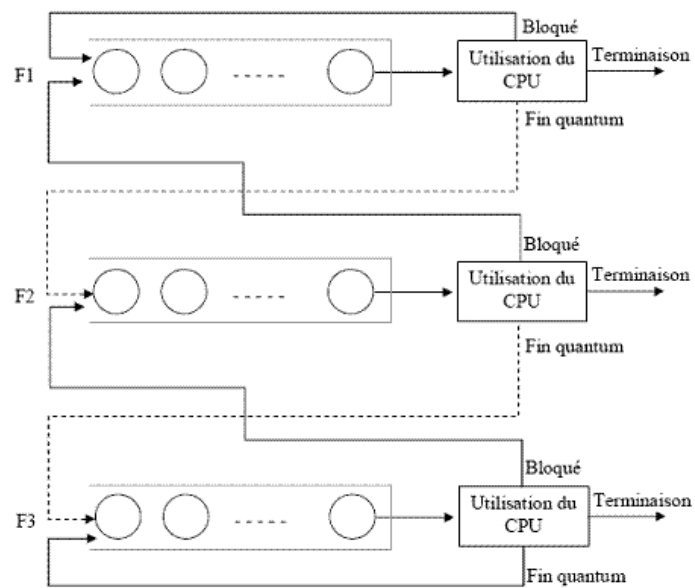


Ordonnancement de processus

Version 1



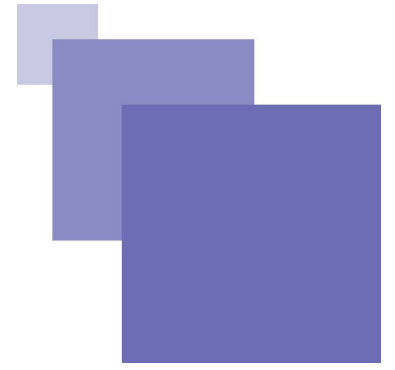
Y. CHALLAL, H. BETTAHAR, M. VAYSSADE

Table des matières



Objectifs	5
I - Gestion du Processeur Central	7
A. Concept d'ordonnancement de processus.....	7
B. Algorithmes d'ordonnancement.....	8
1. Ordonnancement sans réquisition du CPU.....	8
2. Ordonnancement avec réquisition du CPU.....	9

Objectifs



- Analyser la fonction d'allocation du processeur central
- Analyser les algorithmes d'ordonnement de processus

Gestion du Processeur Central

Concept d'ordonnancement de processus

7

Algorithmes d'ordonnancement

8

Sur les anciens systèmes de traitement par lots, le choix du job à exécuter était simple : on prenait le job suivant sur la bande. Dans un système multiprogrammé, plusieurs processus peuvent être en concurrence pour l'obtention de temps processeur. Cette situation se produit chaque fois que deux processus ou plus sont en état prêt en même temps. S'il n'y a qu'un seul processeur, un choix doit être fait quant au prochain processus à exécuter. La partie du système d'exploitation qui effectue ce choix se nomme l'**ordonnanceur** (*scheduler*) et l'algorithme qu'il emploie s'appelle **algorithme d'ordonnancement**. Dans cette partie du cours nous examinerons ces différentes notions liées à l'ordonnancement des processus en vue d'allouer le processeur central judicieusement [*tanenbaum03*].

A. Concept d'ordonnancement de processus

Multiprogrammation et temps partagé

Dans un système multiprogrammé, on "démontre" plusieurs programmes à la fois, et lorsque l'un d'eux lance une E/S et donc se met "en attente" de la fin de cette Entrée/Sortie, on en profite pour utiliser le cpu pour exécuter un autre programme : on profite de la différence de vitesse importante entre l'exécution des E/S et le cpu. La multiprogrammation s'est généralisée et s'est encore compliquée avec l'apparition des systèmes "temps partagé" interactifs. L'idée de base à l'origine de la notion du "temps partagé" est d'utiliser la différence de vitesse entre le CPU et le "temps de réaction" des utilisateurs : on alloue à chaque utilisateur une petite "tranche" de temps du CPU. Si, par exemple, le CPU peut exécuter 10 millions d'instruction par seconde, avec 10 utilisateurs, chacun aura l'impression de disposer d'un CPU travaillant à 1 millions d'instructions par seconde. Ce schéma simpliste peut être considérablement amélioré si on tient compte des E/S : en effet quand un programme fait une E/S, il va passer un temps important (à l'échelle du CPU) à attendre la fin de cette E/S : pendant ce temps d'attente on peut utiliser le CPU pour faire des calculs pour d'autres programmes, sans pénaliser le programme qui a fait une E/S.



Fondamental : Allocation du CPU et changement de contexte

"Retirer" le processeur à un processus et l'allouer à un autre processus passe par une opération de sauvegarde du contexte de l'ancien processus et chargement du contexte du nouveau processus. Si plusieurs processus sont à l'état prêt, le système d'exploitation doit choisir un processus en particulier qui deviendra le

processus actif. Ce choix est basé sur un algorithme d'ordonnancement et le module du système d'exploitation qui joue ce rôle s'appelle l'ordonnanceur (scheduler).



Définition : Ordonnancement des processus

L'ordonnanceur (scheduler) est un module du système d'exploitation qui attribue le contrôle du CPU à tour de rôle aux différents processus en compétition suivant une politique définie à l'avance par les concepteurs du système.

Critères d'ordonnancement

La politique d'ordonnancement doit optimiser les performances du système à travers plusieurs critères :

- **Rendement d'utilisation du CPU** : pourcentage de temps pendant lequel le CPU est actif. Plus ce paramètre est élevé mieux c'est. Ceci permettra d'augmenter le nombre de jobs achevés par unité de temps.
- **Utilisation globale des ressources**: c'est assurer une occupation maximale des ressources de la machine et minimiser le temps d'attente pour l'allocation d'une ressource à un processeur [belkhir96].
- **Équité** : capacité de l'ordonnanceur à allouer le CPU d'une façon équitable à tous les processus de même priorité (éviter la famine).
- **Temps de rotation** : durée moyenne nécessaire pour qu'un processus termine son exécution.
- **Temps d'attente** : durée moyenne qu'un processus passe à attendre le CPU.
- **Temps de réponse** : temps moyen qui s'écoule entre le moment où un utilisateur soumet une requête et celui où il commence à recevoir les réponses.

B. Algorithmes d'ordonnancement

Deux catégories d'algorithmes

L'ordonnanceur du CPU permet de décider à quel processus (dans la file d'attente des processus prêts) attribuer le contrôle du CPU. Les stratégies d'ordonnancement peuvent être regroupées en deux catégories : sans réquisition du CPU (stratégie non préemptive) ou avec réquisition du CPU (stratégie préemptive).

1. Ordonnancement sans réquisition du CPU

Dans cette catégorie, un processus conserve le contrôle du CPU jusqu'à ce qu'il se bloque ou qu'il se termine. Cette approche correspond aux besoins des travaux par lots (systèmes batch). Il existe plusieurs algorithmes dans cette catégorie :



Méthode : Premier Arrivé Premier Servi (FCFS : First come First Served)

Les tâches sont ordonnancées dans l'ordre où elles sont reçues. Le processus qui sollicite le CPU le premier sera servi le premier. On utilise une structure de file.



Exemple : Premier Arrivé Premier Servi (FCFS : First come First Served)

PID	Heure d'arrivée	Durée d'exécution	Début d'exécution	Heure de fin d'exécution	Temps de traitement
1	10	2	10	12	2
2	10,1	1	12	13	2,9
3	10,25	0,25	13	13,25	3

Tableau 1 : FCFS

Temps de traitement moyen = 2.6

La stratégie FCFS désavantage les processus courts s'ils ne sont pas exécutés en premier.



Méthode : Le Plus Court Job d'abord (SJF: Shortest Job First)

Cet algorithme nécessite la connaissance du temps d'exécution estimé de chaque processus. Le CPU est attribué au processus dont le temps d'exécution estimé est minimal.



Exemple : Le Plus Court Job d'abord (SJF: Shortest Job First)

PID	Heure d'arrivée	Durée d'exécution	Début d'exécution	Fin d'exécution	Temps de traitement
1	10	2	10	12	2
2	10,1	1	12,25	13,25	3,15
3	10,25	0,25	12	12,25	2

Tableau 2 : SJF

Temps de traitement moyen = 2.38

Si tous les processus sont présents dans la file d'attente des processus prêts au moment où commence l'allocation du CPU, l'algorithme SJF minimise le temps moyen de traitement, pour l'ensemble des algorithmes sans réquisition. Cette propriété n'est plus vraie si des processus entrent dans la file d'attente des processus prêts au cours de l'allocation du CPU.

2. Ordonnement avec réquisition du CPU

Dans cette catégorie, l'ordonnanceur peut retirer le CPU à un processus avant que ce dernier ne se bloque ou se termine afin de l'attribuer à un autre processus. A chaque quantum (unité de temps) l'ordonnanceur choisi dans la liste des processus prêts un processus à qui le CPU sera alloué. Cette approche correspond aux systèmes interactifs.



Méthode : Le temps restant le plus court (SRT : Shortest Remaining Time)

C'est la version préemptive de l'algorithme SJF. Au début de chaque quantum, le CPU est attribué au processus qui a le plus petit temps d'exécution restant.



Exemple : Le temps restant le plus court (SRT : Shortest Remaining Time)

On prend un quantum = 1.

PID	Heure d'arrivée	Durée d'exécution	Début d'exécution	Fin d'exécution	Temps de traitement
A	0	5	0	10	10
B	1	2	1	3	2
C	2	5	10	15	13
D	3	3	3	6	3

Tableau 3 : SRT

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	B	B	D	D	D	A	A	A	A	C	C	C	C	C

Tableau 4 : Déroulement de SRT

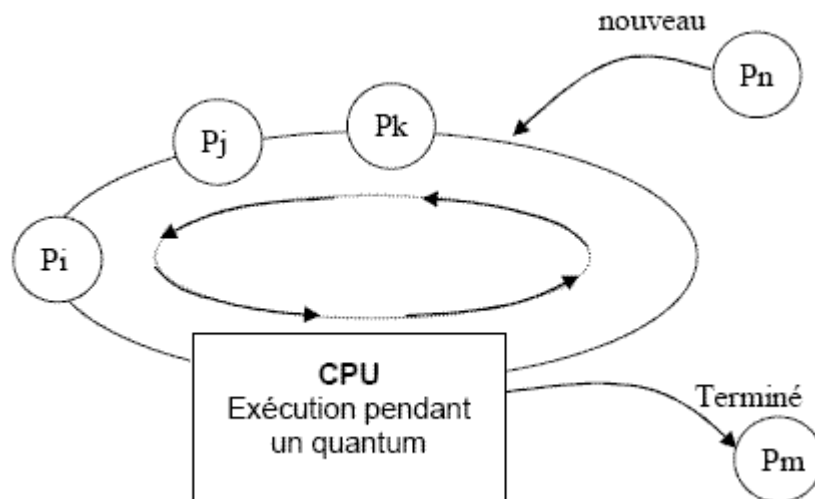
Temps de traitement moyen = 7.

Les travaux longs peuvent être victimes de famine.



Méthode : Algorithme à Tourniquet (RR : Round Robin)

Le contrôle du CPU est attribué à chaque processus pendant une tranche de temps Q , appelée quantum, à tour de rôle. Lorsqu'un processus s'exécute pendant un quantum, le contexte de ce dernier est sauvegardé et le CPU est attribué au processus suivant dans la liste des processus prêts (Figure (cf. 'Algorithme à tourniquet (Round Robin)' p 10)).



Algorithme à tourniquet (Round Robin)

Les performances de cette politique dépendent de la valeur du quantum. Un quantum trop grand augmente le temps de réponse, un quantum trop petit multiplie les commutations de contexte, qui, à partir d'une certaine fréquence, ne peuvent plus être négligées.



Exemple : Tourniquet

PID	Heure d'arrivée	Durée d'exécution	Début d'exécution	Fin d'exécution	Temps de traitement
A	0	30			
B	0,2	5			
C	0,5	2			

Tableau 5 : RR

Avec un quantum = 1 : temps de traitement moyen = $(37+11.8+5.5) / 3 = 18.1$

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
A	B	C	A	B	C	A	B	A	B	A	B	A	A	A	A	A	A	A

19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36
A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A

Tableau 6 : RR avec $q=1$

Avec un quantum = 10 : temps de traitement moyen = $(37+14.8+16.5) / 3 = 22.76$

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
A	A	A	A	A	A	A	A	A	A	B	B	B	B	B	C	C	A	A

19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36
A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A

Tableau 7 : RR avec $q=10$ 

Méthode : Algorithme avec priorité

Le système associe à chaque processus une priorité : nombre mesurant l'importance du processus dans le système. Le CPU est attribué au processus de plus haute priorité. En cas d'égalité, l'algorithme FCFS est utilisé. L'ordonnancement des priorités peut être avec ou sans réquisition.



Exemple : Algorithme avec priorité

On suppose dans cet exemple que la valeur la plus petite correspond à la priorité la plus élevée.

PID	Temps d'arrivée	Durée d'exécution	Priorité
A	0	10	3
B	0	1	1
C	10	2	2
D	0	1	4
E	0	5	2

Tableau 8 : Algorithme avec priorité

Sans réquisition:

1	6	16	18	19
B(1)	E(2)	A(3)	C(2)	D(4)

Avec réquisition:

1	6	10	12	18	19
B(1)	E(2)	A(3)	C(2)	A(3)	D(4)

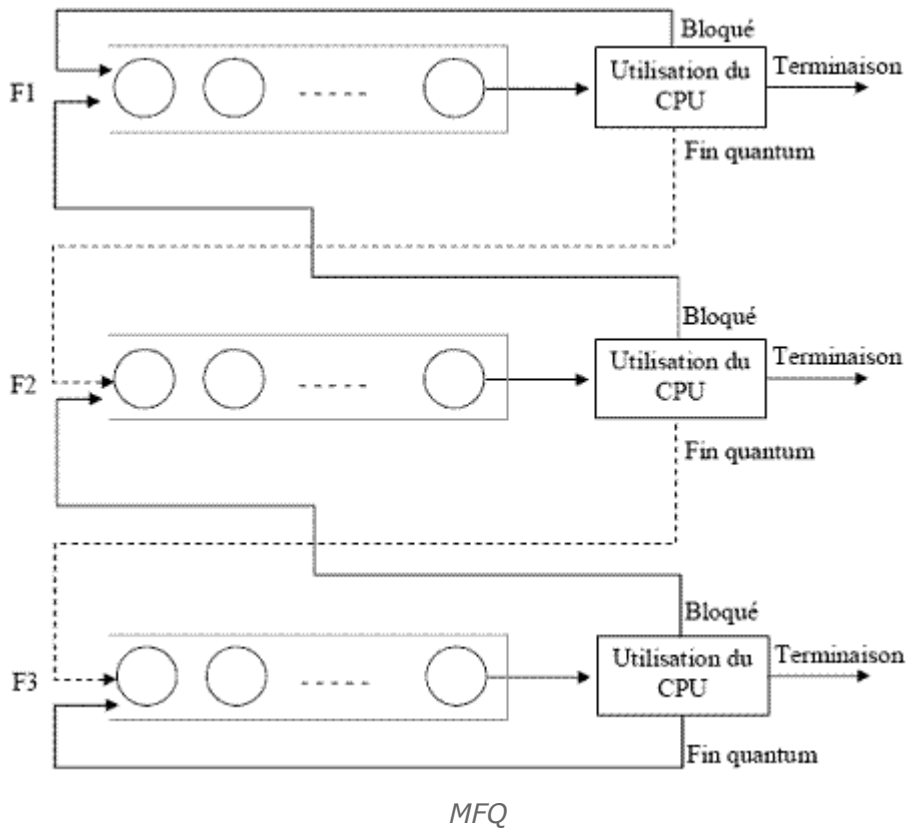
Tableau 9 : Déroulement algorithme avec priorité



Méthode : Algorithme à files multi-niveaux à retour (MFQ : Multilevel Feed-back Queues)

MFQ met en oeuvre plusieurs files d'attente dépendantes. Ces files peuvent être gérées de plusieurs façons donnant une panoplie d'algorithmes. Dans le cas des files MFQ, les processus demandeurs du processeur central sont rangés dans N ($N \geq 2$) files F_1, F_2, \dots, F_n . A chaque file est associé un quantum de temps Q_i tel que $Q_i < Q_{i+1}$. Un processus entrant est inséré dans la file F_1 . Un processus situé en tête de la file F_i ($i > 1$) ne sera pris en compte que si toutes les files F_j ($0 < j < i$) sont vides. Si un processus pris dans la file F_i ne s'est pas terminé à la fin de son compte, il sera placé dans la file F_{i+1} , sauf pour F_n où il y restera jusqu'à sa

terminaison. Par contre, un processus qui se bloque pour une E /S sera placé dans la file supérieure où il sera plus prioritaire (Figure (cf. 'MFQ' p 12)).



Complément : Ordonnancement de processus sous UNIX

L'ordonnanceur d'UNIX associe une priorité dynamique à chaque processus, recalculée périodiquement, et il utilise la stratégie du tourniquet (Round Robin) pour les processus de même priorité [rifflet03].

La fonction de calcul de la priorité courante utilise les paramètres suivants :

- une priorité de base correspondant au niveau d'interruption du processus;
- le temps CPU consommé récemment;
- une priorité ajustable par le processus lui-même grâce à la primitive nice:

```
#include<unistd.h>
int nice(int incr)
```

qui permet d'augmenter la valeur de la priorité de incr et donc de diminuer sa priorité. La valeur de incr doit appartenir à [0-39]. Seul le super-utilisateur peut augmenter la priorité en donnant une valeur négative à incr.