Chapitre 03
Gestion de processeur
(Gestion de processus)

1 Introduction

Selon la nature du SE, monoprogrammée ou multiprogrammée, l'évolution du programme et son exécution dépendent largement de la stratégie adoptée par le SE pour l'allocation du processeur.

La notion de *processus* a été introduite pour désigner l'entité dynamique qui correspond à l'exécution d'une suite d'instructions. Le processus est l'entité de travail du système. L'exploitation d'un ordinateur est alors vue comme un ensemble de processus s'exécutant en parallèle. La simultanéité est réalisée par l'exécution de plusieurs processus en alternance sur un seul processeur. Ce dernier commute entre plusieurs processus.

Un processus est une action, une séquence d'opération qui se déroule pour réaliser une tâche déterminée. C'est un programme en exécution ; c'est une entité active capable de générer des évènements.

Un processus est composé de :

- un code exécutable provenant du programme source augmenté des bibliothèques introduites lors de l'édition de liens.
- Un contexte : c'est toutes les ressources qu'il possède ; une image des registres, en mode d'état du processus etc…

Nous insistons sur le fait qu'un programme par lui-même n'est pas un processus ; un programme est une entité *passive*, tel que le contenu d'un fichier stocké sur disque, tandis qu'un processus est une entité active avec un compteur d'instructions qui spécifie l'instruction suivante à exécuter et un ensemble de ressources associées.

2 Concepts sur les processus

Un processus ou process est un programme en exécution, il est décrit par un contexte.

2.1 Contexte d'un processus

Il est formé de trois sous-contextes (les informations présentées dans le **PCB**):

Identification de processus :

. Identifiant de ce processus.

. Identifier les processus qui ont créé ce processus (le processus parent).

. Identifiant de l'utilisateur.

Le contexte du processeur : qui représente les informations liées au CPU (information d'état de processeur) .

- Compteur Ordinal CO
- Mot d'état PSW
- Registres généraux
- Pile

Information de contrôle de processus :

- Ordonnancement et les informations d'état.
- Structuration des données.
- Interprocess communication.
- Privilèges de processus.
- Gestion de la mémoire (@ d'implantation des segments code et donnée).
- Propriété et utilisation des ressources.

2.2 Etat d'un processus

Un processus prend un certain nombre d'état durant son exécution, déterminant sa situation dans le système vis-à-vis de ses ressources. Les trois principaux états (Figure 1) d'un processus sont:

Prêt ou en attente: Le processus attend la libération du processeur pour s'exécuter

Actif ou élu: le processus est en exécution

Bloqué ou endormi: le processus attend une ressource physique ou logique autre que le processeur pour s'exécuter (mémoire, fin d'E/S, …).

Avec **l'étape de création** (Etat **Nouveau**) : le processus en cours de création.

Avec **l'étape de terminaison** (**Etat Terminé**) : le processus a fini son exécution.

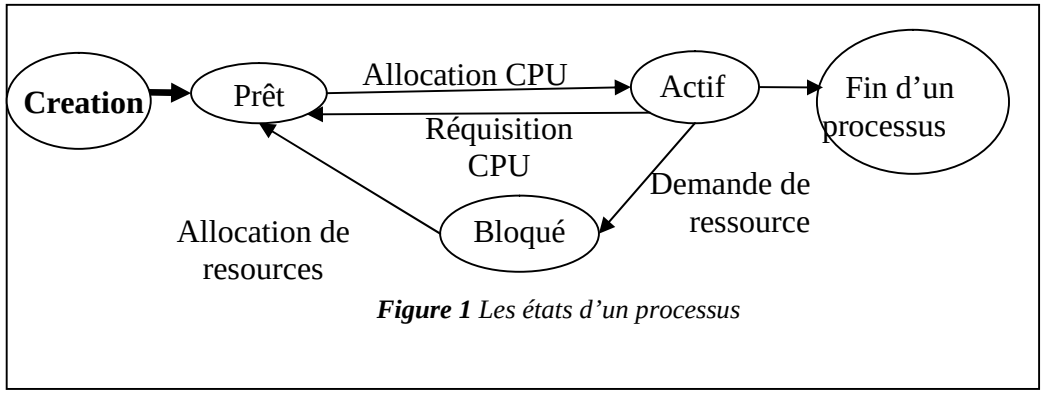


Figure 1 *Les états d'un processus*

Représentation interne des processus

Au moment du chargement d'un programme en MC, le SE crée une structure représentant le processus associé au programme exécutable. Cette structure contient toutes les informations nécessaires pour le changement d'état d'un du processus dans le SE, on l'appelle Bloc de Contrôle du Processus (**Process Control Bloc** (**PCB**)).

Le PCB d'un processus contient :

- ID_processus : identificateur unique du processus (un entier)
- L'état du processus (prêt, bloqué, actif)
- Contexte du processus.
- Compteur d'instructions.
- Les registres de processeur.
- Information de comptabilisation.
- Information sur la gestion de mémoire.
- Informations sur l'ordonnancement du processeur.
- Informations sur les ressources occupées (nature ressource, nombre), (fichiers, E/S, .etc)
- …

➔ Pour que le SE puisse gérer les différents process, il crée une table appelée table des processus dont chaque entrée contient un pointeur vers un processus chargé.

➔ Chaque état (bloqué, prêt) est géré par une file d'attente propre à lui dont la stratégie de gestion est propre au SE

2.3 Opérations sur les processus

a) Création de processus
b) Destruction de processus
c) Mettre en attente et réveiller

a) Création de processus

Un processus P1 peut créer un autre processus P2. Le premier P1 est le père le deuxième P2 est son fils.

- ➔ La création d'un processus implique la création et l'insertion de son PCB dans la table des processus
- ➔ Selon les SE, le fils s'exécute en parallèle avec son père ou le père se bloque jusqu'à la fin de son fils

Exemple: sous Unix ou Linux : id = fork() permet de créer un processus fils.

b) Destruction de processus :

Un processus se termine de trois façons différentes :

- il se termine normalement, après l'exécution de sa dernière instruction.
- il exécute une instruction autodestructive. (exit())
- il est détruit par un autre processus (kill(id_process) sous Unix/Linux).

➔La destruction d'un processus implique la destruction de son PCB et la libération des ressources détenues.

➔ Un processus fils ne peut détruire un autre de sa hiérarchie

➔ Généralement, la destruction du processus père implique la destruction de tous les fils

2.4 Commutation de contexte

Dans un système à temps partagé, le processeur assure l'exécution de plusieurs processus en parallèle (pseudo-parallèle)

Le passage dans l'exécution d'un processus à un autre nécessite une opération de sauvegarde du contexte du processus arrêté, et de chargement de celui du nouveau processus. Ceci s'appel la commutation de contexte.

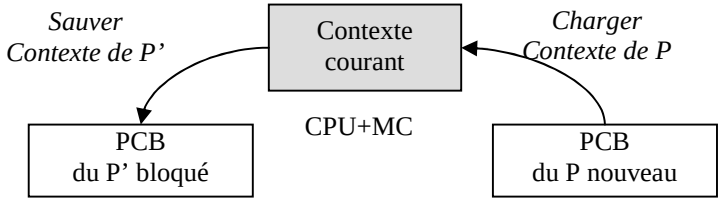


Figure 2 *Mécanisme de commutation de contexte*

La commutation de contexte consiste à changer les contenus des registres de processeur central par les informations de contexte du nouveau processus à exécuter.

Exemple d'un changement de contexte entre deux processus P1 et P2 :

Le SE fait une commutation de contexte en trois opérations indivisibles :

Sauvegarder le contexte du processus bloqué P1 dans son PCB.

Charger le contexte du processus P2 à partir de la file des processus prêts dans le contexte du processeur.

Se brancher à l'adresse existante dans le CO (c'est une @ d'une instruction de P2).

Remarques:

- L'instant de commutation de contexte s'appel **point observable**.
- La table des processus est généralement sauvegardée dans deux types d'emplacement possibles:
  - Emplacement Mémoire
  - Pile

Notion des ressources :

Une ressource est une entité dont a besoin un processus à un instant donné pour s'exécuter. Elle est caractérisée, en particulier, par le nombre de processus qui peuvent l'utiliser au même moment. Ce nombre peut être ***illimité***, et aucun contrôle n'est nécessaire lors de l'allocation.

- Il peut y en avoir un nombre quelconque. Il n'y a alors pas de contrôle à mettre en oeuvre. Dans l'exemple culinaire, l'horloge a été considérée comme une ressource qui peut être accédée par un nombre quelconque de processus.
- Il peut y en avoir plusieurs, mais en nombre limité. Il faut alors contrôler lors des allocations quece nombre n'est pas dépassé. Dans l'exemple culinaire, le four ne peut être utilisé simultanément que par deux processus.
- Il peut y avoir au plus un processus qui utilise la ressource. Dans l'exemple culinaire, le batteur ne peut être utilisé que par au plus un processus. On dit alors que la ressource est une ***ressource critique*****.** On dit aussi que les processus sont en ***exclusion mutuelle*** pour l'accès à cette ressource. Il en est ainsi du processeur physique, d'une imprimante, d'un dérouleur de bande, etc...

3 Ordonnancement du processeur

Dans un SE multiprogrammé, plusieurs processus peuvent être à l'état prêt en même temps. Ces processus rentrent en concurrence pour l'acquisition de CPU. Cette dernière est une **ressource critique.**

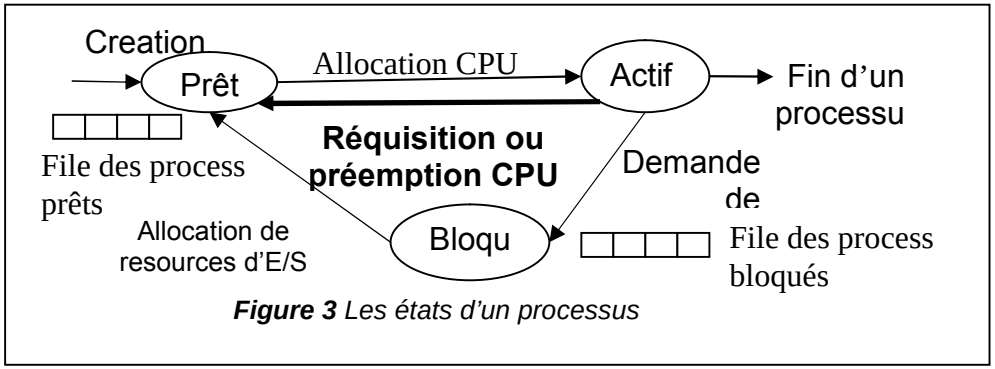


Figure 3 *Les états d'un processus*

On parle d'**allocation de processeur,** ou allocation du temps processeur aux différents processus en attente dans la file des processus prêts. Le SE, à travers son programme de gestion du processeur détermine **qui** exécuté et pour **combien de temps**. Ce programme s'appel l'Ordonnanceur ou scheduler (en anglais).

Questions :

- Dans le cas de N processus prêts, à quel processus doit on allouer le CPU ?
- Un processus en attente est il obliger d'attendre la fin de celui actif ?
- Si un processus urgent (prioritaire) demande le CPU, doit il attendre ?
- Quelles sont les politiques (ou stratégies) d'allocation du processeur ?
- Comment peut-on optimiser l'utilisation du CPU pour maximiser les performances du SIQ ?

3.1 Types d'ordonnancement

L'ordonnancement est de deux types :

1) Préemptive (avec réquisition)
2) Non préemptive (sans réquisition ou sans suspension)
Une politique d'ordonnancement est dite non préemptive si une fois le processeur est alloué à un **processus**, **celui-ci** le gardera jusqu'à :

- la fin de son exécution,
- s'il se bloque sur une ressource d'E/S non disponible,
- attente de la satisfaction d'une demande d'une ressource d'E/S.

Dans une politique préemptive, un processus qui est en cours d'exécution est suspendu (ou réquisitionner), ce processus est inséré dans une file d'attente des processus prêts.
Exemple :
Un système à temps partagé est préemptif.
Un système temps réel est non préemptif (traitement critique non interruptible).

3.2 Critères de performance d'une politique d'ordonnancement

Les performances d'une politique d'ordonnancement sont déterminées suivant les critères suivants :

Taux d'occupation du CPU = $\frac{Temps\_d'utilisation\_du\_CPU}{Temps\_total\_d'exécution}$

Taux_CPU → 0% : La politique est mauvaise
Taux_CPU → 100% : La politique est très performante.

Rendement de l'UC = R= $\frac{nombre\_de\_processus\_exécutés}{unité\_de\_temps}$

R→faible : La politique avec un rendement faible.
R→grand : Le rendement de la politique est bon.

Temps d'attente d'un processus Pi = t_i : c'est le temps passé par un processus Pi dans la file d'attente des processus Prêts.
La politique d'ordonnancement doit minimiser le temps d'attente t.

Temps d'Attente Moyen =TAM= la moyenne des temps d'attente de N processus :

$$TAM = \frac{\sum_{i=1}^{N} t_i}{N}$$

TAM → 0 : La politique est très performante.
TAM→max : La politique est mauvaise.

Temps de réponse d'un processus Pi = tr_i = temps écoulé (passé) entre le lancement du programme et la fin de son exécution.
tr_i = l'instant de fin de processus – instant d'arriver du processus.
L'Ordonnanceur doit minimiser tr_i pour chaque processus Pi.

Temps de Réponse Moyen = TRM = la moyenne des temps de réponse de N processus :

$$TRM = \frac{\sum_{i=1}^{N} tr_i}{N}$$

TRM→0 : La politique est très performante.
TRM→max : La politique est mauvaise.

Remarque :
Les politiques d'un Ordonnanceur pour améliorer les performances à attendre diffèrent d'un type de SE à un autre.

3.3 Politiques d'ordonnancement

3.3.1 Politiques non préemptives

Dès que le CPU est donné à un processus, ce dernier ne peut être interrompu avant la fin de son temps CPU.

3.3.1.1 Premier Arrivé Premier Servi (PAPS) « First Com First Served-FCFS-FIFO »

Consiste à servir le processus qui arrive le premier dans le système.
Avantage :

- implémentation simple.

Inconvénient :

- Les processus longs pénalisent les processus courts.
- Performances généralement mauvaises.

3.3.1.2 Politique du job le plus court d'abord (SJF : Shortest Job First)

Consiste à servir les jobs (processus) courts avant les jobs longs. C'est une politique qui nécessite la connaissance préalable du temps d'exécution des jobs.

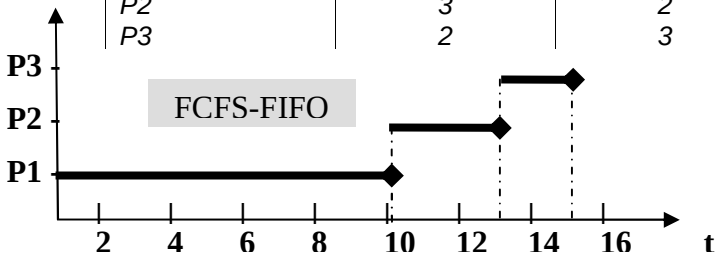
Avantage :

- Performances meilleures en temps d'attente moyen TAM.

Inconvénients :

- Il est difficile d'estimer le temps d'exécution d'un processus.
- Génération de la **famine** (stravation, privation) : l'arrivée successive de processus de court temps pénalise les processus de long temps d'exécution.

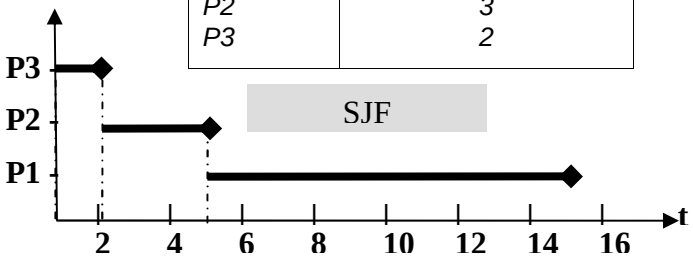
Processus	*Temps CPU*	*Ordre d'arrivée*
P1	*10*	*1*
P2	*3*	*2*
P3	*2*	*3*



t1= 0, t2=10, t3=13 → TAM = (0+10+13)/3 =7.66 unités
tr1=10, tr2= 13, tr3=15 → TRM = (10+13+15)/3 =12.66 unités

Figure 3 *Diagramme de Gant d'un exemple d'ordonnancement FIFO*

Processus	*Temps CPU*
P1	*10*
P2	*3*
P3	*2*



t1 = 5 ;t2 = 2, t3 = 0 →TAM = 7/3 = 2,33 unités
Tr1 = 15, Tr2 = 5, tr3 = 2→ TRM = 22/3 = 7,33 unités

Figure 4 *Diagramme de Gant d'un exemple d'ordonnancement SJF*

3.3.1.3 Ordonnancement par priorité et sans réquisition

Chaque processus possède une priorité fixe consistant en un nombre entier positif fixe. Le processus élu est celui dont la priorité est la plus haute.
Cette politique est sans réquisition, un processus élu garde le CPU même si un autre processus plus prioritaire vient d'entrer dans la file des prêts.

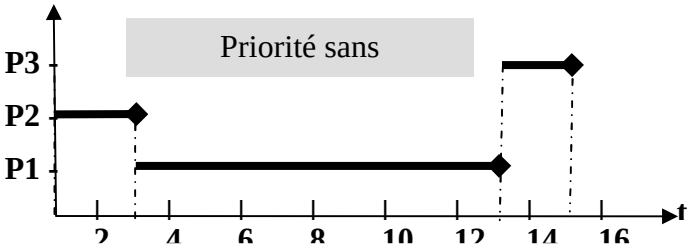
Avantage :

- Possibilité de spécifier la priorité à priori.

Inconvénients :

- L'arrivée successive de processus de hautes priorités pénalise les processus de faibles priorités.

Processus	*Temps CPU*	*priorité*
P1	*10*	*2*
P2	*3*	*1*
P3	*2*	*3*



t1 = 3 ;t2 = 0, t3 = 13→ TAM = 16/3 = 5,33 unités
tr1 = 13; tr2 = 3, tr3 = 15 → TRM = 30/3 = 10 unités

Figure 5 *Diagramme de Gant d'un exemple d'ordonnancement par priorité et sans réquisition.*

3.3.2 Politique préemptives (avec réquisition)

3.3.2.1 SRTF (Shortest remaining time first ou SJF avec requisition)

Lorsqu'un processus P est en cours d'exécution, et qu'un nouveau processus Q ayant un temps d'exécution plus court que celui qui reste pour terminer l'exécution du processus P, ce dernier est arrêté, et le nouveau processus Q est exécuté. Cette méthode est équivalente à SJF mais avec réquisition.

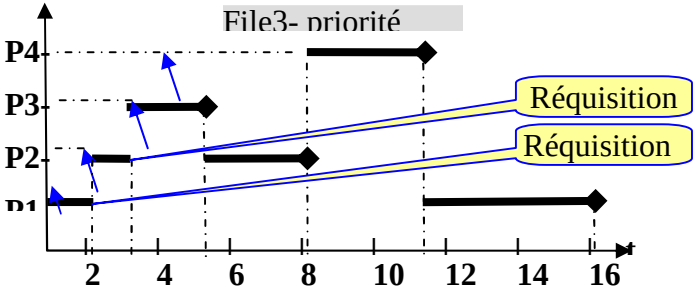
Avantage :

- les processus courts ne sont pas obligés d'attendre de longs périodes de temps, ce qui améliore TAM.

Inconvénient :

- Une famine peut être créé par le traitement des processus court et en laissant attendre les processus long.

Processus	Temps CPU	temps d'arrivée
P1	7	0
P2	4	2
P3	2	3
P4	3	4



t1 = 9 ;t2 = 2, t3 = 0, t4 = 4 → TAM = 15/4 = 3,75 unités
Tr1 = 16; Tr2 = 6, tr3 = 2, tr4 = 7 →TRM = 31/4 = 7,75 unités

Figure 6 *Diagramme de Gant d'un exemple d'ordonnancement SRTF.*

3.3.2.2 Politique par priorité avec réquisition

A l'arrivée d'un processus P plus prioritaire qu'un autre Q en cours d'exécution. Le processeur est alloué immédiatement au processus P.

Avantage :

- Possibilité de spécifier la priorité et de servir les processus urgents.

Inconvénient :

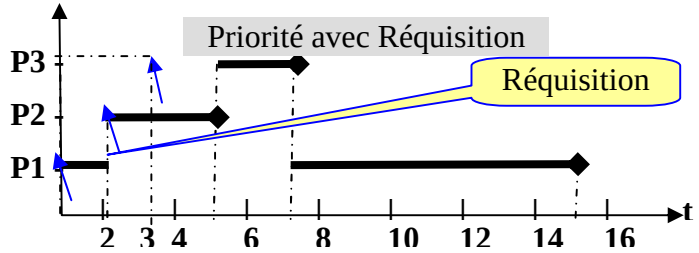
- l'arrivée successive de processus de hautes priorités pénalise les processus de faibles priorités (famine).

3.3.2.3 Round Robin RR (Tournequet ou ordonnancement circulaire)

Le CPU est alloué à chaque processus existants fans la file des prêts d'une façon FIFO circulaire pendant une durée de temps bien déterminée Q appelée **quantum de temps**. A la fin des quantums, le CPU est enlevé du processus actif au profit du processus suivant dans la file des prêts.

Q→ ∞ alors [nombre de commutations → 0 et (RR) → FIFO]
Q→0 alors [nombre de commutation → ∞ et le travail de l'Ordonnanceur consiste à commuter les contextes d'une manière infinie.

Processus	Temps CPU	Temps d'arrivée	priorité
P1	10	0	1
P2	3	2	3
P3	2	3	2



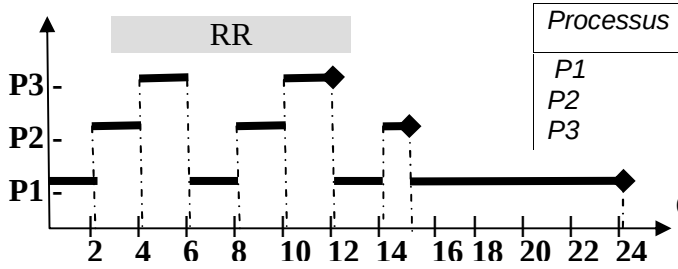
t1 = 5 ;t2 = 0, t3 = 2 → TAM = 7/3 = 2,33 unités
Tr1 = 15; Tr2 = 3, tr3 = 4 → TRM = 21/3 = 7 unités

Figure 7 *Diagramme de Gant d'un exemple d'ordonnancement avec priorité et préemptif*

Avantage

- Politique RR s'adapte bien avec les systèmes à temps partagés.

Inconvénient :



Processus	Temps CPU
P1	15
P2	5
P3	4

t1 = 9 ;t2 = 10, t3 = 8 → TAM = 27/3 = 9unités
Tr1 = 24;Tr2 = 15;tr3 = 12 → TRM = 51/3 = 13,66 unités

Figure 8 *Diagramme de Gant d'un exemple d'ordonnancement avec RR*

- Le temps consommé pour chaque commutation de contexte réduit les performances du système.
- Le temps de réponse d'un processus peut être très long.

3.3.2.4 Round Robin multi niveau RRM

Dans cette stratégie, la file des processus prêts est subdivisée en plusieurs files suivant la classe des processus et leurs priorités. Chaque file est gérée par un tourniquet de quantum Q.

Exemple (voir figure ci-dessous)**:**

File1 → processus système
File2 → processus administrateur
File3 → processus utilisateurs

3.4 Ordonnanceur de processeur de LINUX

Trois politiques d'ordonnancement sont proposées : la première est utilisée pour ordonnancer des processus ordinaires, la deuxième pour des processus temps réel et la dernière pour les processus à temps partagé. Chaque processus crée est attaché à l'une des politiques suivantes :

- Politique SCHED_FIFO non préemptive avec priorité pour les processus temps réel.
- Politique SCHED_RR préemptive gérée par un tourniquet entre processus de même priorité, pour les processus temps partagé.
- Politique SCHED_OTHER pour un processus ordinaire. C'est une politique avec extinction de priorité.

4 Concept de thread (processus léger)

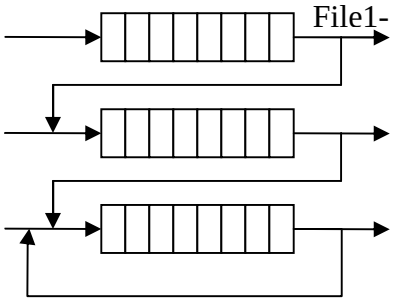


Figure 3 *Ordonnancement Multi niveaux avec* ***extinction de priorités***

Un processus leger (en anglais, thread), également appelé fil d'exécution (unité de traitement, unité d'exécution, fil d'instruction, processus léger) représente l'exécution d'une tâche d'un processus.

Chaque processus peut faire fonctionner simultanément (en parallèle) plusieurs threads. Un processus léger comprend :

- un identificateur
- un compteur de programme
- un ensemble de registre
- une pile

De point de vue de l'utilisateur, les threads s'exécutent en parallèle. Un thread partage avec d'autres threads appartenant au même processus le segment de code, le segment de données et des ressources d'E/S (fichiers).

Avantages :

- Partage de ressources entre threads d'un même processus.
- Augmenter le degré de parallélisme lors de l'utilisation d'une architecture multiprocesseur. Ceci améliore les performances du système.

Inconvénients :

- La programmation utilisant des processus légers est plus difficile que la programmation simple.
- Nécessite des stratégies de synchronisation pour l'accès aux ressources partagées.
- La complexité des programmations utilisant des processus légers est plus grande que celle des programmes simples.

5 Le problème de la Section Critique & L'exclusion Mutuelle & Interblocage

P1 : Process Crédit

lire(compte) ; (a)
compte := compte + 1000 ; (b)
écrire(compte) (dans la MC) (c)

P2 : Process Débit

lire(compte) ; (a')
compte := compte - 500 ; (b')
écrire (compte) (dans la MC) (c')

Fin

Position du problème : soit deux processus qui s'exécutent en parallèle pour mettre à jour un compte bancaire.
Supposons que compte contient initialement 3000.
1er cas : exécution séquentielle : le compte a une valeur cohérente
- exécution suivant (P1 puis P2) ou (P2 puis P1) : le résultat est compte = 3000 + 1000-500 = 3500.
2ème cas : exécution concurrentielle (imbriquée*):* le compte pourrait avoir une valeur non cohérente
- Soit la suite d'exécution suivante : a, a', b, b', c, c' ; la valeur finale de compte est : 2500 ! → erreur !
➔ L'incohérence est due en fait qu'on a autorisé l'accès **simultané** à la variable partagée compte pour que chaque process en fasse une copie chez lui. La dernière mise à jour de P2 a annulé la première mise à jour de P1 qui n'a pas été signalée chez P2 !

5.1 Section critique

C'est la partie d'un programme où la ressource partagée (variable, fichier, .etc) est seulement accessible par un seul processus à un moment donné. Autrement dit, l'accès simultané à la section critique sur une même ressource critique est interdit. On dit que la ressource est en accès exclusif et les processus sont en **exclusion mutuelle**.
L'exclusion mutuelle ou Mutex est la primitive qui protège une section critique.
Exemple : dans l'exemple précédent **compte** est la section critique.

5.2 Interblocage

C'est la situation où un ensemble de processus sont bloqués indéfiniment. Chacun en attente de ressources pour sa progression détenue par un autre processus.

Exemple :
Lorsqu'un thread T1 ayant déjà acquis la ressource R1 demande l'accès à une ressource R2, pendant que le thread T2, ayant déjà acquis la ressource R2, demande l'accès à la ressource R1. Chacun des deux threads attend alors la libération de la ressource possédée par l'autre. La situation est interblocage.

6 Conclusion

- Un programme est une entité statique alors qu'un processus auquel on associe un contexte est une entité dynamique.
- Dans un système multiprogrammé, les processus cherchent à avoir les ressources, en particulier le CPU qui est une ressource critique. L'objectif d'un système d'interruption est de réduire le temps de supervision des ressources par le SE.
- Les interruptions sont de deux types, les ITs logicielles et matérielles.
- L'Ordonnanceur est un module du SE chargé d'allouer le CPU aux **processus concurrents**.
- En partageant le CPU, l'Ordonnanceur applique des politiques visant à maximiser les performances globales du système. Ces politiques sont soit préemptives ou non préemptives.
- Les politiques sont choisis selon la nature du SE. L'application de la politique RR simple dans un système Batch est un mauvais choix, alors que c'est meilleur dans un système à temps partagé tandis qu'elle est dangereuse dans un système de type temps réel. Ce type préfère une des stratégies avec priorité fixe.
- Le partage de ressources entre processus génère deux problèmes de synchronisation :
    - Protection d'une section critique
    - Interblocage.