisation :

1/ Définition :

La synchronisation est un terme lié au temps, c'est une définition qui décrit les contraintes par rapport au temps selon lesquelles les instructions d'un processus doivent être exécutées par rapport aux autre processus.

On dit qu'un processus est synchronisé avec d'autres processus s'il a la possibilité de :

- Se bloquer,
- Bloquer un autre processus,
- Réveiller un autre processus.

2/ Mécanismes de synchronisation :

2.a) Mécanisme d'action directe :

Un processus P peut faire passer un autre processus Q à l'état bloqué (Q passe à l'état bloqué dans un point observable) ou le faire passer à l'état prêt (par l'instruction éveiller).

Exemple :

```
P1                P2                P3
    ------            ------            ------
    Bloquer(P3)       Éveiller(P3)      ------
    ------            ------
```

2.b) Mécanisme d'action indirecte :

A/ Les évènements :

Les évènements sont des objets manipulés par les langages de programmation. Un évènement est présenté par un identificateur, créé par une déclaration, il ne peut être manipulé que par certaines opérations particulières.

Un évènement peut être attendu par le ou les processus qui y ont accès, ou bien il peut être déclenché. Il peut être mémorisé ou non mémorisé.

Un évènement mémorisé :

Un évènement peut être représenté par une variable booléenne, à un instant donné la valeur 1 veut dire l'évènement est arrivé, la valeur 0 veut dire l'évènement n'est pas arrivé.

- Un processus se bloque si et si seulement l'évènement n'est pas arrivé.
- Le déclenchement d'un évènement débloque les processus qui l'attendent.
- L'évènement mémorisé peut être remit à 0 :
  - Soit explicitement par un processus au moyen d'une primitive spécifique.
  - Soit implicitement dès qu'un processus qui l'attend s'active.

Un évènement non mémorisé :

- Lorsqu'il n'y a pas de mémorisation, un évènement émit où aucun processus ne l'attend sera perdu.
- Par contre, si plusieurs processus sont bloqués en attente de cet évènement au moment quand il se produit, ces processus seront rendus actifs.

Exemple d'utilisation : Dans les langages conçus pour le contrôle des processus industriels.

B/ Les sémaphores :

Ce mécanisme est utilisé aussi pour résoudre des problèmes généraux de la synchronisation.

- Un signal d'activation est envoyé par V,
- Le signal est attendu par P,

- Le sémaphore utilisé pour la synchronisation est initialisé à 0,
- Il est utilisé comme suit :

Sémaphore s = 0	
Processus 1 ------ Instructions ------- P(s) • Si P1 s'exécute le premier il sera bloqué, P2 le réveille. ------ Instructions -------	**Processus 2** ------ Instructions ------- V(s) • Si P2 s'exécute le premier il réveille P1, c'est un signal mémorisé jusqu'à l'exécution de P1 ------ Instructions -------

C/ Les moniteurs :

On utilise les moniteurs de la même manière que dans l'exclusion mutuelle avec les mêmes définitions.

IV/ La communication inter-processus :

Il est nécessaire à des processus de communiquer entre eux, par exemple, lorsqu'un compilateur exécute un assembleur, il a besoin de lui communiquer le code assemblé d'un programme. La communication c'est l'activité de transfert de données entre les processus par différents moyens.

1/ Communication directe :

Dans cette forme de communication les processus qui s'échangent de l'information utilisent des primitives dans lesquelles figure de façon explicite le nom de destinataire ou de l'émetteur.

P1	**P2**
SEND(P2, msg) Envoyer au processus P1 un message	RECEIVE(P1, msg) Recevoir un message du processus P1

C'est une liaison propre seulement à deux processus.

2/ Communication indirecte :

2.a) Les variables communes :

Les problèmes les plus généraux de la communication entre processus peuvent être résolus par un ensemble de variables communes accessibles à tous les processus qui les utilisent.

Puisque l'accès simultané de plusieurs processus à de telles variables pose des problèmes de cohérence (Exclusion mutuelle) → il faut inclure tout accès à des données communes dans une section critique.

2.b) Boites aux lettres :

Une boite aux lettres est un objet dans lequel les processus peuvent déposer ou prélever des messages :

- Une boite aux lettres est connue par un nom unique.
- Deux processus communiquent via une boite commune.
- Les primitives qui utilisent les boites aux lettres ont la forme suivante :

SEND(A, msg)

RECEIVE(A, msg) / A est le nom de la boite.

- C'est une liaison qui peut être associée au plus de deux processus.
- Une boite aux lettres peut être un objet local aux processus ou appartenir au système.
  - Dans le premier cas, ce processus est propriétaire de cette boite et les autres sont des utilisateurs dont ils peuvent déposer des messages.
  - Dans le deuxième cas, ces boites aux lettres ne sont associées à aucun processus, et le système responsable de sa gestion (créer une boite, déposer et prélever les messages, détruire une boite).

V/ Exemples de problèmes de coopération :

1/ Le modèle de producteur et de consommateur :

Le schéma connu sous le nom « Producteurs/Consommateurs » permet de présenter les principaux problèmes de la communication entre processus par accès à des variables communes avec synchronisation :

- L'information communiquée est constituée des messages de tailles fixes.
- Aucune hypothèse n'est faite sur les vitesses des deux processus.
- La zone mémoire commune (Tampon) a une capacité fixe de messages.
- L'exclusion mutuelle est faite au niveau des messages :
  - Le consommateur ne peut prélever un message que le producteur est en train de ranger.
  - Le producteur ne peut déposer un message dans le tampon si celui-ci est plein.
- Le consommateur doit prélever un message une seule fois.
- Si le tampon est vide le consommateur réveille le producteur et l'inverse si le tampon est plein.
- Le producteur s'exclue mutuellement avec les autres producteurs.
- Le consommateur s'exclue mutuellement avec les autres consommateurs.

Producteur	**Consommateur**
Produire(msg)	Prélever(tampon, msg)
Déposer(msg, tampon)	Consommer(msg)

2/ Le modèle Lecteurs et rédacteurs :

Le modèle de « lecteurs/rédacteurs » constitue un autre problème classique qui modélise les accès à une base de données où plusieurs processus tentent de lire et d'écrire des informations.

- Plusieurs lecteurs peuvent lire la base de données simultanément.
- Si un rédacteur est en train de modifier le contenu de la base de données aucun autre processus lecteur ou rédacteur ne peut y accéder.
- Le rédacteur ne travaille pas tant qu'il y a des lecteurs.
- Mettre en attente tous les lecteurs ayant adressé leurs demandes de lecture après celle du rédacteur.

VI/ L'interblocage (Deadlock) :

1/ **Problématique :**

On se trouve souvent dans les systèmes d'exploitation devant cette situation :

- Tous les flots d'exécution sont bloqués.
- Le déblocage d'un flot ne peut être provoqué que par l'exécution d'un autre flot.

C'est une situation résultat de l'exécution de plusieurs processus concurrents et synchronisés ainsi qui partagent des ressources dans le système, cette situation est connue sous le nom « Interblocage ou Deadlock ».

2/ **Caractéristiques des ressources :**

La ressource qui est un objet dont le processus a besoin pour se progresser dans son exécution peut avoir comme caractéristiques :

2.a) Physique ou logique : une ressource peut être physique (existence réelle), par exemple : Disque dur, Processeur, interfaces, bus, … Comme elle peut être logique (existence abstraite), par exemple : fichiers, verrous, page mémoire, …

2.b) Préemptible ou non préemptible :

- On dit qu'une ressource est préemptible (retirable) si elle peut être retirée d'un processus sans effets négatifs, par exemple : processeur, mémoire, contrôleur disque, …
- Une ressource est dite non préemptible si elle ne peut être retirée d'un processus, puisqu'elle peut causer des problèmes, par exemple : graveur, fichier, bloc du disque, imprimante, …

2.c) Les exemplaires : une ressource peut avoir plusieurs exemplaires.

2.d) L'ordre d'attribution : l'ordre dans lequel les ressources sont attribuées est important.

2.e) Ressources séquentiellement réutilisables ou consommables :

- Une ressource est dite séquentiellement réutilisable si après son utilisation par un processus P1 elle sera restituer (libérer) pour être utilisée par un autre processus P2.
- Une ressource est dite consommable si après son allocation à un processus P elle sera consommée d'une manière à ne pas être libérer et restituer au système.

2.f) Critique ou partageable :

La ressource peut être à un seul point d'accès par un seul processus à la fois → critique, comme elle peut être à plusieurs points d'accès par plusieurs processus simultanément → partageable.

2.g) Séquences d'utilisation :

La séquence d'évènements nécessaires pour utiliser une ressource par un processus est :

- Demande de la ressource.
- Allocation et utilisation de la ressource.
- Libération de la ressource.

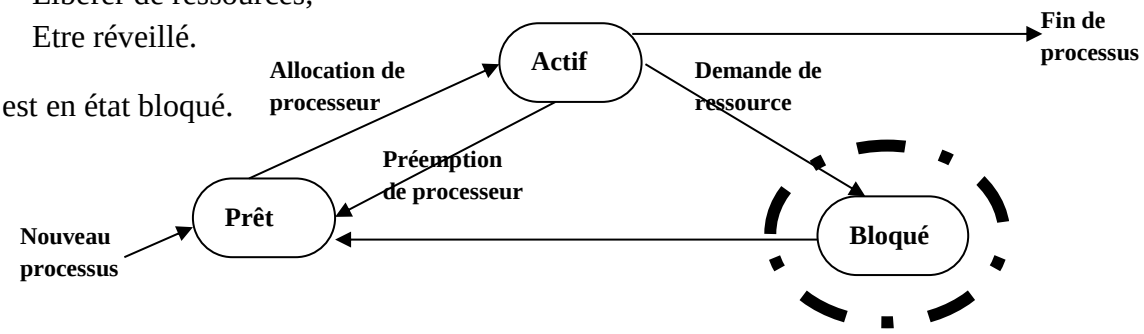
3/ <u>Définition de l'interblocage</u> :

Un ensemble de processus est en interblocage si chacun d'eux attend un évènement qui ne peut être provoqué que par un autre processus de l'ensemble → un évènement attendu est habituellement la libération d'une ressource.

Alors, ces processus se bloquent mutuellement où aucun de ces processus ne peut :

- S'exécuter,
- Libérer de ressources,
- Etre réveillé.

Il est en état bloqué.



<u>Exemples</u> :

1/ R1, R2 des ressources à un seul exemplaire chacune

- Processus P1 détient R1 et demande R2,
- Processus P2 détient R2 et demande R1 → interblocage

2/ Deux sémaphores S1, S2 = 1

Processus 1	**Processus 2**
P(S1) **1**	P(S2) **2**
P(S2) **3**	P(S1) **4**
---	---
V(S2)	V(S1)
V(S1)	V(S2)

L'ordre de l'exécution 1, 2, 3, 4 conduit à un interblocage.

4/ <u>Les conditions de l'interblocage</u> :

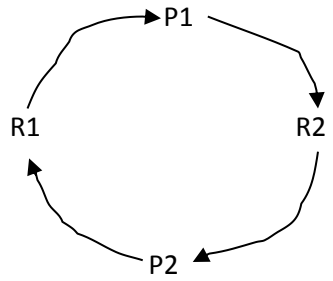
Pour qu'un interblocage ait lieu selon Coffman, il faut vérifier :

4.a) <u>Condition de l'exclusion mutuelle</u> : chaque ressource est disponible, soit attribuée à un seul processus (non partageable).

4.b) <u>La détention et l'attente</u> : les processus ayant déjà obtenu des ressources peuvent en demander d'autres.

4.c) <u>Pas de préemption</u> : les ressources déjà détenues ne peuvent être retirées de force à un processus.

4.d) <u>L'attente circulaire</u> : il doit y avoir un cycle d'au moins deux processus → chacun attend une ressource détenue par un autre processus du cycle.



Remarque :

- En présence des trois premières conditions, une attente circulaire est un interblocage.
- Les trois premières conditions n'impliquent pas nécessairement un interblocage, car l'attente circulaire pourrait ne pas se vérifier.

5/ Graphe d'allocation de ressources :

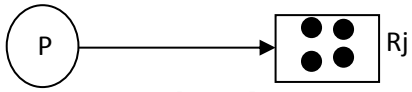
Le graphe G (graphe orienté) est constitué d'un ensemble de sommets V et d'arcs E, tel que :

- V est composé de :
  - P = {P1, P2, …, Pn} l'ensemble de tous les processus dans le système.
  - R = {R1, R2, …, R_m} l'ensemble de tous les types de ressources dans le système
- E peut être :
  - Arc de requête ou demande $P_i \rightarrow R_k$
  - Arc d'allocation $R_i \rightarrow P_k$.

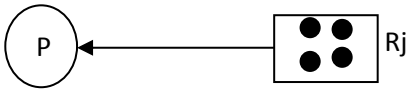
On désigne un processus avec la notation ◯ et la ressource avec □

Exemple :

- Pi a besoin pour se terminer l'un des exemplaires de Rj :



- Pi a requis un exemplaire de Rj :



Exemples de graphes :

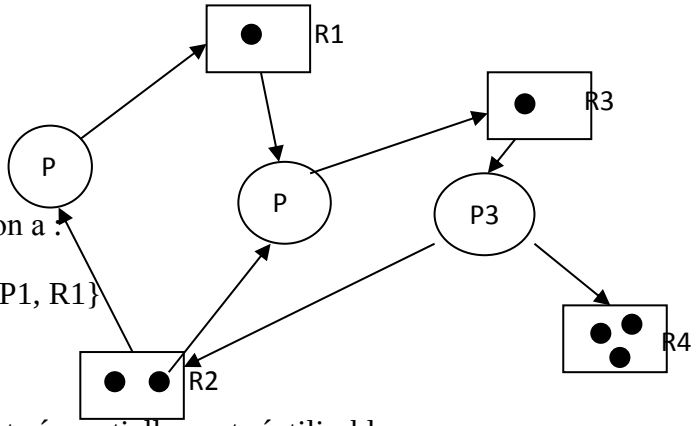
P = {P1, P2, P3}

R = {R1, R2, R3, R4}

Il y a un interblocage puisqu'on a :

Cycle1={R1, P2, R3, P3, R2, P1, R1}

Cycle2={R3, P3, R2, P2, R3}



Ces ressources sont critiques et séquentiellement réutilisables

6/ Le traitement de l'interblocage :

Puisque les quatre conditions sont nécessaires pour dire qu'il y a un état d'interblocage, il suffit de faire en sorte que l'une de ces conditions ne soit remplisse pour **prévenir** l'interblocage. Comme il faut aussi pouvoir le **détecter et reprendre** l'état stable du système. Autre solution c'est d'**ignorer** la présence de l'interblocage.

6.a) La prévention de l'interblocage :

La prévention de l'interblocage c'est de faire perturber l'une des conditions nécessaires pour que l'interblocage se produise.

A/ *La condition de l'exclusion mutuelle* :

C'est-à-dire rendre une ressource non partageable une ressource partageable → c'est presque impossible à changer, au lieu d'avoir des interblocages nous allons avoir des concurrences critiques. Des fois les ressources peuvent être logique ou la ressource peut avoir des copies, par exemple l'imprimante dans Windows, au lieu d'écrire directement sur l'imprimante on écrit sur un spooler et les jobs sortent à leurs tours.

Spool : est un buffer de requêtes désynchronisées par un processus Démon.

B/ *La condition de détention et attente* :

On évite l'interblocage si on peut empêcher les processus qui détiennent déjà des ressources de détenir d'autres ressources → on impose aux processus de demander les ressources dont ils ont besoin avant de commencer leurs exécutions.

- Si les ressources demandées par un processus sont disponibles, elles lui sont allouées.
- Si une ou plusieurs ressources demandées sont déjà attribuées alors aucune ressource ne lui est allouée → les processus attendent la libération de toutes les ressources demandées.

Mais :

- Beaucoup de processus ne connaissent pas à l'avance les ressources nécessaires.
- Les ressources ne sont pas attribuées d'une manière optimale (utilisation pour une grande durée d'une ressource → résulte la famine pour certains processus).

Remarque : une approche peut être adoptée pour cette condition consiste à imposer aux processus qui demandent une nouvelle ressource de libérer les ressources détenues

C/ *La condition de non préemption* :

C'est de retirer à force les ressources occupées par un processus (en attente d'autres ressources), mais ça apparait impossible de le réaliser.

Exemple : Retirer une imprimante d'un processus au milieu d'impression.

D/ *La condition d'attente circulaire* :

1ère approche : limiter le nombre de ressources détenues par un processus à un instant donné, s'il a besoin d'une nouvelle, il faut libérer une autre.

2ème approche : numéroter toutes les ressources d'une manière croissante, si un processus demande une ressource de numéro supérieur de ce qu'il détient on lui donne cette ressource, sinon si le nombre est inférieur on refuse sa demande → il faut demander une ressource avec un numéro supérieur.

6.b) L'ignorance de l'interblocage :

Nommée aussi politique de l'Autruche où les ingénieurs ne choisissent pas de réduire les performances ou la facilité d'utilisation du système pour éliminer l'interblocage. Donc ils ignorent complètement l'interblocage.

6.c) La détection et la reprise :

Le système ne cherche pas à empêcher les interblocages mais il les laisse se produire puis tente de les détecter et reprendre l'état du système.

A/ La détection de l'interblocage :

On dit que l'état du système est interbloqué si dans la progression des exécutions des processus on passe par plusieurs états $état_1$ → $état_2$ → … → $état_n$ où après le dernier état on ne peut pas passer vers un autre alors que cet état contient des processus qui ne peuvent pas se progresser dans leurs exécutions.

On peut suivre ces états par la réduction du graphe de processus-ressource ou la formalisation mathématique de l'interblocage.

La réduction de graphe :

On dit qu'un graphe est réduit par un processus Pi qui n'est pas bloqué et qui n'est pas isolé si ce processus peut acquérir toutes les ressources puis les libérer toutes (suppression de tous les arcs allant de et vers Pi). Cette méthode est utilisée pour la détection de l'interblocage dans le cas des ressources à un seul exemplaire.

- Un graphe est dit irréductible s'il ne peut être réduit dans un état n d'évolution des processus ainsi la présence d'un cycle.
- Un graphe est réductible s'il existe une séquence d'état dont le dernier état ne laisse aucun arc.

Algorithme de réduction de graphe :

1- Pour chaque nœud N dans le graphe, suivre les cinq étapes ci-après, avec N comme point de départ.
2- Initialiser une liste vide et désigner tous les arcs comme non marqués.
3- Ajouter le nœud en cours à la fin de la liste et vérifier que le nœud apparaît deux fois dans la liste, si oui, le graphe contient un cycle et l'algorithme s'arrête.

4- Pour un nœud donné, voir s'il y a des arcs non marqués sortants. Dans ce cas aller à l'étape 5, sinon aller à l'étape 6.
5- Choisir au hasard un arc sortant non marqué et marquer le. Suivre jusqu'au prochain nouveau nœud en cours et aller à l'étape 3.
6- Si ce nœud est le nœud initial, le graphe ne comprend pas de cycle et l'algorithme se termine. Dans le cas contraire, on se trouve dans une impasse, supprimer l'arc et revenir au nœud précédent, c'est-à-dire à celui qui était actif juste avant celui-ci, mettre le comme nœud en cours et retourner à l'étape 3.

La représentation mathématique de l'interblocage :

On définit que le système est un ensemble de processus P = {P_1, P_2, …, P_n} et un ensemble de ressources R = {R_1, R_2, …, R_m}

- L'état initial du système est décrit par le vecteur X donnant le nombre total de ressources existantes : X = (X_1, X_2, …, X_m) → constant par hypothèse.
- A l'instant t, l'état du système est définit par :
  - Matrice de toutes les allocations A(t) = $\begin{pmatrix} a_{11(t)} & \cdots & a_{1m(t)} \\ \vdots & \ddots & \vdots \\ a_{n1(t)} & \cdots & a_{nm(t)} \end{pmatrix}$

Où P1 détient (a_{11}, a_{12}, .., a_{1m}) de ressources → $a_{ij(t)}$ est le nombre de ressources de la classe R_j allouées au processus P_i.

  - Matrice de toutes les demandes D(t) = $\begin{pmatrix} d_{11(t)} & \cdots & d_{1m(t)} \\ \vdots & \ddots & \vdots \\ d_{n1(t)} & \cdots & d_{nm(t)} \end{pmatrix}$

Où P1 demande (d_{11}, d_{12}, .., d_{1m}) de ressources → $d_{ij(t)}$ est le nombre de ressources de la classe R_j demandées par le processus P_i.

  - R(t) définit le nombre de ressources restantes à l'instant t R(t) = X - $\sum_{i=1}^{n} \sum_{j=1}^{m} a_{ij(t)}$
- L'état réalisable du système, si et seulement si les trois conditions suivantes sont vérifiées :
  - Di(t) ≤ X, $\forall\, i \in [1, n]$ les demandes ne dépassent pas le disponible.
  - Ai(t) ≤ Di(t), $\forall\, i \in [1, n]$ les allocations ne dépassent pas les demandes.
  - $\sum_{j=1}^{m} a_{j(t)} \leq X$
- L'état sain du système, si on arrive à exécuter tous les processus, c'est-à-dire il faut montrer une suite saine complète de processus tel que pour chaque processus i on vérifie la relation :

$$D_{i(t)} - A_{i(t)} \leq R(t) + \sum A_{j(t)} \ \ tel\ que\ j < i$$

- L'état d'interblocage du système, s'il n'existe aucune suite saine complète, alors le système est en interblocage dans l'état t, c'est-à-dire, il existe au moins deux processus dont on ne peut pas vérifier la relation précédente :

$$D_{i(t)} - A_{i(t)} > R(t) + \sum A_{j(t)} \ \ tel\ que\ j < i$$

- Les systèmes d'annonces,

Dans un système d'annonces, soit C la matrice des annonces où :

$C_{max} = \begin{pmatrix} c_{11(t)} & \cdots & c_{1m(t)} \\ \vdots & \ddots & \vdots \\ c_{n1(t)} & \cdots & c_{nm(t)} \end{pmatrix}$ → c_{ij} représente le nombre maximal de ressources de la classe R_j que pourra utiliser le processus P_i.

On impose un état réalisable par la vérification des nouvelles conditions :

- $Ci \leq X$, quel que soit i
- $Ai(t) \leq Ci$, quel que soit i, t
- $Di(t) \leq Ci$, quel que soit i, t

B/ La reprise de l'état stable du système :

Que faut-il faire lorsque l'algorithme de détection a détecté un interblocage ? Plusieurs méthodes sont adoptées pour la reprise après l'interblocage.

La reprise au moyen de préemption :

Dans certains cas il est possible de retirer temporairement une ressource à un processus pour l'attribuer à un autre. Dans de nombreux cas l'intervention manuelle peut être requise.

Exemple : Retirer l'imprimante de son processus actuel par l'intervention d'un utilisateur qui prend tous les papiers de l'imprimante, il suspend le processus actuel, il démarre un autre processus en rangeant les feuilles dans l'imprimante, après sa fin il redémarre le premier.

Cette solution dépend de la nature de la ressource et de la nature de processus.

La reprise au moyen du Rollback :

Les concepteurs posent régulièrement des points de reprises sur les processus → l'état du processus est sauvegardé et il pourra être restauré ultérieurement.

Les concepteurs créent plusieurs points de reprise d'un processus chacun sauvegardé indépendamment (Exp : Office). Lorsqu'un interblocage se produit, il est facile de déterminer les ressources demandées, le processus qui détient une ressource est restauré au point avant cette allocation et tout le travail effectué après ce point sera perdu. La ressource ainsi est attribuée à un autre processus bloqué.

La reprise au moyen de suppression de processus :

La méthode simple pour supprimer l'interblocage c'est de tuer un ou plusieurs processus, le processus choisi c'est un processus du cycle dont les autres pourront poursuivre leurs exécutions.

Une autre méthode consiste à tuer un processus hors du cycle de manière de libérer les ressources. Il est préférable de tuer un processus qui peut être re-exécuté sans effets secondaires.

C/ Les états sûrs et non sûrs :

- On dit que le système est en *état sûr* (état fiable) si elle existe une séquence d'allocation qui permet à tous les processus de s'exécuter jusqu'au bout dans le pire des cas où tous les processus demandent toutes les ressources annoncées dans la matrice C_{max}.

$$C_i - A_{i(t)} \leq R(t) + \sum A_{j(t)} \ tel\ que\ j < i$$

  - Si on trouve cette suite → cet état du système va conduire à un état sain.
- On dit que le système est en état non sûr si on ne trouve pas une séquence d'allocation qui permet à tous les processus de s'exécuter jusqu'au bout
  - S'il existe au moins deux processus qui ne vérifient la relation précédente → alors on prévient un interblocage dans les états suivants (prévention dynamique).
- Il convient de noter qu'un état non sûr ne conduit pas nécessairement à un interblocage.
- La différence entre un état sûr et un état non sûr tient à ce que dans un état sûr, le système peut garantir que tous les processus s'achèvent.

Dans ce cas on peut éviter la création d'une situation d'interblocage en utilisant **l'algorithme de Banquier** :

- Modélise le principe de travail d'un agent bancaire pour satisfaire les demandes des clients qui veulent avoir de l'argent.
- Il faut que les processus annoncent leurs besoins avant de commencer l'exécution.
- L'algorithme vérifie pour chaque processus ses demandes actuelles
  - Si la satisfaction de ces demandes conduit à un état d'interblocage, elles seront refusées et ce processus se mis en attente jusqu'à la libération des ressources demandées.
  - Si la satisfaction de ces demandes conduit à un état sain du système, l'algorithme attribue ces ressources au processus pour qu'il termine son exécution.
- S'il y a plusieurs demandes simultanées, l'algorithme choisis de satisfaire le processus qui conduit à un état sain et le reste sont mis en attente.