Université Mohamed Kheider Biskra
Faculté des sciences exactes et sciences de la nature et de la vie
Département d'informatique

Support de cours

Module : Systèmes d'exploitation I

Chapitre 3 : Ordonnancement de processus

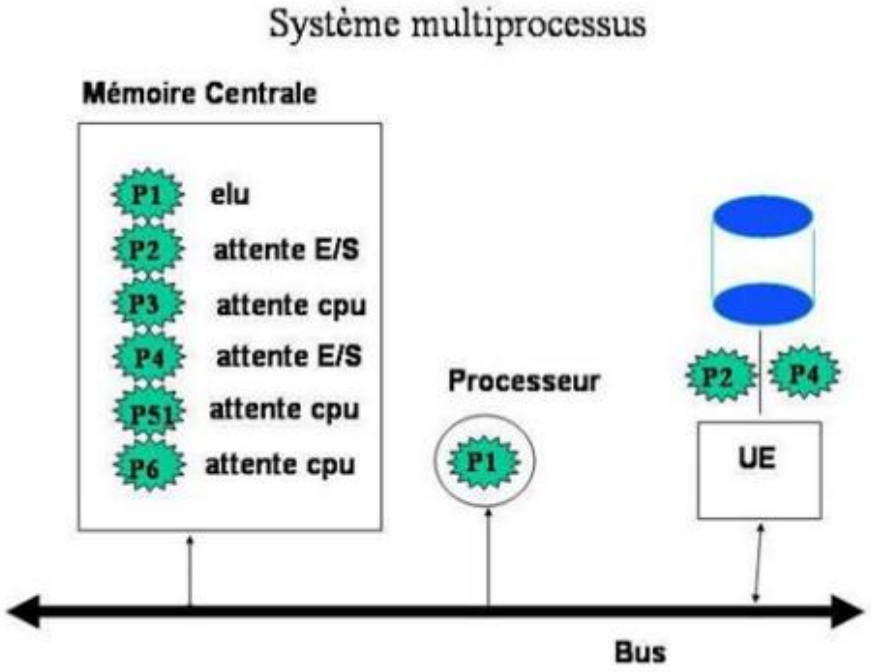
Niveau : 2LMD
Chargé de module : Mme. D. Boukhlouf

Année universitaire 2018/2019

Chapitre 3 : Ordonnancement de processus

1. Introduction

Dans un système multiprogrammé, on "démarre" 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.

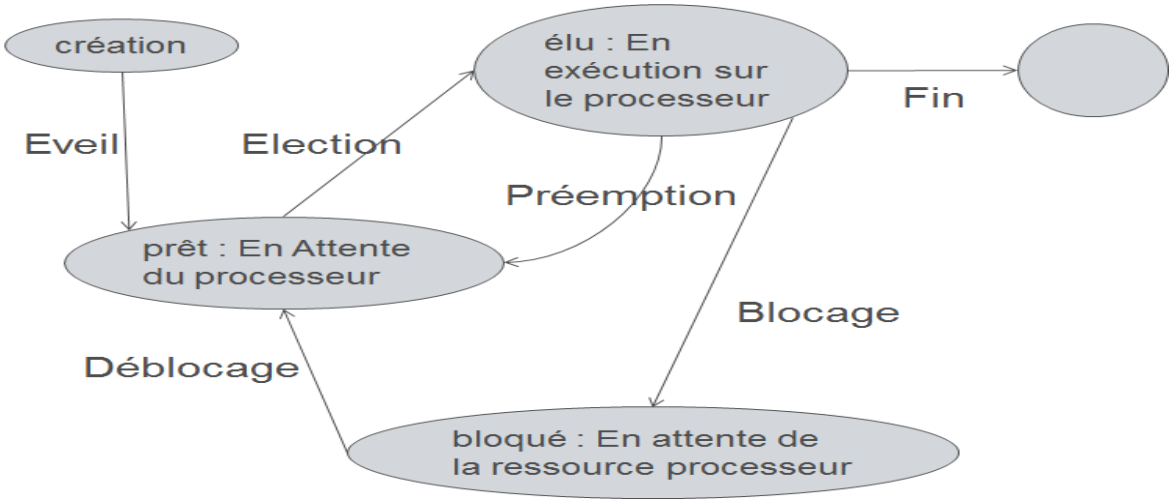


2. Définition de processus

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 (dynamique), 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.

2.1 Etats de 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 :



Création : 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.
Elu : le processus a été affecté à un processeur libre. Ses instructions sont en exécution.
Bloqué (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.
Fin : le processus a terminé son exécution.

2.2 Bloc de contrôle de processus :

Chaque processus est représenté dans le SE par un PCB (process control block).

PCB	
Pointeur	État du processus
Numéro du processus	
Compteur d'instructions	
Registres	
Limite de la mémoire	
Liste des fichiers ouverts	
. . .	

Il contient plusieurs informations concernant un processus spécifique, comme par exemple: L'état du processus. Compteur d'instructions: indique l'adresse de l'instruction suivante devant être exécutée par ce processus. Informations sur le scheduling de la CPU: information concernant la priorité du processus. Informations sur la gestion de la mémoire: valeurs des registres base et limite, des tables de pages ou des tables de segments. Informations sur l'état des E/S: liste des périphériques E/S allouées à ce processus, une liste des fichiers ouverts, etc.

3. 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.

4. Le scheduler (ordonnanceur):

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

5. 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.

6. 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 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 réponse.
- **Temps de rotation (turnaround)**: dit aussi temps de ***séjour***, est le temps pris par un processus de son arrivée à sa terminaison.

7. Algorithmes d'ordonnancement

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 regrouper en deux catégories : sans réquisition du CPU (stratégie non préemptive) ou avec réquisition du CPU (stratégie préemptive).

8. Modes d'ordonnancement

Deux classes d'ordonnanceur:
A. Non préemptif: (sans réquisition) Sélectionne un processus, puis le laisse s'exécuter jusqu'à ce qu'il bloque (soit sur une E/S, soit en attente d'un autre processus) où qu'il libère volontairement le processeur. Même s'il s'exécute pendant des heures, il ne sera pas suspendu de force. Aucune décision d'ordonnancement n'intervient pendant les interruptions de l'horloge.
B. Préemptif: (avec réquisition) Sélectionne un processus et le laisse s'exécuter pendant un délai déterminé. Si le processus est toujours en cours à l'issue de ce délai, il est suspendu et le scheduler sélectionne un autre processus à exécuter.

8. 1 Ordonnancement sans réquisition du CPU (sans préemption)

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 :

8.1.1 Premier Arrivé Premier Servi (FCFS : First come First Served) (appelée aussi FIFO)

L'algorithme de scheduling 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.

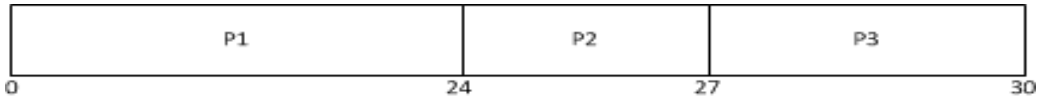
Exemple :

•On considère l'ensemble des processus suivants :

Processus	Cycle CPU
P1	24
P2	3
P3	3

On suppose que les processus arrivent à l'instant 0 et admis dans l'ordre : P1 , P2 , P3

•Le diagramme de Gantt pour l'ordonnancement FCFS de cet ensemble est :



•Temps d'attente pour P1 = 0; P2 = 24; P3 = 27

•Temps d'attente moyen : (0 + 24 + 27)/3 = 17

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

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.

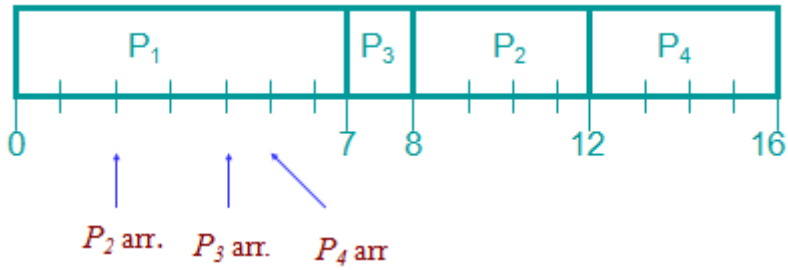
8.1.2 Plus court d'abords : Shortest Job First (SJF)

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 :

Processus	Arrivée	Cycle UCT
P_1	0	7
P_2	2	4
P_3	4	1
P_4	5	4

Le diagramme de Gantt :



Temps d'attente moyen = (0+(8-2)+(7-4)+(12-5))/4
(0 + 6 + 3 + 7)/4 = 4

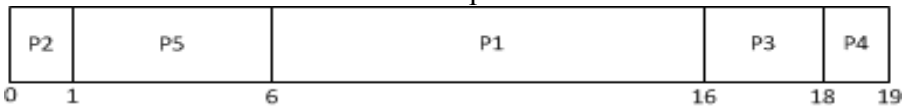
Critique de la méthode :
Il a été prouvé que l'algorithme SJF est optimal dans le temps dans le sens qu'il obtient le temps d'attente le plus court pour un ensemble de processus donné. Toutefois, cet algorithme est difficile à implémenter pour une raison simple : Comment peut-on connaître le temps d'exécution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

8.1.3 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	Cycle CPU	Priorité
P1	10	3
P2	1	1
P3	2	4
P4	1	5
P5	5	2

•Le diagramme de Gantt avec un ordonnancement avec priorité est comme suit :



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

Critique de la méthode :

Un algorithme d'ordonnancement avec priorité non préemptif dépose le nouveau processus en tête de la file des processus prêt :

•Un algorithme d'ordonnancement avec priorité pose un problème majeur

–Blocage infini ou famine (starvation)

–Peut laisser des processus à priorité basse en attente indéfinie

•Solution :

–Vieillissement (aging): Technique qui consiste à incrémenter graduellement la priorité des processus qui attendent dans le système depuis une longue période.

8. 2 Ordonnancement avec réquisition du CPU (avec préemption)

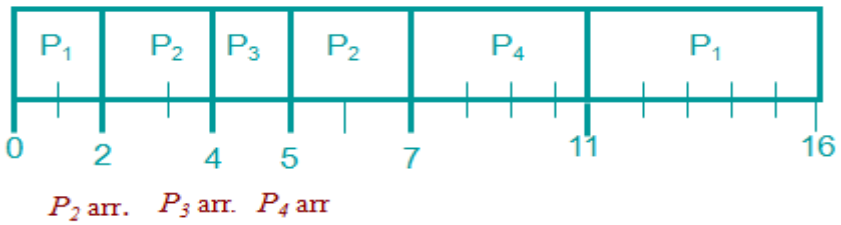
Dans cette catégorie, l'ordonnanceur peut retirer le CPU à un processus avant que ce dernier ne se bloque ou se termine afin de l'attribuer à un autre processus. A chaque \emph{quantum} (unité de temps) l'ordonnanceur choisi dans la liste des processus prêts un processus à qui le CPU sera alloué. Cette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

8.2.1 Le temps restant le plus court (SRTF : Shortest Remaining Time First)

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 de SJF avec préemption (SRTF)

Processus	Arrivée	Cycle
P_1	0	7
P_2	2	4
P_3	4	1
P_4	5	4

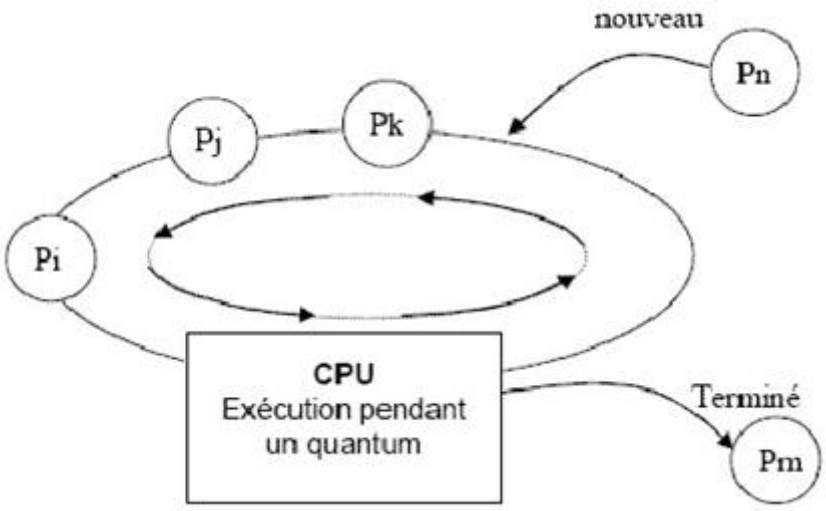


Temps moyen d`attente = (9 + 1 + 0 +2)/4 = 3

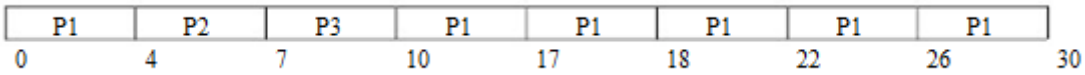
P1 attend de 2 à 11, P2 de 4 à 5, P4 de 5 à 7

8.2.2 L'algorithme de Round Robin (Tourniquet) :

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 (voir la figure ci-dessous). 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'exécutions, 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 :

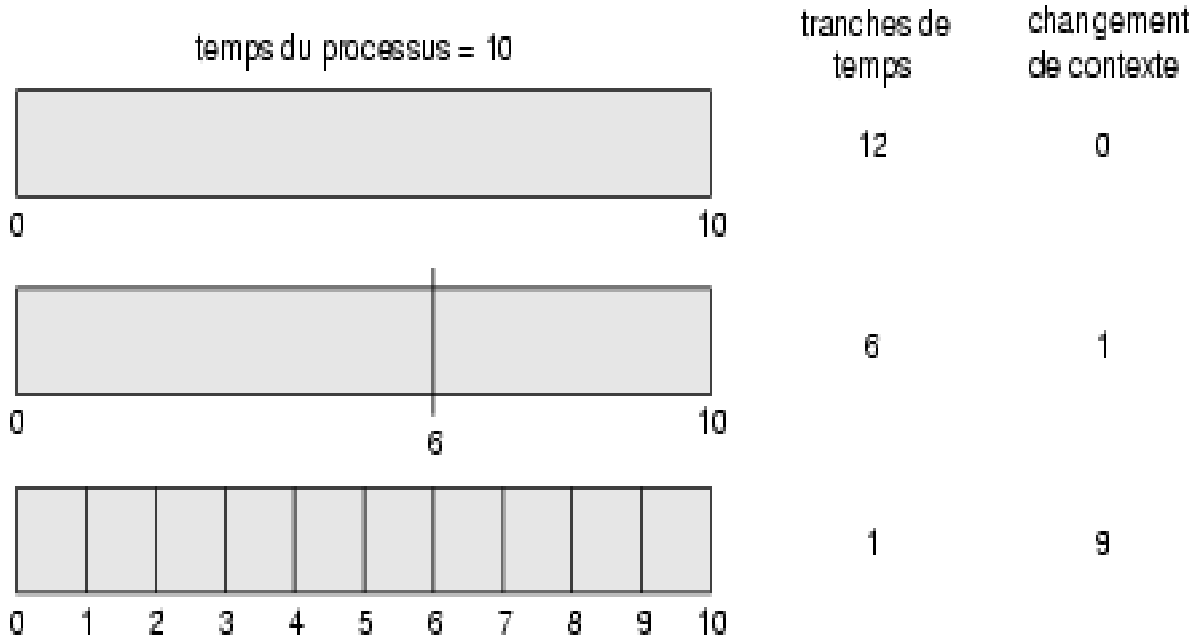


Le temps moyen d'attente est de : (6+4+7)/3 = 17/3 = 5.66 ms

Critique de la méthode

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

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.



8.2.3 Priorité avec préemption

Exemple :

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

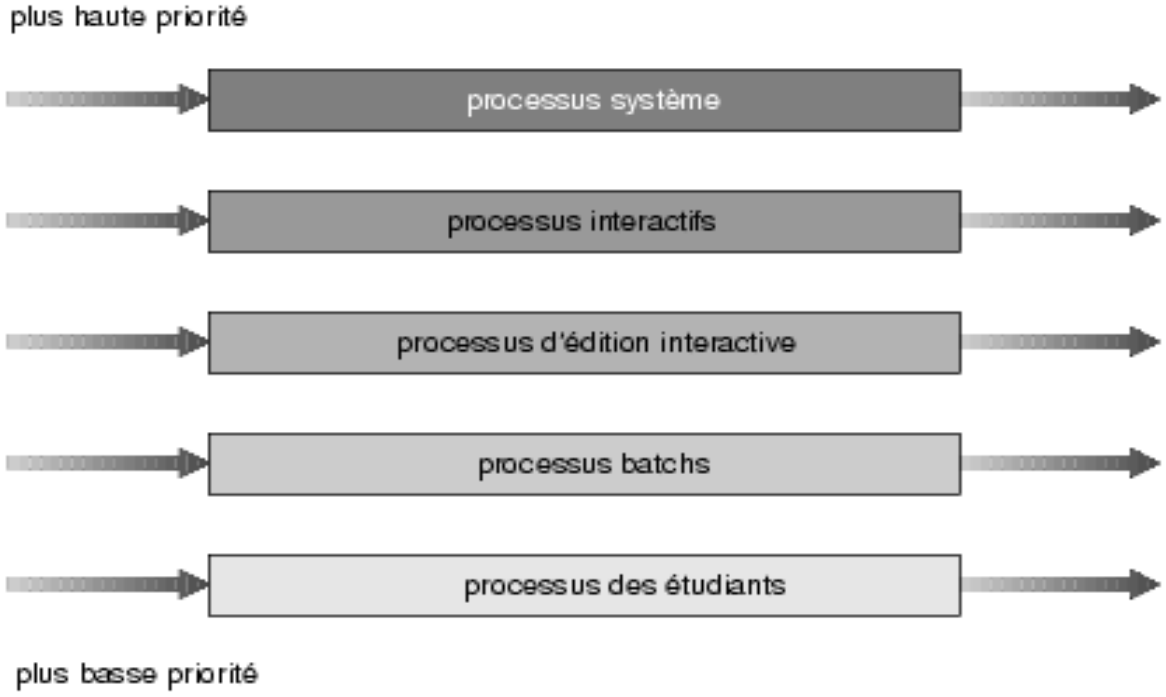
Avec réquisition:

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

9. 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 donne un exemple de découpage de ces files d'attente (ex.: serveur principal d'une université) :



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

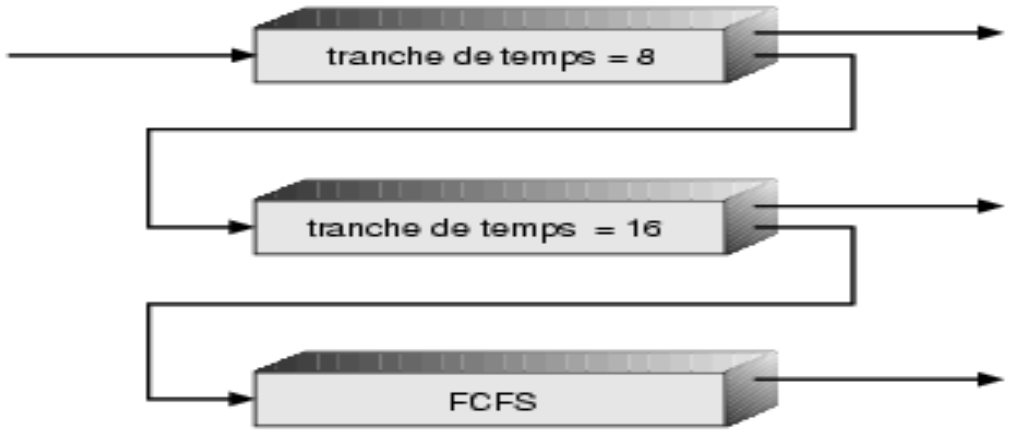
d'attente possède son propre algorithme de scheduling. Par exemple , la file d'attente des processus systèmes est schedulée selon algorithme de plus haute priorité, celles des processus interactifs est gérée selon l'algorithme Round Robin et la file des processus batch est gérée selon l'algorithme FCFS. 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.

10. 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 entête de la file 1. S'il ne termine pas, il est interrompu et il est mis dans la file d'attente 2. Le processus de la file d'attente 2 sont exécutées seulement quand les files 0 et 1 sont vides.

Références

SILBERSCHATZ, A. et P.B. GALVIN, Operating System Concepts. 8th Edition, Addison Wesley.2012.