CHAPITRE III : GESTION DES PROCESSUS

3.1 CONCEPT DE PROCESSUS :

On peut trouver plusieurs appellations possibles des activités que peut avoir un processeur. Un système de traitement par lots exécute des *travaux*, tandis que un système en temps partagé possède des *programmes utilisateurs* ou des *tâches*.

3.1.1 Définition :

On peut définir un processus (process) comme un programme en exécution. Autrement dit, un programme par lui-même n'est pas un processus. Un programme est une entité *passive*, comme le contenu d'un fichier stocké sur disque, tandis qu'un processus est une entité *active*, avec un compteur d'instructions spécifiant l'instruction suivante à exécuter et un ensemble de ressources associées.

Evidemment, il est possible d'avoir plusieurs processus différents associés à un même programme. C'est le cas , par exemple, de plusieurs utilisateurs qui exécutent chacun une copie du programme de messagerie. Il est également courant qu'un processus génère à son tour plusieurs processus lors de son exécution.

3.1.2 Etats d'un processus :

Durant son séjour dans un SE, un processus peut changer d'état à plusieurs reprises. Ces états peuvent être résumés dans le schéma suivant :

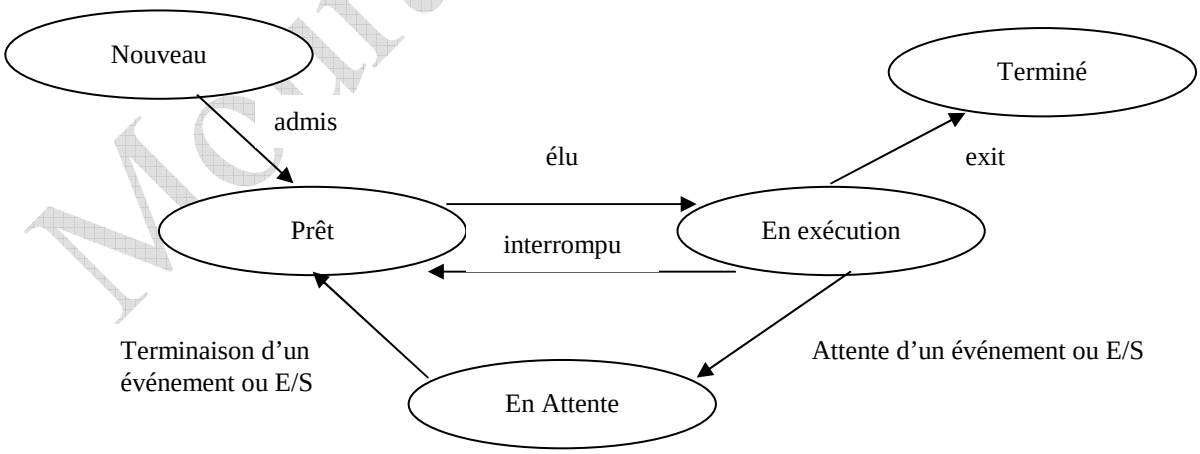


Fig. 3.1 Etats d'un processus.

Voici une brève explication de chacun de ces états :

- *Nouveau* : Le processus vient d'être créé.
- *Prêt* : le processus est placé dans la file d'attente des processus prêts, en attente d'affectation du processeur.
- *En exécution* : le processus a été affecté à un processeur libre. Ses instructions sont en exécution.
- *En attente* : Le processus attend qu'un événement se produise, comme l'achèvement d'une opération d'E/S ou la réception d'un signal.
- *Terminé* : le processus a terminé son exécution.

3.1.3 Bloc de contrôle de processus :

Pour suivre son évolution, le SE maintient pour chaque processus une structure de données particulière appelée bloc de contrôle de processus (PCB : Process Control Bloc) et dont le rôle est de reconstituer tout le contexte du processus.

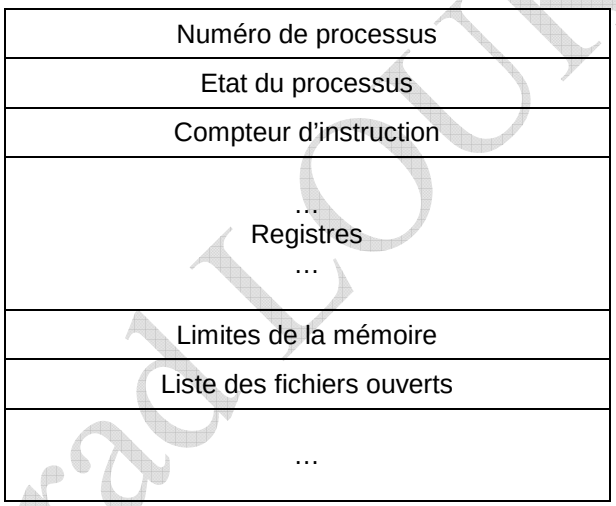


Fig. 3.2 Bloc de contrôle de processus.

Les informations contenues dans le PCB, et permettant de reconstituer le contexte d'un processus, sont multiples. On peut citer entre autres :

- *L'état du processus* : Il peut avoir l'une des valeurs suivantes : Nouveau, Prêt, en exécution ou en attente.
- *Le compteur d'instructions* : Le compteur indique l'adresse de la prochaine instruction à exécuter par le processus.
- *Les registres du processeur* : Les registres varient en nombre et en type en fonction de l'architecture de l'ordinateur. Ils englobent des accumulateurs, des registres d'index, des pointeurs de pile et des registres à usage général. Ces informations doivent être sauvegardées avec le compteur d'instructions quand il se produit une interruption, afin de permettre au processus de poursuivre correctement son exécution après la reprise.
- *Informations sur le scheduling du processeur* : Ces informations comprennent la priorité du processus, les pointeurs sur les files d'attente de scheduling, …

- *Informations sur la gestion de la mémoire* : Ces informations peuvent inclure les valeurs de registres de base et limites, les tables de pages ou les tables de segments selon le système de mémoire utilisé.
- *Informations sur l'état des E/S* : Les fichiers ouverts, la liste des périphériques d'E/S.

3.2 SCHEDULING DE PROCESSUS :

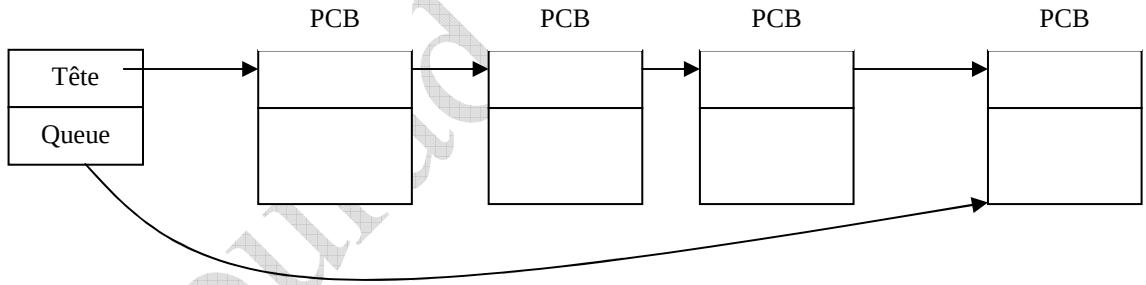
3.2.1 Files d'attentes de scheduling :

Pour gérer les processus durant leur séjour, le SE maintient plusieurs files d'attente. On peut citer entre autres :

- *File d'attente des processus prêts* : Cette file contient tous les processus en attente du processeur.
- *File d'attente de périphérique* : Pour réguler les demandes d'allocation des différents périphériques, on peut imaginer une file d'attente pour chaque périphérique. Quand un processus demande une opération d'E/S, il est mis dans la file d'attente concernée.

Concrètement une file d'attente est représentée par une liste chaînée de PCB, comme le montre le schéma suivant.

File d'attente des processus prêts :



File d'attente d'un périphérique :

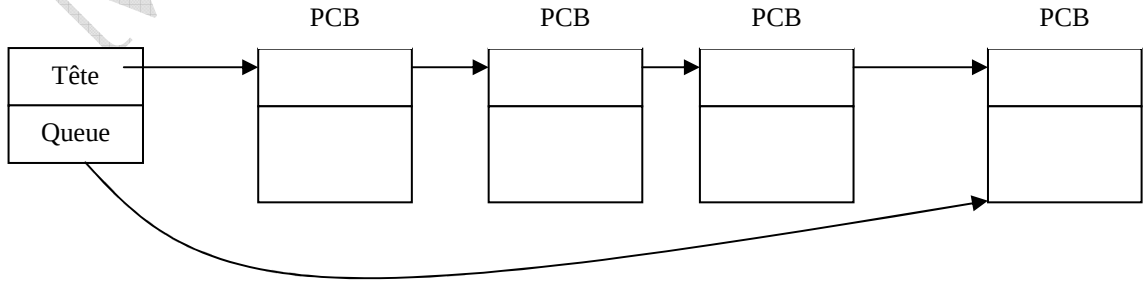


Fig. 3.3 Files d'attente de scheduling.

Un nouveau processus est initialement placé dans la file d'attente des processus prêts. Il attend dans cette file jusqu'à ce qu'il soit sélectionné pour son exécution et qu'on lui accorde le processeur. Une fois qu'on a alloué le processeur au processus et que celui-ci est en cours d'exécution, il pourrait se produire l'un des événements suivants :

- Le processus pourrait émettre une requête d'E/S et ensuite placé dans une file d'attente d'E/S.
- Le processus pourrait créer un nouveau processus et attendre la fin de celui-ci.
- Le processus pourrait être enlevé du processeur ; on dit aussi que le processeur a été réquisitionné. Dans ce cas, le processus est remis dans la file d'attente des processus prêts.
- Le processus pourrait se terminer.

Les différents états de transition du processus entre les files d'attente sont résumés par la figure suivante.

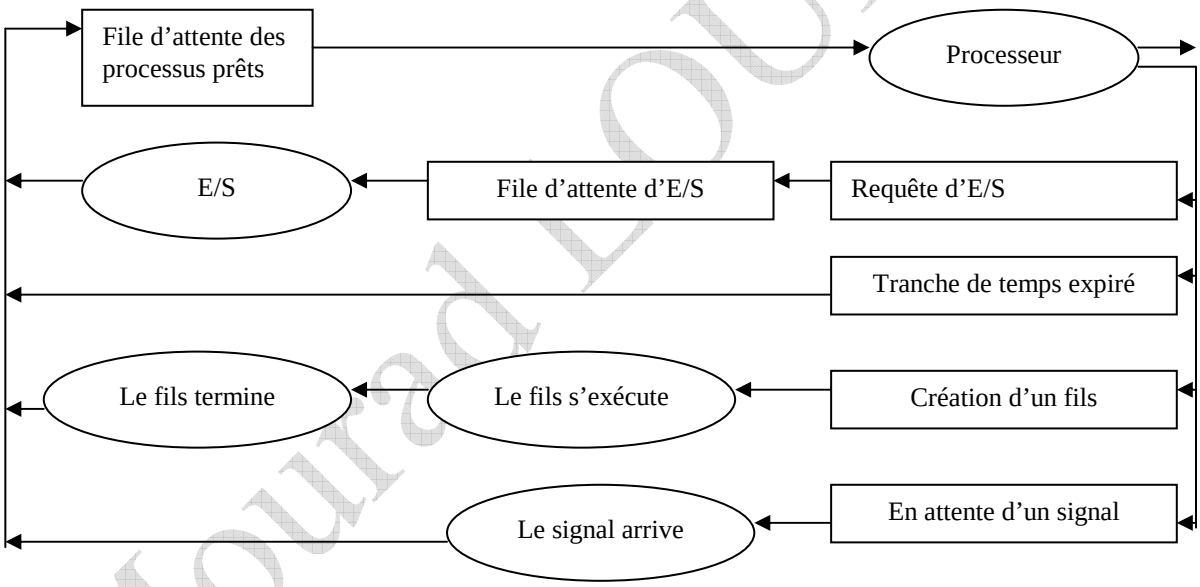


Fig. 3.4 Files d'attente de scheduling des processus.

3.2.2 Le scheduler :

Le scheduler est un programme du SE qui s'occupe de choisir , selon une politique de schedulind donnée, un processus parmi les processus prêts pour lui affecter le processeur.

3.2.3 Commutation de contexte :

Le fait de commuter le processeur sur un autre processus demande de sauvegarder l'état de l'ancien processus et de charger l'état sauvegardé par le nouveau processus. Cette tâche est connue sous le nom de *commutation de contexte*.

3.2.4 Les critères de scheduling :

Les divers algorithmes de scheduling du processeur possèdent des propriétés différentes et peuvent favoriser une classe de processus plutôt qu'une autre. Pour choisir quel algorithme utiliser dans une situation particulière, nous devons tenir compte des propriétés des divers algorithmes.

Plusieurs critères ont été proposés pour comparer et évaluer les performances des algorithmes de scheduling du processeur. Les critères les plus souvent utilisés sont :

- *Utilisation du processeur* : Un bon algorithme de scheduling sera celui qui maintiendra le processeur aussi occupé que possible.
- *Capacité de traitement* : C'est la quantité de processus terminés par unité de temps.
- *Temps de restitution* : C'est le temps s'écoulant entre la soumission du travail et sa terminaison.
- *Temps d'attente* : C'est le temps passé à attendre dans la file d'attente des processus prêts.
- *Temps de réponse* : C'est le temps passé dans la file d'attente des processus prêts avant la première exécution.

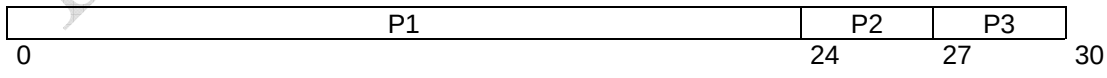
3.3 LES ALGORITHMES DE SCHEDULING :

3.3.1 L'algorithme du Premier Arrivé Premier Servi (FCFS) :

L'algorithme de schedulind du processeur le plus simple est l'algorithme du *Premier Arrivé Premier Servi* (First Come First Served : FCFS). Avec cet algorithme, on alloue le processeur au premier processus qui le demande. L'implémentation de la politique FCFS est facilement gérée avec une file d'attente FIFO. Quand un processus entre dans la file d'attente des processus prêts, son PCB est enchaînée à la queue de la file d'attente. Quand le processeur devient libre, il est alloué au processeur en tête de la file d'attente.

Exemple : Trois processus P1, P2 et P3 arrivent dans cet ordre au système. Leurs durées d'exécution sont respectivement : 24, 3, 3 unités de temps. Pour représenter l'historique d'occupation du processeur, on utilise le diagramme suivant, appelé *diagramme de Gantt* :

Diagramme de Gantt :

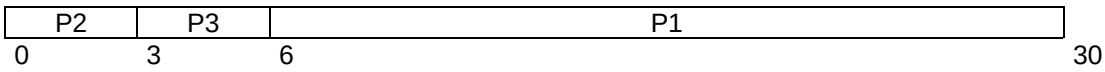


Ce diagramme montre que le processus P1 occupe le processeur de l'instant 0 jusqu'à l'instant 24. A l'instant 24, le processeur devient occupé par le processus P2, puis à l'instant 27 il sera suivi du processus P3.

Le temps d'attente est égal à 0 pour le processus P, 24 pour le processus P2 et 27 pour le processus P3. Le temps d'attente moyen est égal à : (0+24+27)/3, soit 17 unités de temps.

Si les processus étaient arrivés dans l'ordre P2, P3 et P1, les résultats seraient différents :

Diagramme de Gantt :



Le temps moyen d'attente serait : (0+3+6)/3=3 unités.

Ainsi le temps moyen d'attente avec une politique FCFS n'est généralement pas minimal et peut varier substantiellement si les durées d'exécution des processus varient beaucoup.

Critique de la méthode :

La méthode FCFS tend à pénaliser les travaux courts : L'algorithme du FCFS n'effectue pas de réquisition. C'est à dire qu'une fois que le processeur a été alloué à un processus, celui-ci le garder jusqu'à ce qu'il le libère, soit en terminant, soit après avoir demandé une E/S.

L'algorithme FCFS est particulièrement incommode pour les systèmes à temps partagé, où il est important que l'utilisateur obtienne le processeur à des intervalles réguliers. Il peut paraître désastreux de permettre qu'un processus garde le processeur pendant une période étendue.

3.3.2 L'algorithme du Plus Court d'abord (SJF) :

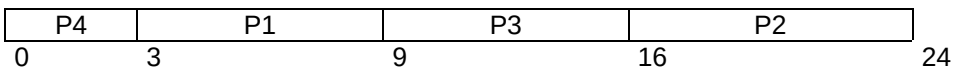
Cet algorithme (en anglais Shortest Job First : SJF) affecte le processeur au processus possédant le temps d'exécution le plus court. Si plusieurs processus ont la même durée, une politique FIFO sera alors utilisée pour les départager.

Exemple : On soumet au système quatre processus P1, P2, P3 et P4 dont les durées d'exécution sont données par le tableau suivant :

Processus	Durée d'exécution
P1	6
P2	8
P3	7
P4	3

L'algorithme du travail le plus court donnera alors le résultat suivant :

Diagramme de Gantt :



Le temps moyen d'attente est = (0+3+9+16)/4=7. Alors que si on avait choisi une politique FCFS, le temps moyen serait de : 10.25 unités de temps.

Critique de la méthode :

Il a été prouvé que l'algorithme SJF est optimal dans le temps dans le sens qu'il obtient le temps d'attente le plus court pour un ensemble de processus donné. Toutefois, cet algorithme est difficile à implémenter pour une raison simple : Comment peut-on connaître le temps d'exécution d'un processus à l'avance ?.

3.3.3 Scheduling avec priorité :

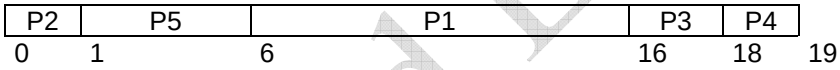
Cet algorithme associe à chaque processus une priorité, et le processeur sera affecté au processus de plus haute priorité. Cette priorité varie selon les systèmes et peut aller de 0 à 127.

Les priorités peuvent être définies en fonction de plusieurs paramètres : le type de processus, les limites de temps, les limites mémoires, …etc.

Exemple : On dispose de 5 processus ayant des priorités différentes, comme le montre ce tableau :

Processus	Durée d'exécution	Priorité
P1	10	2
P2	1	4
P3	2	2
P4	1	1
P5	5	3

Diagramme de Gantt :



Le temps moyen d'attente est = (0+1+6+16+18)/5=8.2 unités de temps.

Critique de la méthode :

Une situation de blocage peut survenir si les processus de basse priorité attendent indéfiniment le processeur, alors que des processus de haute priorité continuent à affluer.

Pour éviter une telle situation, on peut utiliser la technique dite du *vieillissement*. Elle consiste à incrémenter graduellement la priorité des processus attendant dans le système pendant longtemps. Par exemple, nous pourrions incrémenter de 1 la priorité d'un processus en attente toutes les 15 minutes. En fin de compte, même un processus ayant une priorité initiale égale à 0 aurait la plus haute priorité dans le système et serait exécuté.

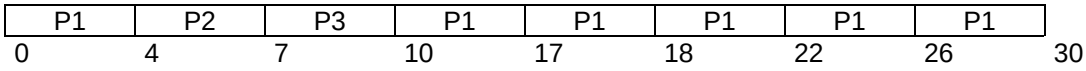
3.3.4 L'algorithme de Round Robin (Tourniquet) :

L'algorithme de scheduling du tourniquet, appelé aussi **Round Robin**, a été conçu pour des systèmes à temps partagé. Il alloue le processeur aux processus à tour de rôle , pendant une tranche de temps appelée **quantum**.

Dans la pratique le quantum s'étale entre 10 et 100 ms.

Exemple : On dispose de 3 processus P1, P2 et P3 ayant comme durée d'exécution, respectivement 24, 3 et 3 ms. En utilisant un algorithme Round Robin, avec un quantum de 4 ms, on obtient le diagramme de Gantt suivant :

Diagramme de Gantt :



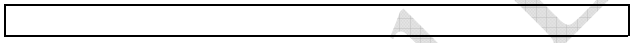
Le temps moyen d'attente est de : (6+4+7)/3 = 17/3 = 5.66 ms

La performance de l'algorithme de Round Robin dépend largement de la taille du quantum. Si le quantum est très grand , la politique Round Robin serait similaire à celle du FCFS. Si le quantum est très petit, la méthode Round Robin permettrait un partage du processeur : Chacun des utilisateurs aurait l'impression de disposer de son propre processeur .

Cependant le quantum doit être choisi de sorte à ne pas surcharger le système par de fréquentes commutations de contexte.

Exemple : On dispose d'un processus P dont le temps d'exécution est de 10 ms. Calculons le nombre de commutations de contexte nécessaires pour un quantum égal respectivement à : 12, 6 et 1.

Quantum=12

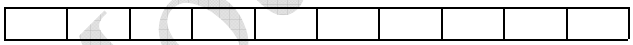


Nombre de commutations de contexte = 1

Quantum=6

Nombre de commutations de contexte = 2

Quantum=1



Nombre de commutations de contexte = 9

3.3.5 Scheduling avec files d'attente multiniveaux :

Une autre classe d'algorithmes de scheduling a été développée pour des situations où on peut facilement classer les processus dans des groupes différents. Par exemple, il serait intéressant de faire une distinction entre les processus de *premier plan* (interactifs) et les processus *d'arrière plan* (traitement par lot). En effet, ces deux types de processus possèdent des besoins différents en ce qui concerne le temps de réponse et ils pourraient donc devoir être schedulés différemment. De plus les processus de premier plan peuvent être prioritaires par rapport aux processus d'arrière plan.

Ainsi, un algorithme de scheduling avec des files d'attente multiniveaux découpe la file d'attente des processus prêts en plusieurs files d'attentes séparées. La figure suivante doonné un exemple de découpage de ces files d'attente :

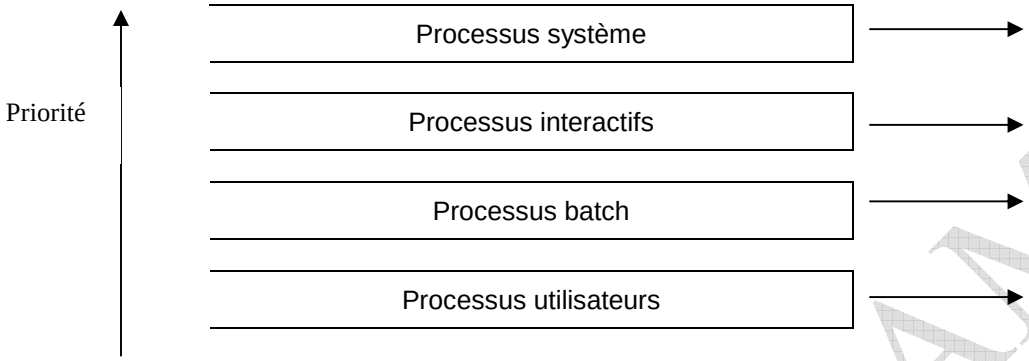


Fig. 3.5 Sheduling avec files d'attente multiniveaux.

Les processus sont en permanence assignés à une file d'attente en se basant généralement sur certaines propriétés du processus, comme : le type de processus, la taille de la mémoire, la priorité, … etc. Chaque file d'attente possède son propre algorithme de scheduling. Par exemple , la file d'attente des processus systèmes est schedulée selon algorithme de plus haute priorité, celles des processus interactifs est gérée selon l'algorithme Round Robin et la file des processus batch est gérée selon l'algorithme FCFS.

D'autre part, il doit y avoir un scheduling entre les files d'attente elles-même.

Observons par exemple, l'ensemble des files d'attente multiniveaux suivants :

1. Processus systèmes
2. Processus interactifs
3. Processus batch
4. Processus utilisateurs

Chaque file d'attente est absolument prioritaire par rapport aux files d'attente de niveau inférieur. Par exemple, aucun processus de la file d'attente des processus batch ne pourra s'exécuter à moins les files d'attente des processus système et interactifs ne soient toutes vides. De plus, si un processus interactif arrive au système, alors qu'un processus batch est en train de s'exécuter, celui-ci doit être interrompu.

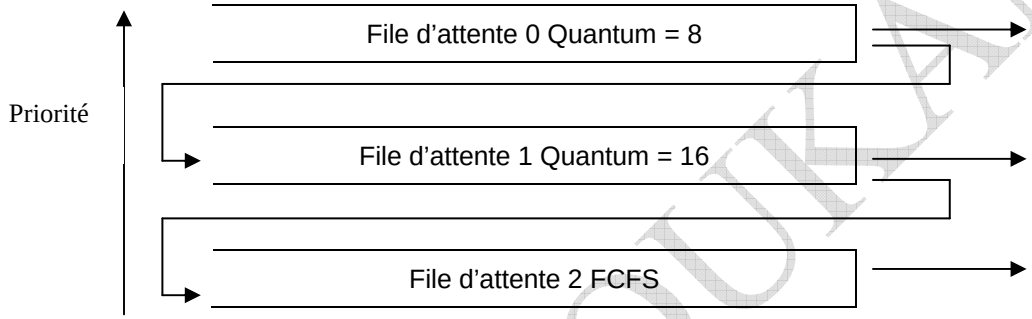
Une autre manière de procéder serait d'affecter des tranches de temps aux files d'attente. Chaque file d'attente obtient une certaine partie du temps processeur, lequel doit se scheduler entre les différents processus qui la composent. Par exemple, on peut attribuer 80% du temps processeur à la file d'attente des processus de premier plan et 20% pour la file d'attente des processus d'arrière plan.

3.3.6 Scheduling avec files d'attente multiniveaux et feedback :

Normalement, dans un algorithme avec des files d'attente multiniveaux, les processus sont assignés en permanence à une file d'attente dès qu'ils rentrent dans le système. Les processus ne se déplacent pas entre les files d'attente. Cette organisation possède l'avantage d'une basse surcharge due au scheduling, mais elle manque de souplesse.

Ainsi, le scheduling avec des files d'attente feedback multiniveaux permet aux processus de se déplacer entre les files d'attente. L'idée revient à séparer les processus en fonction de l'évolution de leurs caractéristiques dans le système.

Exemple : Un système est doté de 3 files d'attentes mutiniveaux : File 0, File 1 et File 2. La file 0 est la plus prioritaire. Les files 0 et 1 sont gérées selon la politique Round Robin. La file 2 est gérée selon la technique FCFS.



Un processus entrant dans le système sera rangé dans la file d'attente 0. On donne une tranche de temps de 8 ms au processus. S'il ne finit pas, il est déplacé vers la file d'attente 1. Si la filme d'attente 0 est vide, on donne une tranche de temps de 16 ms au processus en tête de la file 1. S'il ne termine pas, il est interrompu et il est mis dans la file d'attente 2. Les processus de la file d'attente 2 sont exécutées seulement quand les files 0 et 1 sont vides.