Rappel

Multiprogrammation : son objectif est de disposer de quelques programmes en exécution à tout moment. Quand un processus en exécution demande une E/S, l'UC ne reste pas inactive et commute vers le processus suivant et ainsi de suite.

Temps partagé : son objectif est de commuter l'UC entre les processus si fréquemment que les utilisateurs peuvent interagir avec chaque programmes pendant qu'il est en cours d'exécution.

Dans un système *mono-processeur*, il n'y aura jamais plus d'un processus en exécution. S'il existe plus de processus, le reste devra attendre jusqu'à ce que l'UC soit libre et puisse être schedulée de nouveau.

1. Le scheduling

Quand deux processus rentrent dans le système, ils sont placés dans une file d'attente de travaux ou **pool des travaux**, se trouvant dans la mémoire secondaire (un disque par exemple). Elle contient tous les processus du système.
Les processus résidents en mémoire et qui sont prêts à s'exécuter sont maintenus dans une **file d'attente des processus prêts**.
Les processus en attente d'une E/S sont mis dans une file d'attente rattachée au périphérique correspondant : **file d'attente du périphérique**.
.
Un processus passe entre les diverses files d'attente de scheduling pendant toute sa durée de vie. Il doit être sélectionné à partir de ces files d'attente par le SE. Le processus de sélection s'appelle le ***scheduleur.***

2. Mécanisme de scheduling

Chaque fois que l'UC devient inactive, le système d'exploitation doit sélectionner l'un des processus dans la file d'attente des processus prêts à l'exécution. Le processus de sélection est mené à bien par le scheduleur à court terme (ou scheduleur de l'UC). Le scheduleur sélectionne entre les processus prêts (en mémoire centrale) et alloue l'UC à l'un d'eux.
La file d'attente des processus prêts n'est pas nécessairement une file d'attente FIFO. Il existe plusieurs techniques de sélection. Les éléments de la file d'attente sont les PCBs des processus. La file d'attente se trouve évidemment en mémoire centrale.

2.1. Scheduling avec réquisition

Les décisions de scheduling de l'UC peuvent avoir lieu sous les quatre circonstances suivantes :

1. Quand un processus commute de l'état en exécution vers l'état en attente (requête d'E/S ou création de fils) ;
2. Quand un processus commute de l'état en exécution vers l'état prêt (interruption) ;
3. Quand un processus commute de l'état en attente vers l'état prêt (fin d'une E/S) ;
4. Quand un processus se termine.

Pour les cas 1 et 4, il n'existe aucun choix en fonction du scheduling : un nouveau processus (s'il en existe dans la file d'attente des processus prêts) doit être sélectionné pour l'exécution.

Cependant, il existe un choix pour les cas 2 et 3.
Quand le scheduling a lieu dans les circonstances 1 et 4, nous parlons d'un *scheduling sans réquisition*. Dans le cas contraire, il est *avec réquisition.*

Avec le scheduling sans réquisition, une fois l'UC a été allouée à un processus, celui-ci garde l'UC jusqu'à ce qu'il la libère, soit parce qu'il a terminé, soit parce qu'il commute à l'état en attente. Cette méthode de scheduling est utilisée par Microsoft Windows.
Le scheduling avec réquisition demande du matériel spécial, par exemple une horloge mémoire. Celui-ci pose des problèmes de partage des ressources (discutés dans les problèmes de synchronisation).

2.2. Le dispatcheur (ou répartiteur)

C'est un autre composant impliqué dans la fonction de scheduling de l'UC. C'est le module qui donne le contrôle de l'UC au processus sélectionné par le scheduleur à court terme. Son rôle est donc d'exécuter les **commutations de contexte** nécessaires.

Le dispatcheur doit être le plus rapide possible, car il est appelé à chaque commutation de processus. On parle de latence de dispatching pour exprimer le temps que met le dispatcheur pour arrêter un processus et commencer l'exécution d'un autre.

3. Les critères de scheduling

Pour comparer les algorithmes de scheduling, on utilise les critères suivants :

1**. Utilisation de l'UC** : on désire maintenir l'UC aussi occupée que possible. Un système légèrement chargé suppose une utilisation de l'UC à près de 40%. Un système lourdement chargé l'occuperait à près de 90%.

2. **Capacité de traitement** : c'est la quantité de processus terminés par unité de temps : 10 processus/s pour les processus courts, 01 processus/h pour les processus très longs.

3. **Temps de restitution** : c'est l'intervalle de temps qui sépare le moment de soumission du travail au moment de sa terminaison. C'est aussi la somme des temps passés à attendre d'entrer dans la mémoire centrale, à rester dans la file d'attente des processus prêts, à s'exécuter sur l'UC et à effectuer des E/S.

4. **Temps d'attente** : l'algorithme de scheduling n'affecte pas la durée pendant laquelle un processus s'exécute ou effectue des E/S. Il affecte seulement le temps qu'un processus passe à attendre dans la file d'attente des processus prêts. Le temps d'attente est la somme des temps à attendre dans la file des processus prêts.

5. **Temps de réponse** : dans un système *temps partagé*, le temps de réponse est le temps écoulé à partir du moment ou on soumet une requête jusqu'à l'arrivée de la première réponse. C'est aussi le temps qu'il faut pour *commencer* à répondre, mais pas le temps qu'il faut pour sortir cette réponse, car la restitution finale dépend de la vitesse du dispositif de sortie.

4. Les algorithmes de scheduling

Le scheduling de l'UC permet de décider quel processus dans la file d'attente des processus prêts doit être alloué à l'UC. On utilise généralement un **diagramme de Gantt** pour représenter le scheduling.

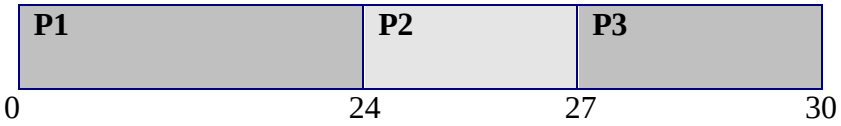
4.1. Algorithme du premier arrivé, premier servi (First come, first served ou FCFS)

On alloue l'UC au premier processus qui la demande. L'implémentation de la politique FCFS est une file d'attente FIFO.

Exemple : On considère l'ensemble suivant de processus arrivant au temps 0 avec les cycles suivants :

Processus	Temps du cycle (ms)
P1	24
P2	3
P3	3

Si les processus arrivent dans l'ordre p1, p2, p3, on obtient le diagramme de Gantt suivant :



Les temps d'attente et de restitution sont les suivants :

Processus	Temps d'attente	Temps de restitution
P1	**0**	24
P2	**24**	27
P3	27	30

Le temps d'attente moyen sera :
T_{ma}= (0+24+27)/3 = 17 ms

Si l'ordre d'arrivée était p2,p3,p1, on aurait eu un temps moyen
T_{ma}= (0+3+6)/3 = 3 ms seulement !

Ainsi, le temps moyen d'attente est *variable* dans la politique FCFS.

4.2. Algorithme du travail le plus court d'abord (Shortest Job First ou SJF)

On associe à chaque processus la longueur de son prochain cycle d'UC. Quand l'UC est libre, on l'assigne au processus qui possède le prochain cycle d'UC le plus petit. Si deux processus on la même longueur, le scheduling FCFS est utilisé pour trancher.

Exemple :

Processus	Temps du cycle (ms)
P1	6
P2	8
P3	7
P4	3

On aura le diagramme de Gantt suivant :

Le temps d'attente moyen est :

T_{ma}= (0+3+9+16)/4 = 7 ms

Avec un algorithme FCFS, on aurait eu un temps d'attente

T_{ma}= 10,25 ms.

L'algorithme SJF offre un temps d'attente minimal pour un ensemble de processus donné.

4.3. Algorithme avec priorité

L'algorithme SJF est un cas particulier d'une famille d'algorithme avec *des priorités.* On associe à chaque processus un priorité et l'UC est allouée au processus de plus haute priorité. Les processus ayant la même priorité sont schedulés dans un ordre FCFS.
Un algorithme SJF est simplement un algorithme avec des priorités ou la priorité (p) est l'inverse du prochain cycle d'UC (t) : p=1/t. Plus le cycle d'UC est grand, plus la priorité est petite et vice-versa.
En général, les priorités s'étalent sur un nombre de numéros (de 0 à 7 ou de 0 à 4095). Les priorités *hautes* ont des *numéros bas*, en général (comme c'est le cas dans le système UNIX). D'autres systèmes utilisent les numéros bas pour les priorités *basses*.

Exemple :

Processus	Temps du cycle (ms)	Priorité
P1	**10**	3
P2	**1**	1
P3	**2**	3
P4	**1**	4
P5	5	2

Le diagramme de Gantt est :

Le temps moyen d'attente est :

T_{ma}= 8,2 ms.

4.4. Algorithme du tourniquet (Round-Robin ou RR)

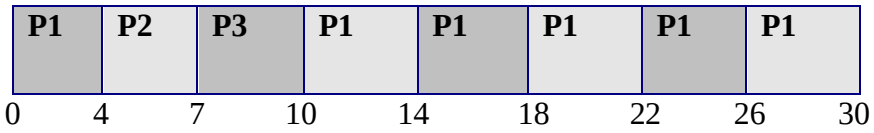
Il a été conçu pour les systèmes en *temps partagé*. Il ressemble à l'algorithme FCFS mais on peut réquisitionner pour commuter entre les processus. On définit une unité de temps, le *quantum* (entre 10 et 100 ms). La file d'attente des processus prêts est traitée comme un *file circulaire*. Le scheduleur parcourt la file d'attente en allouant l'UC à chaque processus pendant un intervalle de temps allant jusqu'à un quantum (ou tranche de temps).
La file d'attente est gérée comme une file FIFO. Le scheduleur de l'UC sort le premier processus de la file, configure une horloge pour qu'elle interrompe après 1 quantum et expédie le processus.

Exemple 1:

Processus	Temps du cycle (ms)
P1	24
P2	3
P3	3

Si le quantum = 4 ms, alors le premier processus obtient les 4 premières ms. Puisqu'il demande 20 ms supplémentaires, il est interrompu après la première tranche de temps et l'UC est allouée au processus p2. puisque p2 n'a pas besoin de 4 ms, il part avant l'expiration du quantum. L'UC est ensuite allouée à p3. une fois que chaque processus a reçu 1 tranche de temps, l'UC est renvoyée au processus p1.

On obtient le diagramme de Gantt suivant :



Les temps d'attente et de restitution sont résumés dans le tableau suivant :

Processus	Temps d'attente	Temps de restitution
P1	**(0-0)+(10-4)=6ms**	**30ms**
P2	**(4-0)=4ms**	**7ms**
P3	**(7-0)=7ms**	**10ms**

Le temps moyen d'attente est :

T_{ma} = 17/3=5,66 ms.

Ici le **temps d'arrivée est égal à 0**. Si cela n'était pas le cas, il faut le prévoir dans les *soustractions* dans le tableau précédent.

L'algorithme RR n'alloue pas l'UC à un processus plus d'une tranche de temps. Il effectue donc une réquisition lorsque le cycle d'UC dépasse le quantum (le processus revient dans la file d'attente des processus prêts).

Exemple 2:

Un quantum = 2 unités de temps.

Processus	Temps estimé	nbre de quanta	Temps d'attente	Temps de Résidence
1	20	10	10q : 20u	40
2	8	4	8q : 16u	24
3	2	1	2q : 4u	14
4	10	5	10q : 20u	30

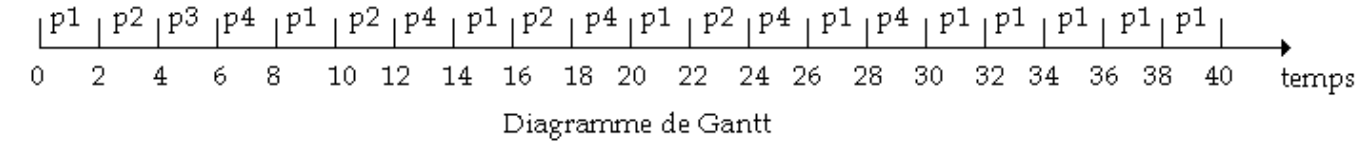


Diagramme de Gantt

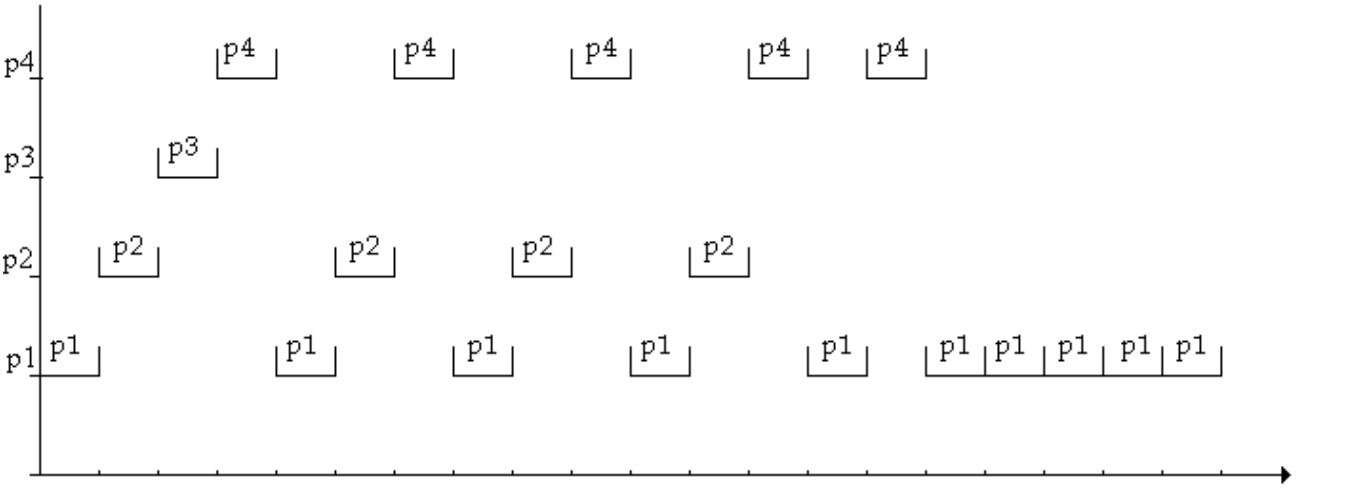


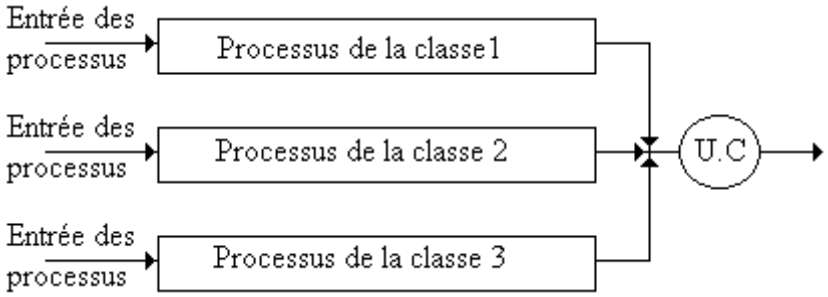
Diagramme de Gantt

4.5. Algorithme SRT (Shortest Remaining Time)

Une version *préemptive* de l'algorithme SJF s'appelle algorithme **SRT** (Shortest Remaining Time, temps restant le plus court) : chaque fois qu'un nouveau processus est introduit dans la file de processus à scheduler, le scheduleur compare la valeur estimée de temps de traitement restant à celle du processus en cours de scheduling. Si le temps de traitement du nouveau processus est inférieur, le processus en cours de scheduling est préempté. L'algorithme SRT favorise les processus courts.

4.6. Algorithme avec files multiniveaux

On définit des classes de processus et on associe à chacune des classes une priorité et une stratégie d'ordonnancement. Le processeur est alloué aux processus de la classe la plus prioritaire. On ne change de file que si la file la plus prioritaire est vide. Un processus ne peut pas changer de classe.



Exemple: Système Unix de SUN (Solaris). On dispose de 3 classes de processus:

- Processus en temps réel
- Processus système
- Processus en temps partagés.

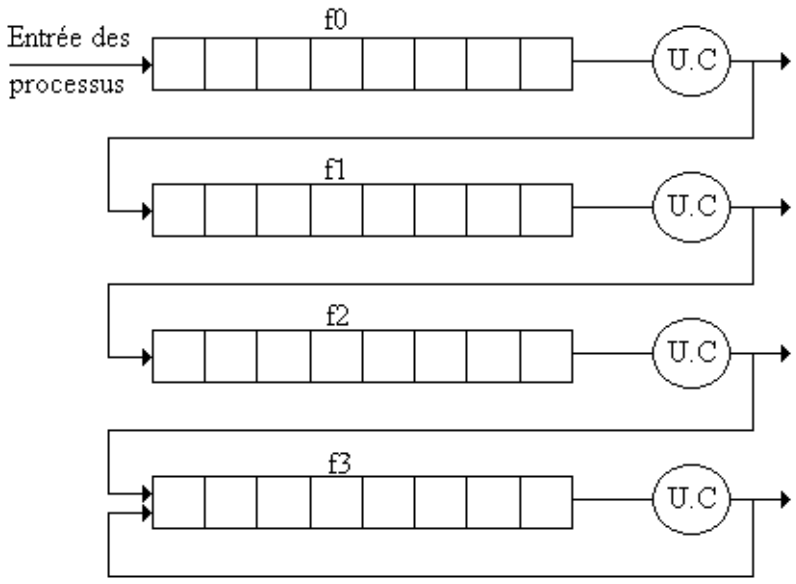
Les processus en temps réel sont plus prioritaires que les processus systèmes qui sont plus prioritaires que les processus en temps partagé.

4.7. Algorithme avec files multiniveaux avec recyclage (MFQ, Multilevel Feedback Queues)

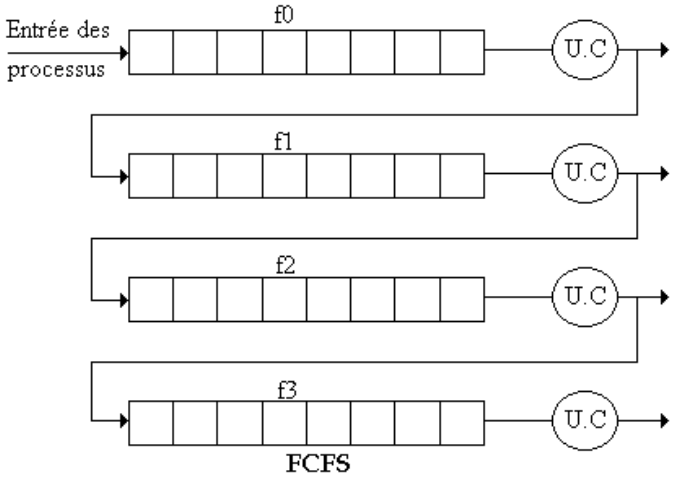
Dans la technique précédente les processus ne pouvaient pas changer de file. Un processus d'une classe A reste dans la file de cette classe jusqu'à ce qu'il quitte le système.

Dans cette stratégie, on dispose de n files de 'processus prêts': f0, f1, f2, ..., fn-1.

A chaque file fi est associé un quantum de temps qi dont la valeur croit avec le rang de la file. Un processus de la file fi n'est servi que si les files de rang inférieur à i sont vides. Lorsqu'un processus de la file fi a épuisé son quantum de temps sans avoir terminé son exécution, il rentre dans la file fi+1. Les processus de la dernière file (fn-1) sont recyclés dans la même file. Les nouveaux processus sont rangés dans la file f0. Cet algorithme favorise les processus courts.



Remarque: Une autre variante de l'algorithme applique FCFS à la dernière file (fn-1).



5. Conclusion

Dans les premiers temps de l'informatique, le scheduling (ou ordonnancement) était dans la plupart du temps **non préemptif et coopératif** : un processus conservait le contrôle de l'UC jusqu'à ce qu'il se bloque ou qu'il se termine. Cette approche correspondait parfaitement aux **systèmes de traitements par lots**. On y trouve les algorithmes FCFS, SJF et l'algorithme de priorité *sans réquisition* (non préemptif). .

Sur les **systèmes interactifs** actuels, c'est l'**ordonnancement préemptif** qui est utilisé. Le scheduleur peut préempter (interrompre) un processus avant qu'il se bloque ou se termine, afin d'attribuer l'UC à un autre processus. On y trouve les algorithmes SRT, RR, l'algorithme de priorité *avec réquisition* (ou préemptif) et MFQ.