Chapitre 4

Gestion des processus

4.1- Introduction

Le processeur central est considéré comme la ressource la plus importante dans une machine, donc il convient à bien gérer ce dernier afin de le rendre plus productif. Pour atteindre un cet objectif, les concepteurs des systèmes d'exploitation délèguent l'allocation du processeur à un module du système d'exploitation en l'occurrence le Dispatcheur. Ce Dispatcheur est exécuté chaque fois que le processus en cours d'exécution se bloque ou se termine, outre ces deux situations il peut être également exécuté en cas de réquisition du processeur si l'ordonnancement prend en considération cette propriété.

4.2 Concept de processus

Un processus est une entité dynamique qui matérialise un programme en cours d'exécution avec ses données, sa pile, son compteur ordinal, son pointeur de pile et les autres valeurs de registres nécessaires à son exécution. Contrairement à un programme (texte exécutable) qui a une existence statique, un processus constitue l'aspect dynamique du programme, c'est à dire un ensemble d'états et de transitions (automate). Le passage d'un état à un autre se réalise en effectuant l'appel système approprie. Les appels système relatifs aux processus sont ceux qui permettent généralement :

- Création d'un processus (fils) par un processus actif (le père)
- Destruction d'un processus
- Mise en attente, et réveil d'un processus
- Suspension et reprise d'un processus
- Attendre la fin d'un processus fils
- Echanges de messages avec d'autres processus
- Modifier la priorité d'un processus

4.2.1- Etats d'un processus

A l'instar de toute entité dynamique, un processus peut changer d'état au cours de son exécution. La transition d'un état à un autre est effectue selon le diagramme de transition ci dessous. :

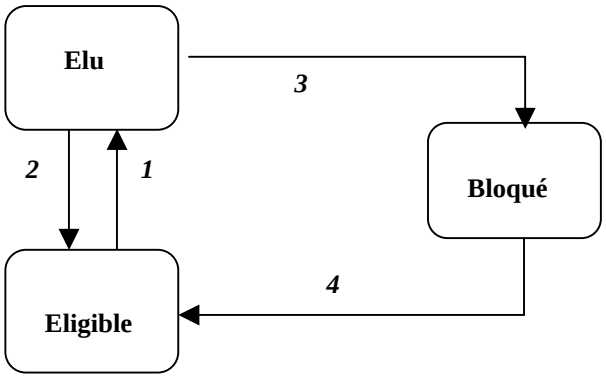


Figure 4.1 : Diagramme de transition d'un processus .

Notons que nous avons omis les états Création et Terminaison de processus puisqu'ils n'interviennent pas dans cette étude.

Sémantique des états

1. Elu (en Exécution) : si le processus est en cours d'exécution.
2. Eligible (Prêt) : Si le processus dispose de toutes les ressources nécessaires à son exécution à l'exception du processeur.
3. Bloqué : Si le processus attend qu'un événement se produit pour progresser.

Sémantique des Transitions

(1) : Allocation du processeur au processus sélectionné
(2) : Réquisition du processeur après expiration de la tranche du temps par exemple
(3) : Blocage du processus élu dans l'attente d'un événement (E/S ou autres)
(4) : Réveil du processus bloqué après disponibilité de l'événement bloquant (Fin E/S, etc…)

Nous soulignons que les transitions (1) et (2) sont liées à l'allocation du processeur, alors que les transitions (3) et (4) résultent de la synchronisation des processus.

4.2.2- Propriété fondamentale d'un processus

L'exécution d'un processus peut être vue comme une séquence de phases. Chaque phase comprend deux cycles : un cycle d'exécution (ou calcul) réalisé par le processeur et un cycle d'entrée sortie assuré par le canal . La dernière phase de tout processus doit comprendre obligatoirement un seul cycle dans lequel sera exécuté la requête informant le système d'exploitation sur la terminaison du processus. Cet appel permet au système de restituer les ressources .utilisées par le processus qui vient de terminé.

4.3- Allocation du processeur

Chaque fois, que le processeur devient inactif, le système d'exploitation doit sélectionner un processus de la file d'attente des processus prêts, et lui passe le contrôle. D'une manière plus concrète, cette tâche est prise en charge par deux routines système en l'occurrence le ***Dispatcheur*** et le ***Scheduleur***.

4.3.1- Dispatcheur

Le Dispatcheur est la routine système, qui s'occupe de l'allocation du processeur à un processus sélectionné par le scheduleur du processeur. Les tâches confient à cette routine sont :

- Commutation de contexte, c'est à dire sauvegarde du contexte du processus qui doit relâcher le processeur et restauration du contexte du processus qui aura le prochain cycle processeur.
- Commutation vers le mode utilisateur, le dispatcheur s'exécute en mode maître et le processus à lancer est souvent un processus utilisateur qui doit être exécuté en mode esclave. (Bien sûr si la machine dispose des deux modes)
- Branchement au bon emplacement dans le processus utilisateur pour le faire démarrer.

Notons que le contexte d'un processus dépend du système, mais dans tous les cas c'est un bloc de contrôle de processus (BCP) qui contient toute information nécessaire à la reprise d'un processus interrompu. Les informations que nous retrouverons dans la plupart des systèmes sont celles données dans l'exemple ci dessous. Les BCP sont rangés dans une table (table des processus) qui se trouve dans l'espace mémoire du système.

Exemple 4.1 Contexte de processus

- une copie du PSW au moment de la dernière interruption du processus
- l'état du processus : prêt à être exécuté, en attente, suspendu, ...
- des informations sur les ressources utilisées
  - o mémoire principale
  - o temps d'exécution
  - o périphériques d'E/S en attente
  - o files d'attente dans lesquelles le processus est inclus, etc...
- Enfin toute information nécessaire pour assurer la reprise du processus en cas d'interruption

4.3.2- Scheduleur

Le Scheduleur du processeur est également une autre routine système qui s'occupe de la sélection du processus qui aura le prochain cycle processeur, à partir de la file d'attente des processus prêts. Concrètement, cette routine n'est rien d'autre que l'implémentation d'un algorithme d'ordonnancement de processus. Nous distinguons plusieurs algorithmes d'ordonnancement, les plus répandus sont :

- Ordonnancement Premier Arrivé Premier Servi
- Ordonnancement du plus court d'abord
- Ordonnancement circulaire : Tourniquet
- Ordonnancement circulaire à plusieurs niveaux
- Ordonnancement avec priorité

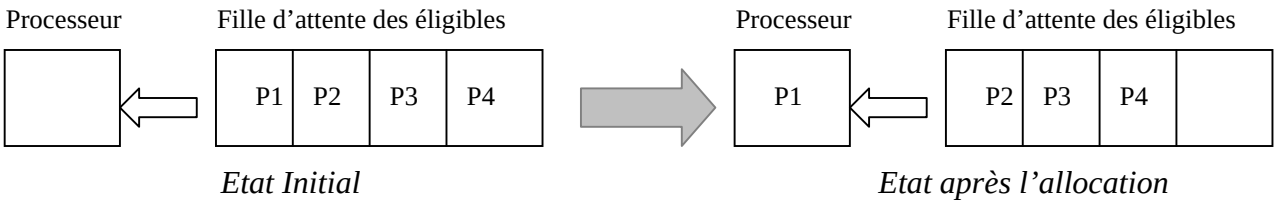
Notons que, le scheduleur du processeur (court terme) opère sur un ensemble de processus présents en mémoire. Dans les machines ayant des contraintes mémoire, c'est à dire la mémoire centrale ne peut pas loger tous les processus prêts. Les systèmes d'exploitation de ces machines sont dotés d'un autre Scheduleur appelé Scheduleur de travail (Scheduleur long terme). Dans cette technique que nous appelons souvent ordonnancement à deux niveaux, les processus sont stockés sur un disque, et le Scheduleur de travail sélectionne le sous ensemble de processus (dépend de la taille de la mémoire et celle des processus) qui vont être chargés en mémoire. En suite ce même Scheduleur retire périodiquement de la mémoire les processus qui sont restés assez longtemps et les remplace par des processus qui sont sur le disque depuis trop de temps. A l'instar du scheduleur court terme, le scheduleur de travail utilise un des algorithmes d'ordonnancement ci dessous.

4.3.2.1- Ordonnancement Premier Arrivé Premier Servi : FCFS

Dans cet algorithme, les processus sont rangés dans la file d'attente des processus prêts selon leur ordre d'arrivé. Les règles régissant cet ordonnancement sont :

1. Quand un processus est prêt à s'exécuter, il est mis en queue de la file d'attente des processus prêts.
2. Quand le processeur devient libre, il est alloué au processus se trouvant en tête de file d'attente des processus prêts.
3. Le processus élu relâche le processeur s'il se termine ou s'il demande une entrée sortie (pas de réquisition)

Exemple 4.2



Etat Initial *Etat après l'allocation*

Si les temps d'exécution de ces processus sont respectivement 8, 4, 4 et 5 alors le temps d'attente moyen (des usagers) Tm = (8+12+16+21)/4 =57/4 =14.25 ms
Si on s'intéresse au temps d'attente moyen au niveau de la file d'attente des processus prêts Tm dans ce cas est égale : Tm = (0+8+12+16)/4 = 36/4 =9.0 ms

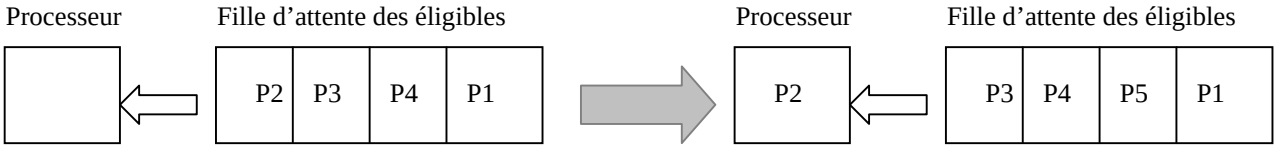
Caractéristiques de Scheduleur
Les principales caractéristiques de ce scheduleur sont :

- Ce scheduleur est sans réquisition
- Ce scheduleur est non adapté à un système temps partagé car dans un système pareil, chaque utilisateur obtienne le processeur à des intervalles régulieres.

4.3.2.2- Ordonnancement du plus court d'abord : SJF (shortest Job First)
L'algorithme SJF (Shortest Job First) trouve son origine dans les systèmes traitement par lot dans lesquels le temps d'exécution des différents processus est connu à l'avance (par exemple un processus qui calcule tous les mois la paie de 100 personnes son temps d'exécution doit être connu par les personnes du centre de calcul).
Dans cet algorithme les différents processus sont rangés dans la file d'attente des processus prêts selon un ordre croissant de leur temps d'exécution (cycle de calcul). Les règles régissant cet ordonnancement sont :

1. Quand un processus est prêt à s'exécuter, il est inséré dans la file d'attente des processus prêts à sa position approprie.
2. Quand le processeur devient libre, il est assigné au processus se trouvant en tête de la file d'attente des processus prêts (ce processus possède le plus petit cycle processeur.). Si deux processus ont la même longueur de cycle, on applique dans ce cas l'algorithme FCFS.
3. si le système ne met pas en œuvre la réquisition, le processus élu relâche le processeur s'il se termine ou s'il demande une entrée sortie. Dans le cas contraire, le processus élu perd le processeur également. quand un processus ayant un cycle d'exécution inférieur au temps processeur restant du processus élu, vient d'entrer dans la file d'attente des prêts. Le processus élu dans ce cas sera mis dans la file d'attente des éligibles, et le processeur est alloué au processus qui vient d'entrer.

Exemple 4.3



Etat Initial *Etat après l'allocation*

Dans ce cas, le temps d'attente moyen (des usagers) Tm=(4+8+13+21)/4 =46/4 = 11.5ms
Si on s'intéresse au temps d'attente moyen au niveau de la file d'attente des processus prêts Tm dans ce cas est égale : Tm = (0+4+8+13)/4 = 25/4 =6.25 ms

Caractéristiques de Scheduleur
Les principales caractéristiques de ce scheduleur sont :

- Ce scheduleur peut être avec ou sans réquisition
- Ce scheduleur est souvent utilisé comme scheduleur long terme
- L'usage de ce scheduleur comme scheduleur de processeur est difficile à implémenter car il n'existe aucune manière pour connaître le cycle suivant du processeur. Cependant, certaines réalisations du SJF comme scheduleur du processeur consistent à

calculer une approximation de la valeur du cycle à l'aide de la formule récurrente suivante :

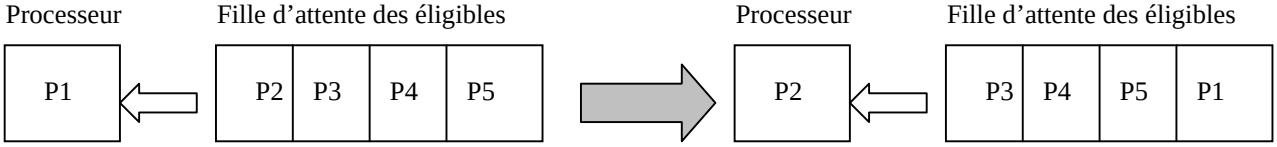
$T_{n+1} = \alpha T_{n-1} + (1-\alpha)T_n$ ou T_n est la longueur du n ^{ieme} cycle d'exécution et $0 <= \alpha <= 1$

4.3.2.3- Ordonnancement circulaire : Le Tourniquet (Round Robin)

Dans cet algorithme les processus sont rangés dans une file d'attente des éligibles, le processeur est alloué successivement aux différents processus pour une tranche de temps fixe Q appelé Quantum. Cet Ordonnancement est régit par les règles suivantes :

1. Un processus qui rentre dans l'état éligible est mis en queue de la file d'attente des prêts.
2. Si un processus élu se termine ou se bloque (pour des raisons de synchronisation par exemple) avant de consommer son quantum de temps, le processeur est immédiatement alloué au prochain processus se trouvant en tête de la file d'attente des prêts.
3. Si le processus élu continue de s'exécuter au bout de son quantum, dans ce cas le processus sera interrompu et mis en queue de la file d'attente des prêts et le processeur est réquisitionné pour être réalloué au prochain processus en tête de cette même file d'attente.

Exemple 4.4



P1 est en possession du processeur Après Q on aura l'état suivant

Caractéristiques de Scheduleur

Les principales caractéristiques de ce scheduleur sont :

- Ce scheduleur est avec réquisition
- Ce scheduleur est bien adapté aux systèmes temps partagé.
- L'efficacité de ce scheduleur dépend principalement de la valeur du quantum Q car :
  - Le choix d'un Q assez petit augmente le nombre de commutation.
  - Le choix d'un Q assez grand augmente le temps de réponse du système

4.3.2.4- Ordonnancement circulaire à plusieurs niveaux

Dans cet algorithme on dispose de n files d'attente ($F_0,\ldots,F_{n-1}$) pour les processus éligibles, à chaque file est associé un quantum de temps dont la valeur croit avec i ($q_0<q_1<...<qn-1$). L'ordonnancement dans ce cas est régit par les règles suivantes :

1. Tout nouveau processus qui entre dans le système est mis dans la file d'attente F0.
2. Tout processus de la file d'attente Fi ne sera servi que si toutes les files Fj tel que j<i sont vides.
3. Si un processus de la file Fi à consommer son qi sans être terminé, il passe dans la file d'attente suivante Fi+1 sauf celui de la file Fn-1 qui doit retourner dans cette même file.

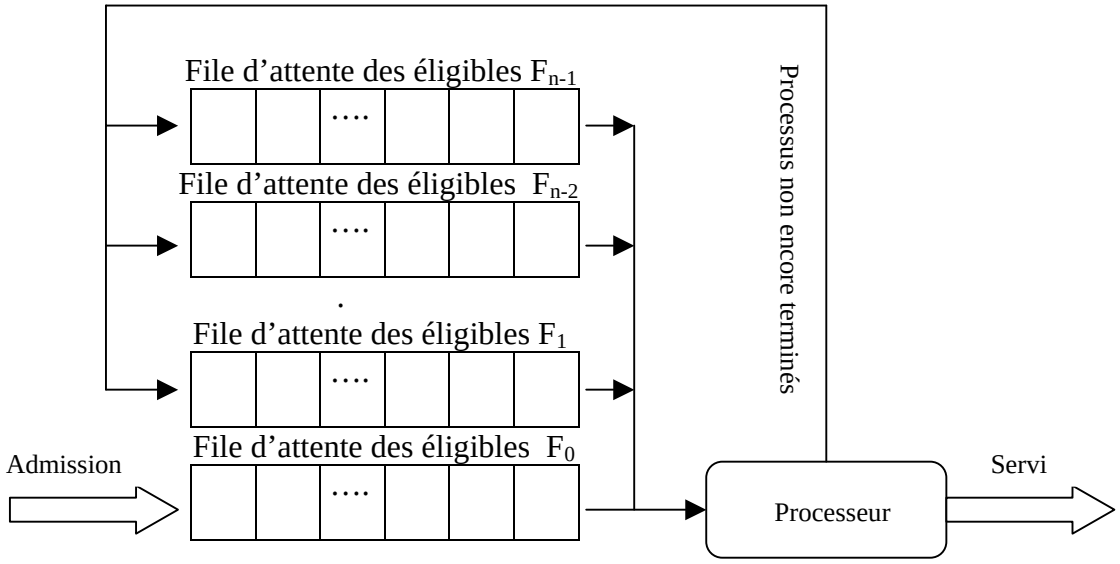


Figure 4.2 : Tourniquet à plusieurs niveaux

Caractéristiques de Scheduleur
Les principales caractéristiques de ce scheduleur sont :

- Ce scheduleur est avec réquisition
- Ce scheduleur favorise les processus courts sans savoir à l'avance combien de temps processeur ceux ci vont utiliser.

4.3.2.5- Ordonnancement Avec priorité
Dans cet algorithme, les processus sont rangés dans la file d'attente des processus prêt par ordre décroissant de priorité. L'ordonnancement dans ce cas est régit par les règles suivantes :

1. Quand un processus est admis par le système, il est inséré dans la file d'attente des processus prêts à sa position approprie (dépend de la valeur de priorité).
2. Quand le processeur devient libre, il est alloué au processus se trouvant en tête de file d'attente des processus prêts (le plus prioritaire).
3. Un processus élu relâche le processeur s'il se termine ou se bloque (E/S ou autre).

Notons que si le système met en œuvre la réquisition, quand un processus de priorité supérieure à celle du processus élu entre dans l'état prêt ; le processus élu sera mis dans la file d'attente des éligibles à la position approprie, et le processeur est alloué au processus qui vient d'entrer.

Caractéristiques de Scheduleur
Les principales caractéristiques de ce scheduleur sont :

- Ce scheduleur peut être avec ou sans réquisition
- Dans ce scheduleur un processus de priorité basse risque de ne pas être servi (problème de famine) d'où la nécessité d'ajuster périodiquement les priorités des processus prêts. L'ajustement consiste à incrémenter graduellement la priorité des processus de la file d'attente des éligibles (par exemple à chaque 15 mn on incrémente d'une unité la priorité des processus prêts)

4.4 - Gestion des activités parallèles
Une application parallèle est définit comme un ensemble de processus séquentiels qui s'exécutent en parallèle (parallélisme réel cas des machines multiprocesseurs ou pseudo parallélisme dans le cas des machines monoprocesseur).Chaque processus de cet ensemble réalise une tâche bien

précise de l'application en question, et il est lié aux autres processus par des relations. Ces relations peuvent être soit des relations d'ordre conflictuel, soit des relations de coopération. Le premier type de relation se présente lorsque les processus mis en jeu partagent des ressources physiques ou logiques. Par contre, le second type de relation intervient lorsque les processus participent à un traitement global. En d'autres termes, ces relations permettent de synchroniser et faire communiquer les processus composant une application parallèle.

4.4.1- Les concepts

4.4.1.1- Synchronisation

La synchronisation est l'activité qui consiste à construire des mécanismes permettant à un processus actif de changer son état. (Actif/Bloqué) ou de changer l'état d'un autre processus.(Bloqué/Actif)

4.4.1.2- Notion de ressources

Les ressources sont des entités nécessaires à l'exécution d'un processus. Ces ressources peuvent être physiques (processeur, mémoire imprimante, etc...) ou logique (données, programmes).On distingue 02 catégories de ressources

1. **Ressource partageable à un point d'accès :** Une ressource est dite partageable à un seul point d'accès ou (critique), si elle peut être attribuée à un seul processus à la fois.

Exemple 4.5

Processeur, Imprimante

2. **Ressource partageable à n points d'accès :** Une ressource est dite partageable à un n points d'accès si elle peut être attribuée à plusieurs (n) processus à la fois.

Exemple 4.6

Blocs mémoire, Programme réentrant, etc…

4.4.2- Les problèmes génériques

Les problèmes génériques sont des problèmes qui se présentent pratiquement dans la plupart des applications parallèles. La connaissance de ces problèmes permet d'aider le programmeur (en s'inspirant des solutions de ces problèmes) à résoudre ses problèmes de synchronisation d'une manière assez élégante. Les problèmes génériques les plus fréquemment rencontrés dans le traitement du parallélisme sont :

- Section critique et exclusion mutuelle
- Schéma des Lecteurs Rédacteurs
- Schéma des Producteurs Consommateurs

1- Problème de la section critique

Ce problème apparu chaque fois qu'un processus veut exécuter une section critique (séquence d'instructions manipulant une ressource critique). En effet, cette section critique contient des objets (variables) partagés entre les différents processus qui utilisent cette ressource critique. Si la gestion de ces objets n'est pas faite d'une manière judicieuse , elle peut entraîner facilement des incohérences.

Exemple 4.7

considération 02 processus P1 et P2 qui veulent mettre à jour une variable commune Réserve. Les 02 processus auront la structure suivante :

Processus P1	*Processus P2*
Debut *.....* *Reserve := Reserve +3* *.....* *Fin*	*Debut* *.....* *Reserve := Reserve +2* *.....* *Fin*

Supposons que la variable Reserve contient une valeur initiale égale à 17
Pour simplifier on suppose que les 02 processus s'exécutent sur deux processeurs différents : (UC1, Acc1) , (UC2, Acc2) et les vitesses des 02 processus sont inconnues
L'instruction d'affectation se traduit en assembleur par les instructions suivantes

1- Load Acc1, Reserve	*a- Load Acc2, Reserve*
2- ADD 3	*b- ADD 2*
3- Store Reserve	*c- Store Reserve*

L'exécution de ces actions dans l'ordre 1a 2b 3c produit un résultat Reserve = 19 au lieu de 22.

Pour résoudre ce problème , il faut que les processus qui partagent un objet commun s'excluent mutuellement pour l'usage de cet objet. D'une manière concret il faut que l'exécution de la section critique soit contrôlée par un protocole formé de deux parties (relatives à l'acquisition et à la libération de la section critique **<SC>**). Dans ce cas, chaque processus aura une structure de la forme :

```
Processus Pi
Debut
.....                    (* Partie non conflictuelle *)
ProtocoleAcquisition
< Section Critique >     (* Partie  conflictuelle *)
ProtocoleLibération
.....                    (* Partie non conflictuelle *)
Fin
```

2- Schéma des Lecteurs Rédacteurs

Dans ce problème, il s'agit d'un ensemble de processus qui partagent une structure de données (Fichier). Ce fichier peut être à tout instant accédé par des processus dits lecteurs qui ne font que la consultation de celui ci, ou par des processus dits rédacteurs qui eux par contre modifient son contenu La cohérence du fichier sera menacée si ce dernier n'est pas manipulé avec précaution selon un protocole bien déterminé. Plusieurs variantes de protocole pour ce schéma ont été développées, la plus simple est celle qui favorise la catégorie des lecteurs et dont la spécification est la suivante :

- Plusieurs lecteurs peuvent lire en parallèle dans le fichier.
- Un rédacteur doit avoir l'accès exclusif au fichier.
- Lorsque le fichier est libre un lecteur et un rédacteur ont la même priorité.
- Si le fichier est déjà accédé en lecture toute nouvelle demande de lecture sera immédiatement honorée, même si des rédacteurs sont en attente.

La structure des processus lecteurs et rédacteurs est la suivante :

Processus Lecteur_i
Debut
.....
(Demander l'autorisation de lire *)*
DemanderLecture
LIRE
(Le processus a fini de lire : le signal*)*
TerminerLecture
.....
Fin

Processus Rédacteur_i
Debut
.....
(Demander l'autorisation d'écrire *)*
DemanderEcriture
ECRIRE
(Le processus a fini d'écrire : le signal*)*
TerminerEcriture
.....
Fin

La solution complète de ce problème réside dans l'écriture des 04 procédures ci dessous :

- DemanderLecture **:** Invoquée par un processus lecteur avant que celui ci commence à lire dans le fichier. Elle autorise le processus appelant à lire si les conditions de lecture sont vérifiées. Dans le cas contraire, le processus appelant sera mis en attente jusqu'à ce que les conditions autorisant celui ci à lire soient vérifiées.
- DemanderEcriture **:** Invoquée par un rédacteur avant de commencer à écrire. Elle autorise le processus à écrire si les conditions d'écriture satisfaites. Dans le cas contraire, il se bloque jusqu'à ce que la demande d'écriture soit acceptée.
- TerminerLecture **:** Invoquée par un processus lecteur dès qu'il a fini de lire. Le dernier lecteur ayant exécuté cette procédure passe le contrôle du fichier à un processus rédacteur en attente.
- TerminerEcriture : Invoquée par un rédacteur pour signaler qu'il a fini d'écrire et passe le contrôle du fichier à un ou plusieurs autres processus.

3- Schéma des Producteurs consommateurs

Ce schéma traduit un modèle de communication couramment rencontré dans le traitement parallèle. Dans ce schéma un processus : le producteur produit des messages et les dépose dans un tampon pour être retirés et traités par un autre processus dit le consommateur. Le protocole qui implémente ce modèle doit prendre en compte les contraintes suivantes :

- Le tampon est composé de N cases, chacune d'elle peut contenir un message.
- Aucune hypothèse n'est faite sur les vitesses des 02 processus.
- Les messages ne doivent pas être perdus, si le tampon contient N messages non retirés, on ne peut y déposer de message supplémentaire
- Un message donné n'est retiré qu'une seule fois après avoir être déposé.
- Une opération impossible (dépôt dans un tampon plein ou retrait depuis un tampon vide) bloque le processus qui tente de l'exécuter.

La structure des deux processus est la suivante :

Processus Producteur
Debut
Cycle
Produire(message)
Deposer(message)
FinCycle
Fin

Processus Consommateur
Debut
Cycle
Retirer(message)
Consommer(message)
FinCycle
Fin

Les procédures Déposer et Retirer doivent implémenter les contraintes mentionnées ci dessus.

4.4.3- Les mécanismes

L'importance vitale du problème de l'exclusion mutuelle, notamment dans le domaine des systèmes d'exploitation a permis le développement de nombreux mécanismes de synchronisation. Ces mécanismes constituent les objets (outils) de base que le programmeur utilise pour exprimer les relations inter processus de son application parallèle. La nature de ce type d'application exige, dans la plupart des cas une fiabilité visant le zéro défaut. Pour garantir une telle exigence, la solution admissible doit vérifier les propriétés de bon fonctionnement ci dessous :

- Exclusion : A tout instant un seul processus au plus est en section critique (*définition de la section critique*).
- Accès : Si plusieurs processus sont bloqués à l'entrée d'une section critique libre l'un d'eux doit y entrer au bout d'un temps fini (*la section critique est atteignable*)
- Indépendance : Si un processus est bloqué en dehors de sa section critique, ce blocage ne doit pas empêcher l'entrée d'un autre processus à la section critique (*Indépendance de la partie conflictuelle et non conflictuelle*).
- Uniformité : Aucun processus ne doit jouer le rôle privilégie, ils doivent tous utiliser le mêmes mécanisme (*Banalisation de la solution*).

Les protocoles d'acquisition et de libération doivent respecter ces propriétés.

4.4.3.1- Le masquage d'interruption

1- Présentation du mécanisme

Le principe de ce mécanisme est de rendre non observable l'état de la machine durant l'exécution de la section critique (rendre indivisible la SC) en utilisant le masquage/Démasquage d'interruption. Le masquage d'interruption permet au processeur d'exécuter le code de la section critique jusqu'à sa fin sans être 'interrompu par un autre processus. Par contre, le démasquage permet de remettre le processeur dans un état autorisant la prise en compte des interruptions.

2- Usage du mécanisme

L'utilisation de ce mécanisme pour résoudre le problème de l'exclusion mutuelle se fait de la manière suivante :

```
ProtocoleAcquisition :   CLI    (* Interdire les interruptions *)
                      <Section Critique>
ProtocoleLibération  :   STI    (* Autoriser les interruptions  *)
```

3- Caractéristiques du mécanisme

Ce mécanisme présente les deux caractéristiques suivantes :

- Valable uniquement sur machine monoprocesseur car le faite de masquer les interruptions d'un processeur n'entraîne pas le masquage des autres processeurs.
- Si la séquence du code critique est assez longue, on risque de perdre des interruptions soulevées, ce qui peut conduire à la non-progression de l'application.

4.4.3.2- Test And Set

1- Présentation du mécanisme

Test And Set (TAS) est une instruction disponible sur certaines machines. Elle permet la consultation et la mise à jour d'une variable de façon indivisible. Le code qui implémente cette instruction est de la forme :

```
Algori thme TAS(m,R) (* Opération indivisible *)
Debut
  < Bloquer l'accès à la cellule mémoire m >
    R := m ;     (* R étant un registre du processeur*)
    m:= 1 ;
  < Libérer l'accès à la cellule de mémoire m >
Fin
```

2- Usage du mécanisme

Pour utiliser ce mécanisme dans la résolution du problème de l'exclusion mutuelle, nous devons déclarer une variable commune (partagée) EtatSCritique qui prend la valeur 0 ou 1 selon que la section critique est libre ou occupée. La consultation et la mise à jour de cette variable sont réalisées par l'instruction TAS de la manière suivante :

```
ProtocoleAcquisition :      Etiq : TAS(EtatSCritique,R)
                                Si(R<>0) alors aller à Etiq;

                            <Section critique>

ProtocoleLibération              EtatSCritique := 0

Valeur initiale de EtatSCritique := 0
```

3- Caractéristiques du mécanisme

Ce mécanisme présente aussi deux caractéristiques :

- Nécessite un mécanisme élémentaire câblé pour protéger la variable EtatRessource
- L'attente active monopolise le processeur et donc dégrade la performance globale de la machine, ceci exclue l'usage de ce mécanisme dans un environnement monoprocesseur

4.4.3.3- Les sémaphores

1- Présentation du mécanisme

Ce mécanisme a été inventé par Dijkastra. Formellement, un sémaphore s est constitué d'un entier e(s) pouvant prendre des valeurs positives, négatives ou nulles ; et une file d'attente f(s).Toute opération sur un sémaphore est indivisible. La création d'un sémaphore se fait par déclaration qui doit spécifier la valeur initiale e0(s) de e(s).

Conceptuellement, un sémaphore peut être vu comme un distributeur de tickets, initialement dispose d'un certain nombre de tickets, éventuellement 0 ticket. Un processus utilisateur demande un ticket en invoquant une opération appelée P (Proberen en Hollandais) : si au moins un ticket est disponible, le processus appelant le prend et continue son exécution. Dans le cas contraire, la demande est enregistrée dans une file d'attente et le processus est bloqué dans l'attente d'une condition (disponibilité d'un ticket pour lui).Une deuxième opération possible sur un sémaphore V(Verthogen) qui permet après avoir été invoquée par un processus :

- Rendre disponible un ticket dans le distributeur.
- Rendre actif un des processus de la file d'attente associée au sémaphore si celle ci n'est pas vide. Dans le cas contraire, le ticket est conservé dans le distributeur et donc le prochain demandeur verra sa demande immédiatement honorée.

```
Algorithme de la primitive P(s)  (* Opération indivisible *)
Debut
  e(s):= e(s) - 1
 Si ( e(s) < 0 )  alors
   Bloquer le processus r ayant effectué la requête
  Mettre le processus r dans une file d'attente f(s)
 Finsi
Fin
```

```
Algorithme de la primitive V(s)  (* Opération indivisible *)
Debut
  e(s):= e(s) + 1
 Si ( e(s) <= 0 )  alors
  Sortir un processus q de la file d'attente f(s).
  Relancer le processus q.
  Finsi
Fin
```

2- Quelques variantes de sémaphores

1- Sémaphore d'exclusion mutuelle

Un sémaphore d'exclusion mutuelle appelé sémaphore mutex (pour Exclusion Mutuelle) est un cas particulier des sémaphores à compte. Ce sémaphore initialisé à 1 autorise un seul processus actif à franchir P(mutex).

2- Sémaphore privé

Un sémaphore Sp est privé à un processus p si seul p peut exécuter les opérations P(Sp) et V'Sp). Les autres processus ne peuvent agir sur Sp que par V(Sp). La valeur initiale de Sp est égale à 0.

3- Usage du mécanisme

La réalisation de l'exclusion mutuelle à l'aide des sémaphores se fait selon le schéma suivant :

```
Semaphore Mutex = 1 ;   (* Contexte global*)

ProtocoleAcquisition :          P(mutex)
                                <Section critique>
ProtocoleLibération             V(Mutex)
```

4- Caractéristiques du mécanisme

Ce mécanisme présente les caractéristiques suivantes :

- Avec ce mécanisme le processus désirant exécuter sa section critique quitte le processeur si les conditions ne sont pas satisfaites.
- Ce mécanisme ne présente aucun risque concernant la perte des interruptions
- L'usage de mécanisme nécessite une intention particulière de la part du programmeur, car un mauvais appel de P ou de V peut entraîner des conséquences néfastes