Matière: systèmes d'exploitation 1 **(Operating System)**
Chapitre 2 : Gestion du processeur central

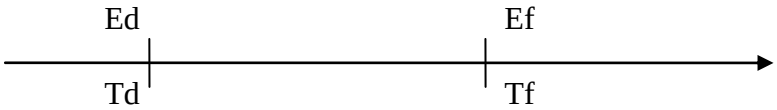
1 Notion d'événement

L'état de la mémoire est défini par le contenu des emplacements correspondants. On considère l'exécution d'une opération comme **indivisible** et **atomique**, c'est-à-dire qu'on s'interdit d'observer l'état de la machine (processeur + mémoire) pendant l'exécution de cette opération.

L'état de la machine peut être observé soit au début, soit à la fin de cette exécution. Soient deux variables **td** et **tf** :

Td : temps au début de l'opération. **Tf** : temps à la fin de l'opération.

Les instants **td** et **tf** sont appelés **Points d'observation**. Et les états correspondants (**ed, ef**) sont appelés **Poins observables.**



Les points d'observations permettent d'associer une date (td, tf) à un état (ed, ef) de la machine on appelle cette association (état - daté) **évènement**.

L'exécution d'une opération ou d'une instruction d'un programme peut être représentée par deux évènements début (instruction) et fin (instruction). On note la relation d'ordre sur un évènement par : Début (instruction) < Fin (instruction)

2 Définition de processus

De manière informelle, un processus est l'exécution séquentielle d'un programme. On peut dire qu'un processus est une représentation abstraite d'un programme.

Un processus est une entité dynamique alors qu'un programme est une entité statique qui peut donner naissance à plusieurs processus.

D'une manière générale un processus peut être décrit par une suite d'évènements qui représente sa trace temporelle.

Une propriété fondamentale que doivent avoir les processus est : Les résultats des processus doivent être indépendants de leur vitesse d'exécution.

3 Etats d'un processus

La période d'existence d'un processus est celle s'écoulant entre l'instant de sa création (naissance) et l'instant de sa destruction (mort). Durant son existence, tout processus peut passer par une série d'états. Les principaux états sont :

Etat actif (élu) : le processus dispose de toutes les ressources y compris le processeur (quand le processeur est en train d'exécuter une de ses instructions.)

Etat prêt (éligible) : le processus dispose de toutes les ressources sauf le processeur.

Bloqué : le processus ne peut plus progresser, il est en attente d'un signal (par exemple un signal de fin d'E/S) ou d'une ressource autre que le processeur.(quand il est en attente d'un évènement, par exemple une lecture de données).

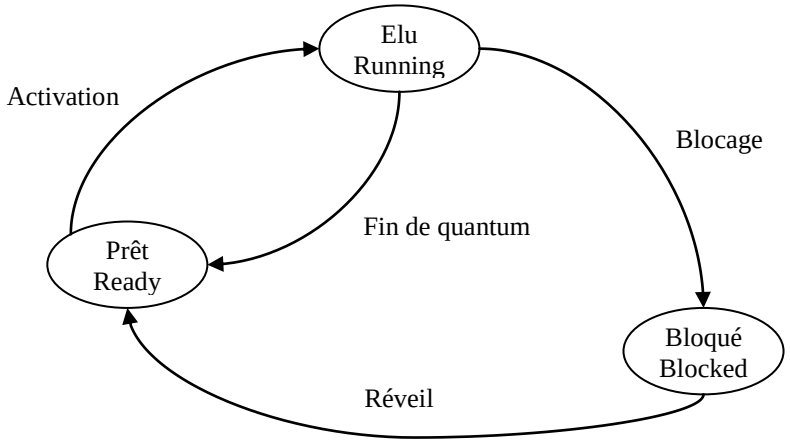


Figure1 : Etats d'un processus

Si l'on considère que le processeur est une ressource de même type que les autres, on n'aura alors que deux états : actif ou bloqué.

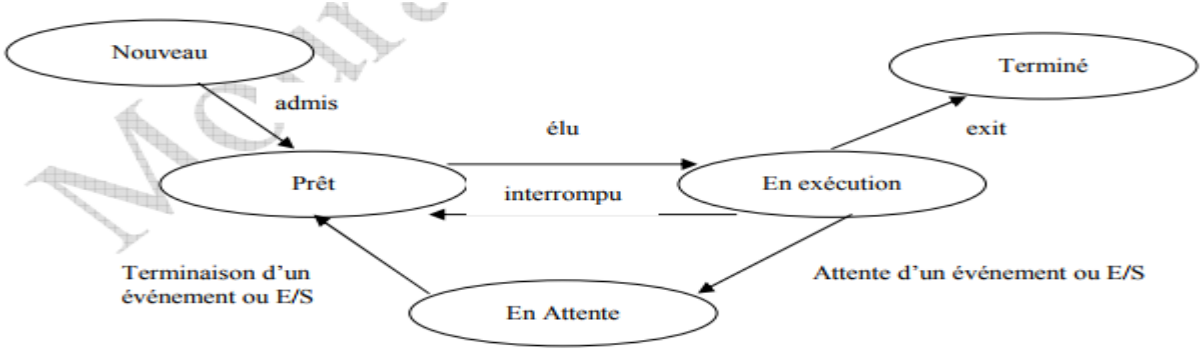


Fig. 3.1 Etats d'un processus.

3.1 Changements d'état

1. Quand le processus actif a besoin de faire une opération d'entrée-sortie (E/S), il en fait la demande auprès du système d'exploitation par un \appel système" (sous Unix : read, write, etc.).

S'il faut attendre la fin de cette opération pour continuer (par exemple pour une lecture), le système change l'état du processus, qui devient bloqué. On parle d'opération bloquante.

2. lorsqu'un périphérique signale au processeur qu'une opération d'E/S est achevée, le système d'exploitation \débloque" le processus qui attendait la fin de cette opération. Le processus est alors marqué comme prêt.

3. enfin, le système d'exploitation peut choisir un processus prêt pour le rendre actif. Ce changement peut se faire quand le processus actif se termine ou se bloque, et \laisse sa place".

A un moment donné il peut y avoir plusieurs processus prêts : le choix du processus à activer est fait par un module appelé ordonnanceur (scheduler). Diverses politiques d'ordonnancement sont envisageables, nous les verrons en détail plus loin.

4 Bloc de contrôle d'un processus (PCB)

Le PCB permet de définir un processus dans le système (***le SE maintient pour chaque processus une structure de données particulière***).
Diverses implémentations existent selon les systèmes d'exploitation, mais un PCB contient en général:

- l'ID (N°) d'un processus (PID) ;
- Les valeurs des registres correspondant au processus (l'état courant du processus, selon qu'il est élu, prêt ou bloqué) ;
- Le compteur ordinal du processus ;

- D'autres informations telles que le temps CPU accumulé par le processus,
- Mot d'état du processus.
- Adresse de base de l'espace mémoire, limite de l'espace mémoire, ……..etc.

Lors d'un changement de contexte, le processus en cours est arrêté et un autre processus peut utiliser le CPU. Le noyau doit arrêter l'exécution du processus en cours, copier les valeurs des registres hardware dans le PCB, et mettre à jour les registres avec les valeurs du nouveau processus.

4.1 Descripteur de processus (PCB)

A la création d'un processus, son PCB est construit. le contenu du PCB varie d'un système à un autre suivant sa complexité. il peut contenir:

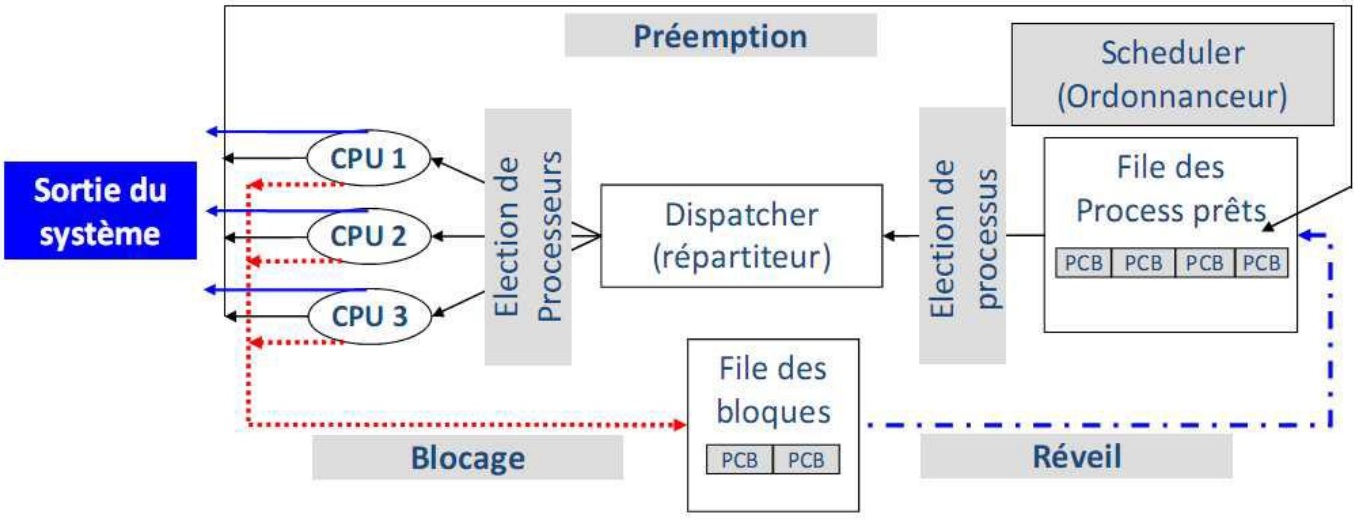
- **Identité du processus (ID)**: possède 2 noms
  1. un nom externe sous-forme de chaine de caractères fourni par l'usager (c'est le nom du fichier exécutable).
  2. un nom interne sous forme d'un entier fourni par le système. toute référence au processus à l'intérieur du système se fait par le nom interne.
- **Le compteur d'instructions** : Le compteur indique l'adresse de la prochaine instruction à exécuter par le processus.
- **Contexte du processus**: CO, mot d'état, registres, pile.
- **Priorité de scheduling**: une priorité peut être affectée à un processus de manière statique ou dynamique. dans le cas dynamique, la priorité changera selon le concepteur de la politique de scheduling adoptée.
- **Etat du processus**: actif, prêt, bloqué, suspendu.
- **Limites mémoires**: spécifient les zones mémoires pour lesquelles le processus peut accéder.

5 Ordonnancement des processus (Scheduling)

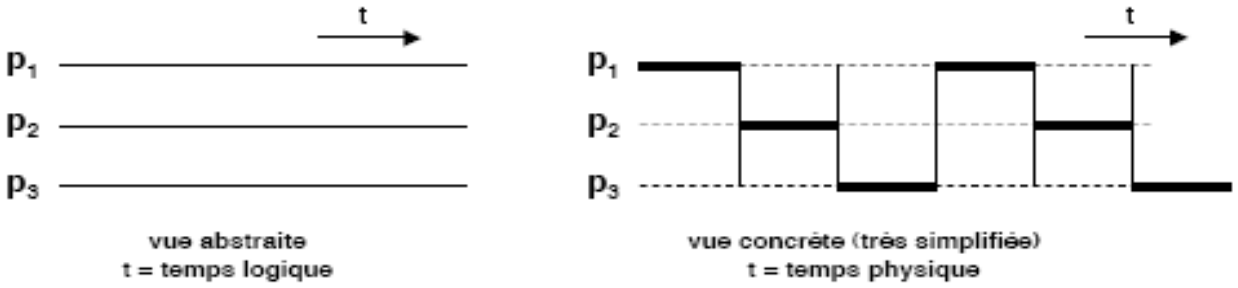
Dans un système multi-utilisateurs à temps partagé, plusieurs processus peuvent être présents en mémoire centrale en attente d'exécution. Si plusieurs processus sont prêts, le système d'exploitation doit gérer l'allocation du processeur aux différents processus à exécuter.

L'ordonnancement (scheduling) consiste donc à ***décider de l'ordre*** dans lequel les différents processus auront accès au CPU. C'est l'Ordonnanceur (scheduler) qui s'occupe de cette tâche. Il s'agit d'un module du SE qui sélectionne le prochain processus à exécuter.

La figure suivante montre le schéma général d'un scheduler :



L'allocation de processeur est réalisée par multiplexage (allocation successive aux processus pendant une tranche de temps fixée)



Dans le cadre des systèmes d'exploitation, le terme est en général employé uniquement pour le CPU, mais dans certains cas il peut être utilisé pour d'autres ressources.

Dans l'informatique en général, le terme et les principes du scheduling s'appliquent très souvent, on peut avoir un scheduler de requêtes dans un serveur Web, un scheduler d'accès aux tables dans une base de données, . . .

Le scheduling est d'ailleurs important aussi dans des problèmes de la vie courante, comme la gestion des files d'attente, l'attribution des salles d'opération dans un hôpital, . . .

Un Ordonnanceur fait face à ***deux problèmes*** principaux :

- ***Le choix du processus à exécuter, et***
- ***Le temps d'allocation du processeur au processus choisi.***

Les principaux objectifs assignés à un Ordonnanceur sont:

- S'assurer que chaque processus en attente d'exécution reçoive sa part de temps processeur.
- Minimiser le temps de réponse.
- Utiliser le processeur à 100%.
- Utilisation équilibrée des ressources.
- Prendre en compte des priorités.

5.1 Critères de performance des algorithmes d'allocation du processeur

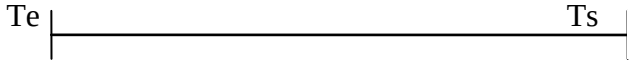
Il existe plusieurs algorithmes d'allocation, chacun d'eux a des propriétés spécifiques qui peuvent favoriser une classe de processus par rapport à une autre. Pour le choix d'un algorithme à utiliser dans une situation particulière on doit tenir compte des différentes propriétés de l'algorithme.

Les principaux critères de performances sont :

- **Utilisation du processeur central :** le taux d'utilisation de l'U.C doit être très élevé.
- **Débit des travaux :** c'est le nombre de travaux exécutés par unité de temps, ce taux est faible pour les travaux longs, par contre il doit être élevé pour les travaux court.
- **Temps de résidence *(séjour)* :** c'est le temps passé par le travail / processus dans le système. C'est la somme du temps d'exécution et du temps passé dans les différentes files (files des processus prêts, files des processus bloqués).

  Te : le temps d'entrée du processus dans le système

  Ts : le temps de sortie du processus.

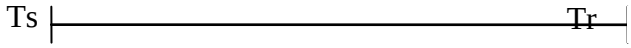


Temps de résidence = ts - te

- **Temps d'attente :** c'est le temps passé dans la file des processus prêts (et non bloqués); le temps d'attente des entrées / sorties n'est pas pris en compte.
- **Temps de réponse :** le temps de résidence n'est pas significatif pour les programmes interactifs, donc on utilise un autre critère de performance qui est le temps de réponse. C'est le temps qui sépare la soumission d'une requête et **la fin de la réponse** à cette requête

  Ts : le temps de soumission de la requête

  Tr : le temps de fin de la réponse à cette requête. ***Temps de réponse = tr – ts***



5.2 Stratégies d'allocation du processeur

On peut classer les stratégies d'allocation en deux catégories :

- ***Sans recyclage*** (***sans préemption***): ce sont les algorithmes utilisés dans les premiers systèmes.
- ***Avec recyclage (préemptives)*** : ce type d'algorithmes favorise les travaux courts et assure, dans les systèmes interactifs, un temps de réponse assez bref.

5.2.1 Algorithmes sans recyclage (*sans préemption*)

5.2.1.1 Premier arrivé premier servi (FIFO) First-Come First-Served FCFS

C'est un algorithme sans recyclage (sans réquisition) qui consiste à allouer le processeur au premier processus de la file des processus prêts. L'ordre de priorité correspond de façon naturelle à l'ordre d'arrivée.

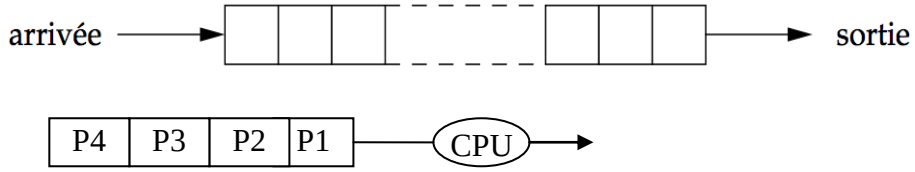
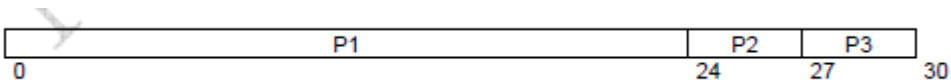


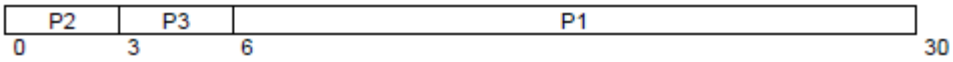
Figure3.1 : L'ordonnanceur FIFO.

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** :



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 :

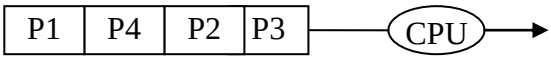


Inconvénient : les travaux courts risquent d'attendre longtemps s'ils sont après des travaux longs.

FIFO présente une faible variance ; il est donc plus prédictible que la plupart des autres techniques. De toute évidence, il n'est pas utilisable dans les systèmes interactifs car il ne garantit pas un bon temps de réponse. C'est en particulier une des principales raisons qui font que cette technique est rarement utilisée aujourd'hui « à l'état brut ». Néanmoins, il faut noter qu'elle peut être associée à une technique plus sophistiquée qui accordera des priorités générales, les processus de même priorité étant considérés selon un schéma FIFO. **[1]**

5.2.1.2 Le plus court travail d'abord (SJF : Shortest Job First)

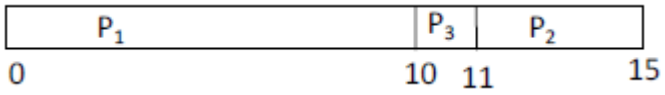
Le CPU est alloué au travail le plus court (temps CPU demandé) ; la priorité d'un processus est inversement proportionnelle au temps CPU demandé. (SJF est un cas particulier de la politique de scheduling par priorité p ou p=***Priorité = 1 / (temps CPU demandé (estimé))***



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

processus	Temps d'arrivé	Temps d'exécution
P1	0	10
P2	3	4
P3	5	1

Diagramme de Gantt



Le temps moyen d'attente est : (0 + 8 + 5)/3 = 4,33 unités. Alors que si on avait choisi une politique FCFS, le temps moyen serait de : (0+7+9) = 5,33 unités de temps.

Inconvénient :

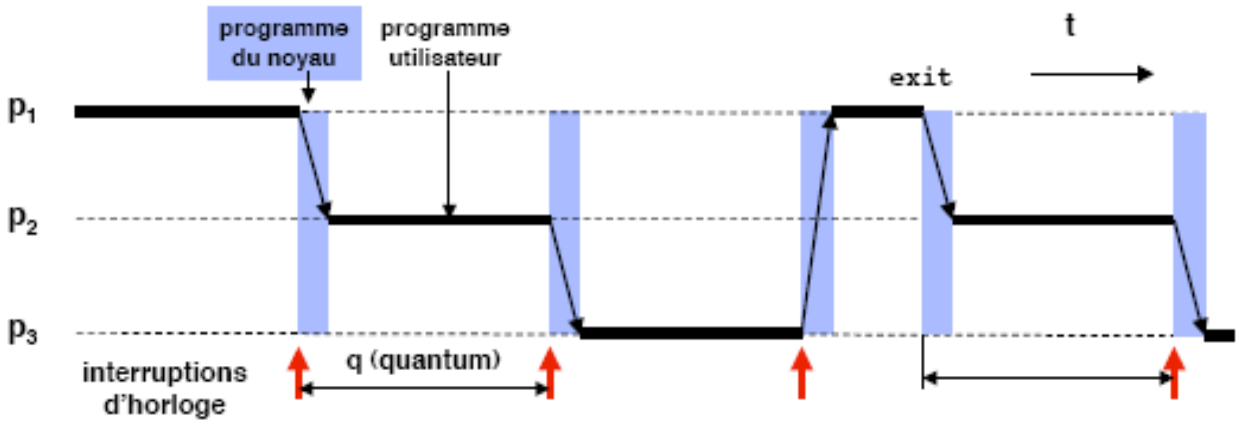
- Risque de privation pour les travaux longs.
- La mise en œuvre nécessite la connaissance préalable du temps d'exécution des jobs (Le seul moyen est de se fier à une estimation donnée par les utilisateurs eux-mêmes.)

5.2.2 Algorithmes avec recyclage *(préemptives)*

Dans les algorithmes précédents, quand le processeur est alloué à un processus il le garde jusqu'à ce qu'il lance une opération d'entrées / sorties ou il se bloque volontairement. Dans les algorithmes avec réquisition le processeur peut être retiré à un processus avant la fin de son exécution et affecté à un autre processus. Le processus choisi peut être plus prioritaire ou de priorité quelconque dans le cas où le processeur est alloué par quantum de temps.

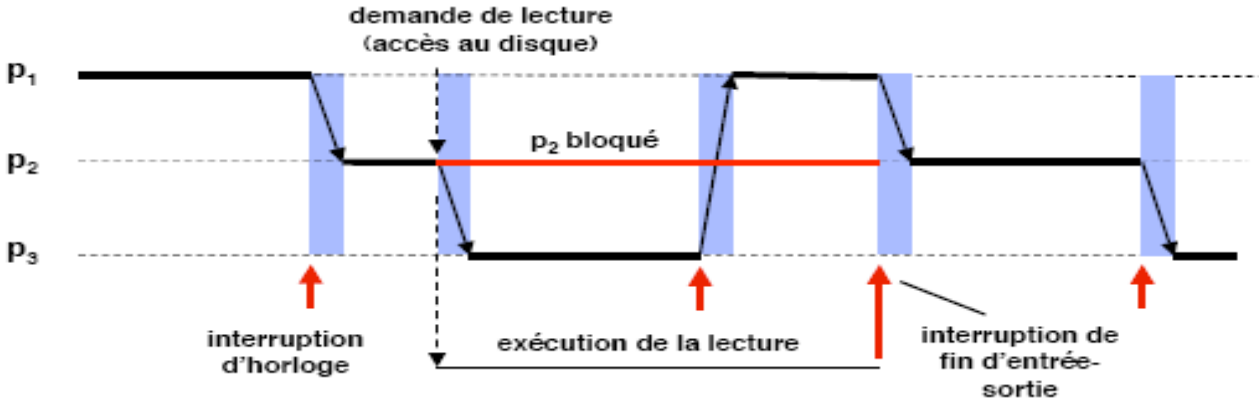
Le processeur est alloué par tranches de temps successives (quanta) aux processus prêts à s'exécuter (non bloqués). Dans la pratique, la valeur du quantum est d'environ 10 ms (temps d'exécution de quelques millions d'instructions sur un processeur à 1 GHz). La commutation entre processus est déclenchée par une interruption d'horloge. La

commutation entre processus prend un temps non nul (de l'ordre de 0,5 ms). Elle est réalisée par l'ordonnanceur (scheduler)



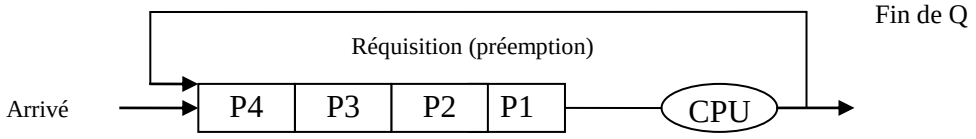
Lorsqu'un processus est bloqué (par exemple parce qu'il a demandé une entrée sortie, ou a appelé sleep), il doit libérer le processeur puisqu'il ne peut plus l'utiliser. Le processeur pourra lui être réalloué à la fin de sa période de blocage (souvent indiquée par une interruption)

Cette réallocation pourra se faire soit immédiatement (comme sur la figure) ou plus tard (après fin de quantum), selon la politique choisie.



5.2.2.1 Tourniquet (RR : Round – Robin) ou Ordonnancement circulaire

Cet algorithme est très utilisé dans les systèmes en ***temps partagé***. Le temps CPU est divisé en tranche de temps appelée quantum de temps ou 'time slice'. Le CPU est alloué au premier processus de la file des processus prêts pendant un quanta de temps ; si le processus n'a pas terminés son exécution, il est recyclé dans la file des processus prêts.



Le processeur est alloué à un autre processus (1er de la file) à la fin du quantum ou si le processus actif se bloque.

Le Schedule alloue le processeur au premier processus de la file des processus prêts, initialise l'horloge pour une interruption après un quantum de temps. Il en résultera une des deux situations:

1) Le temps d'exécution est inférieur au quantum: dans ce cas le processus libère volontairement le processeur central à la suite d'une entrée/sortie ou une terminaison. Le prochain processus de la file est élu (le processeur central lui est alloué).

2) Le temps d'exécution CPU est supérieur au quantum: dans ce cas, le processus consommera son quantum de temps et par conséquent, il y aura une interruption horloge. Les registres du processeur sont sauvegardés dans le bloc de contrôle du processus et il est mis à la queue de la file. Ensuite, le scheduler sélectionne un nouveau processus.

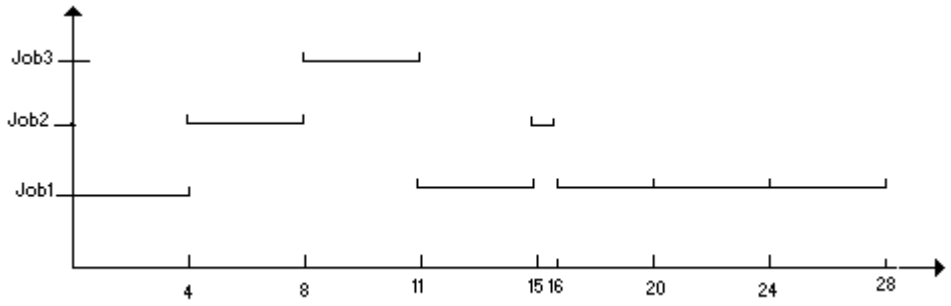
Les performances de cet algorithme dépendent du quantum de temps (Q).

- Si Q est très grand (tend vers l'infini) alors cet algorithme est équivalent à l'algorithme FIFO.
- Si Q est très petit (tend vers 0) alors il y a une mauvaise utilisation du processeur.
- La commutation de processus (overhead) dure un temps non nul pour la mise à jour des tables, la sauvegarde des registres. Un quantum ***trop petit*** provoque trop de commutations de processus et abaisse l'efficacité du processeur. Un quantum ***trop grand*** augmente le temps de réponse en mode interactif. On utilise souvent un quantum de l'ordre de 100 ms.

Exemple: soient les jobs suivants:

N job	Temps d'exécution
1	20
2	5
3	3

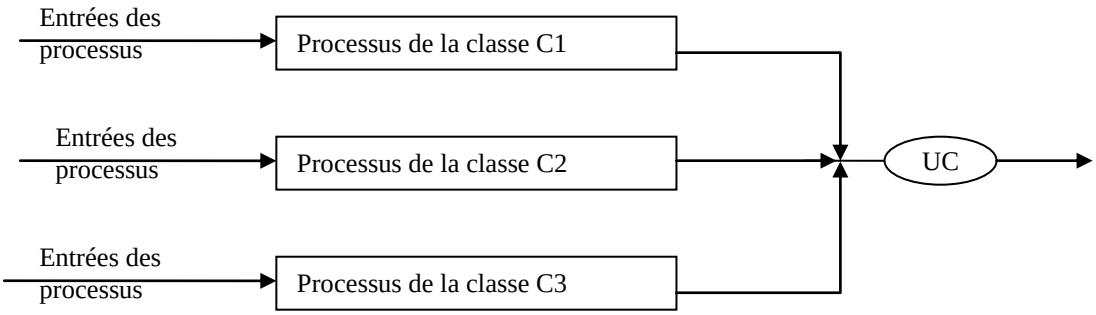
Si on utilise un quantum de temps égal à 4 unités de temps et un temps de commutation négligeable, alors l'algorithme de scheduling ROUND ROBBIN donnera ce qui suit:



$T_{\text{réponse moyenne}} = (11+16+28)/3 = 55/3$

5.2.2.2 Les files multi niveaux indépendantes

On définit des classes de processus et on associe à chacune des classes une ***priorité*** et une ***stratégie d'allocation***.

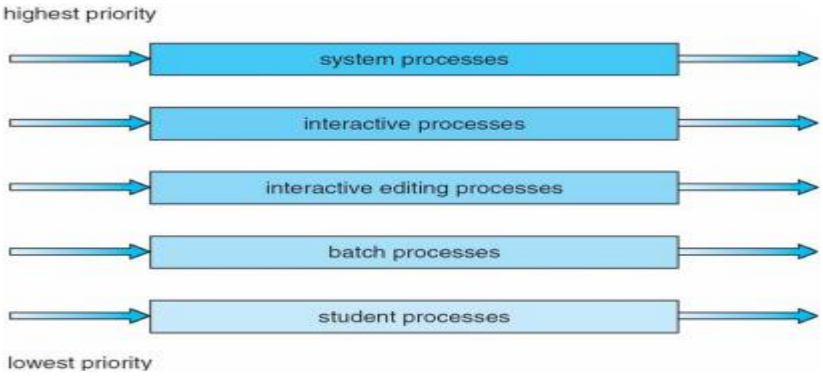


Le processeur est alloué aux processus de la classe la plus prioritaires. On ne change de file que si la file la plus prioritaire est vide.

Les processus peuvent être dans différentes catégories
– Exemple: processus en avant-plan (interactifs) et processus en arrière-plan (batch processes)
– Des exigences différentes en temps de réponses
– Besoins différents en ordonnancement.

• Avec un algorithme d'ordonnancement utilisant des **files multi niveaux**, la file des processus prêt est partitionnée en plusieurs files d'attentes séparées
• Chaque file a son propre algorithme d'ordonnancement
– La file des processus en avant plan : – RR
– La file des processus en arrière plan – FCFS

• En plus, il y a un ordonnancement entre les files d'attentes
– Typiquement : un ordonnancement préemptif avec priorités fixes; (i.e., servir tous les processus en avant plan avant les processus en arrière plan).
– Utiliser une tranche de temps : chaque file obtient une portion du temps de la CPU; P.ex. 80% pour les processus en avant plan avec RR et 20% pour les processus en arrière plan avec FCFS



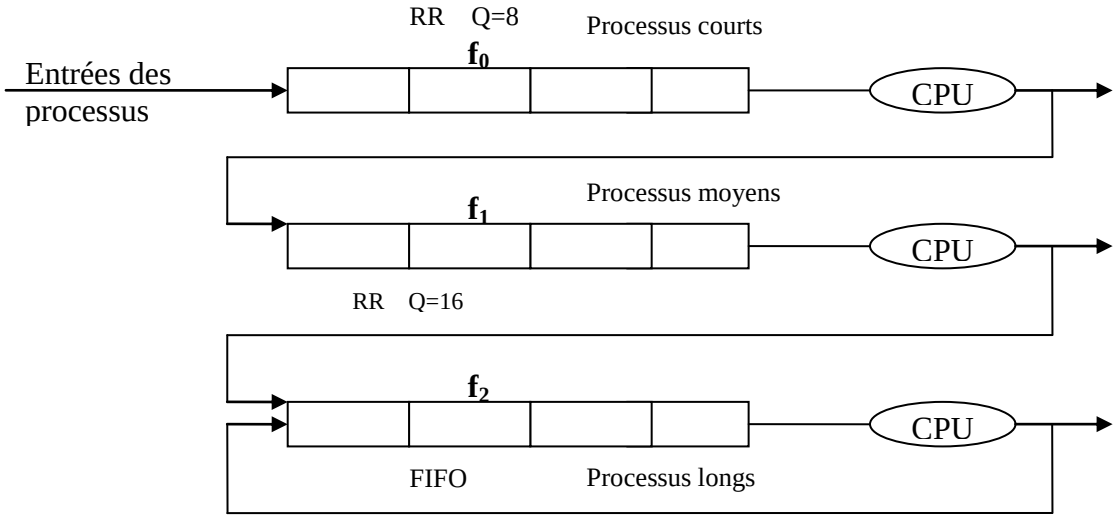
Files d'ordonnancement multi-niveaux [1]

5.2.2.3 Les files multi niveaux avec recyclage

Dans la technique précédente les processus ne pouvaient pas changer de file. Un processus d'une classe A reste dans la file de cette classe jusqu'à ce qu'il quitte le système.

Dans cette stratégie, on a n files de processus prêts : f_0, f_1, f_2,…….f_{n-1}

A chaque file f_i est associé un quantum de temps q_i dont la valeur croit avec le rang de la file. Un processus de la file f_i n'est servi que si les files de rang inférieur à i sont vides. Lorsqu'un processus de la file f_i a épuisé son quantum de temps sans avoir terminé son exécution, il rentre dans la file f_{i+1}. Les processus de la dernière file f_{n-1} sont recyclés dans la même file. Les nouveaux processus sont rangés dans la file f_0.



Remarque : cet algorithme favorise les processus courts sans avoir besoin de savoir à l'avance combien de temps CPU ceux-ci vont utiliser.

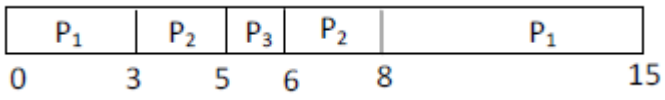
5.2.2.4 SRTF ou SRT plus court temps restant (Shortest Remaining Time First)

C'est un algorithme avec réquisition (avec préemptif) qui choisit le processus dont le temps d'exécution **restant** est le plus petit (en considérant à chaque instant les nouveaux arrivants). A chaque fin de quantum on diminue le temps d'exécution restant au processus qui est actif et on choisit le processus ayant le plus petit temps. A chaque entrée d'un nouveau processus on choisit le processus ayant le plus petit temps.

SRT est équivalent à la méthode SJF mais avec préemption.

Exemple : Reprenons l'exemple précédent (présenté pour la politique SJF):

Diagramme de Gantt

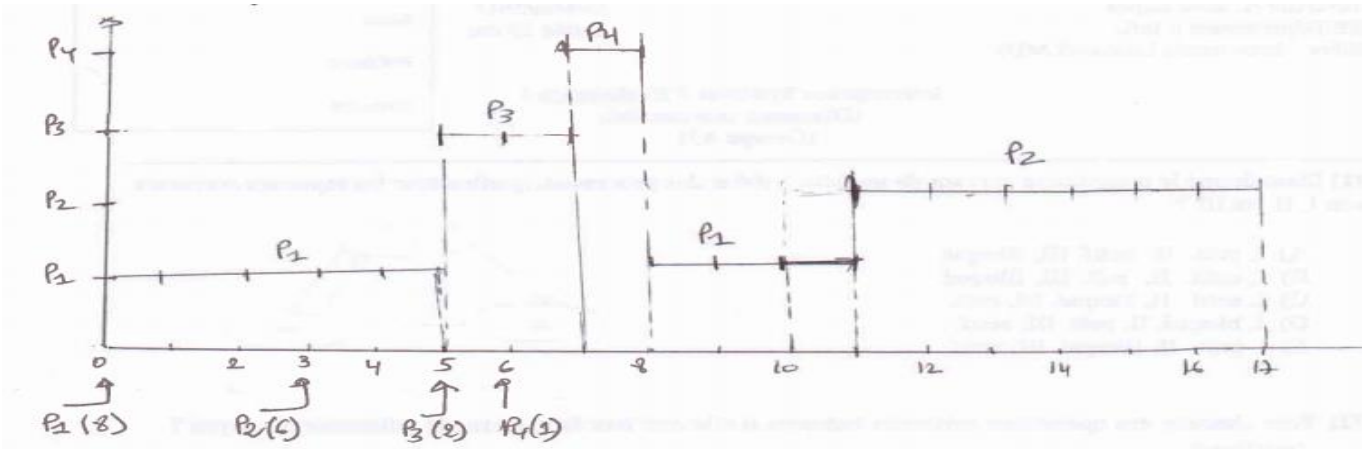


Le temps moyen d'attente est : (5 + 1 + 0)/3 = 2 unités.

Exercice: Soient quatre processus dont les temps d'arrivés et d'exécutions estimé sont données dans la table suivante :

processus	Temps d'arrivés	Temps d'exécution
P1	0	8
P2	3	6
P3	5	2
P4	6	1

1) Donner le diagramme de GANTT illustrant l'ordonnancement des processus en utilisant la méthode du plus court temps restant d'abord « SRTF» (short romained time first)

2) 

Intérêts :

SRT minimise le temps d'attente moyen des processus les plus courts Utilisation limitée à des environnements et à des applications spécifiques

Inconvénients :

- Pas de prise en compte de l'importance relative des processus
- Non équité de service : SRT pénalise les processus longs
- Possibilité de famine pour les processus longs

7 Bibliographie

[1] : Système d'exploitation. Chapitre : Allocation du processeur. Kévin Perrot. Aix-Marseille Université.2014. http://pageperso.lif.univ-mrs.fr/~kevin.perrot/teaching.html.

[1] SILBERSCHATZ, A. et P.B. GALVIN, Operating System Concepts. 8 th Edition, Addison Wesley. (http://profs.etsmtl.ca/agherbi/cours/log710h14/LOG710-Hiver2014-CPUScheduling.pdf).

Enseignant : Omar boukadoum / Email : boukadoum2020@gmail.com
Page Facebook : https://www.facebook.com/boukadoumomar/