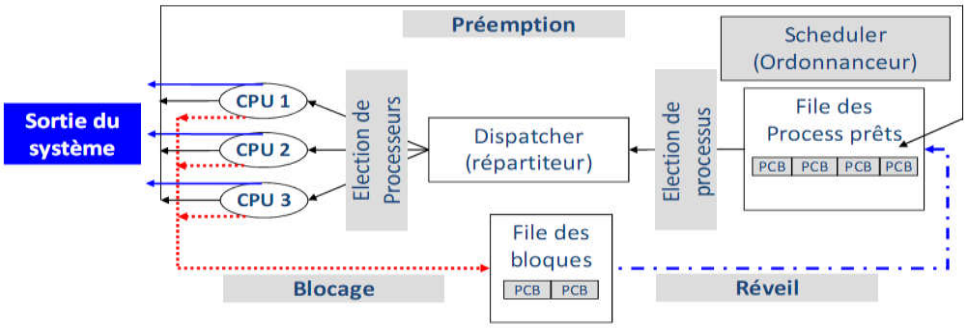
Chapitre III : Gestion du Processeur central

1. **Définition du scheduling/Scheduler** : Le scheduler (ou dispatcher ou ordonnanceur) est un module (programme) du système d'exploitation qui attribue le contrôle de l'unité centrale, à tour de rôles, aux différents processus en compétition suivant une politique définie à l'avance par les concepteurs du système.
   La figure suivante montre le schéma général d'un scheduler :



2. **Objectifs de scheduling :** La politique de scheduling vise essentiellement à optimiser les performances du système à savoir : l'équité, le rendement et l'utilisation des ressources
   - **L'équité** : Le scheduler doit servir les programmes de même priorité d'une manière juste et équitable.
   - **Le rendement** : Désigne le taux d'occupation de l'unité centrale (i.e. Le nombre de travaux réalisés par unité de temps). De même, il minimise le temps de réponse des travaux.
   - **L'utilisation 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 processus.
3. **Critères de scheduling** : Il existe plusieurs critères à prendre en compte lors de l'ordonnancement :
   - **La disponibilité des ressources** : Afin de planifier les exécutions des programmes selon les ressources demandées et celles qui existent.
   - **La classe des programmes** : C'est un élément déterminant pour définir la stratégie de scheduling (d'ordonnancement) :
     - Programmes orientés calcul ou E/S.
     - Programmes interactifs
     - Programmes temps réel…
   - **Scheduling avec ou sans préemption :** Une politique de scheduling est non préemptive : une fois le processeur central est alloué à un processus ; il le gardera jusqu'à sa terminaison.
   - **Scheduling avec ou sans priorité :** L'allocation du processeur central aux processus peut se faire selon certaines valeurs de priorité qui sont affectés aux processus (programmes) automatiquement par le système ou de manière externe par les usagers :

- **Priorité statique** : Une fois allouée à un processus, elle restera inchangée jusqu'à sa terminaison.
- **Priorité dynamique :** Elle change en fonction de l'environnement d'exécution du processus (ex : ancienneté, …).

4. **Politiques (Algorithmes) de scheduling** :

A. Politiques de schudeling sans préemption :

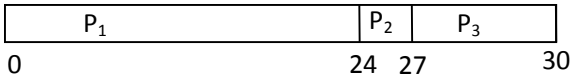
On définit les concepts suivants :

- ***Temps d'attente*** : C'est le temps passé à attendre dans la file d'attente des processus prêts.
- ***Temps de réponse*** ($T=T_{Fin} - T_{Arrivé}$) est le temps de réponse d'un processus, avec T_{Fin} est le temps de fin d'exécution du processus et $T_{Arrivé}$ est son temps d'arrivé.

i. Premier arrive, premier servi (FIFO *(First In First Out)* ou FCFS *(First Come First Served)*) :

Dans ce cas, le processus qui sollicite le CPU en premier, sera servi le premier. Quand le processus (en cours d'exécution) termine, le processeur est alloué au prochain processus. L'implémentation de cette politique est réalisée par une file de processus. Le PCB d'un processus, nouvellement arrivé dans le système, est inséré dans la queue (fin) de la file.

Exemple *:* Trois processus P1, P2 et P3 arrivent, à instants 0, 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. En effet, Le temps d'attente est égal à 0 pour le processus P, 24 pour le processus P2 et 27 pour le processus P3. Par conséquent, 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.

Critique de la méthode :

La politique FIFO désavantage les processus courts s'ils ne sont pas exécutés en premier.

ii. L'algorithme du Plus Court d'abord (SJF (*Shortest Job First*)) : Cet algorithme affecte le processeur au processus possédant le temps d'exécution le plus court.

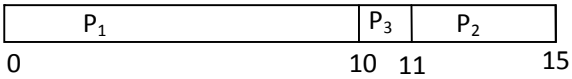
Si plusieurs processus ont la même durée, une politique FIFO sera alors utilisée pour les départager.

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

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

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

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.

Critique de la méthode :

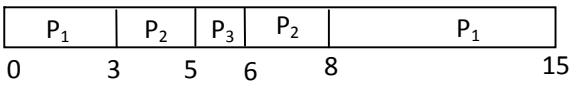
Cette politique avantage les processus les plus courts. Toutefois, cet algorithme est difficile à implémenter pour une raison simple : Comment peut-on connaître le temps d'exécution d'un processus à l'avance ?.

B. Politiques de schudeling avec préemption :

i. Le plus court temps restant le premier (**SRTF** (Shortest Remained Time First)): Le problème qui se pose pour l'algorithme SJF est quand un nouveau processus arrive en queue de la file et qu'est un temps d'exécution plus petit que le temps d'exécution ***restant*** du processus en exécution. Afin de préserver le plus court processus d'abord, on arrête le processus en cours afin d'entamer le nouveau processus.

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.

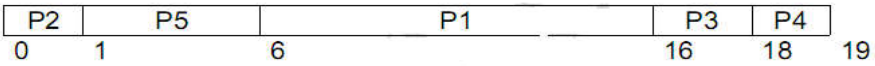
ii. Scheduling avec priorité : Cet algorithme associe à chaque processus une priorité (un entier), et *le processeur sera affecté au processus de plus haute priorité*. Cette priorité varie selon les systèmes et peut aller de 0 à 127.

NB : *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	Temps 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.

Solution : 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.

iii. **L'algorithme de Round Robin (Tourniquet)** : Cet algorithme 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.

Cet Ordonnancement est régit par les règles suivantes :

① Un processus qui rentre dans l'état éligible (prêt) est mis en queue de la file d'attente des processus prêts.

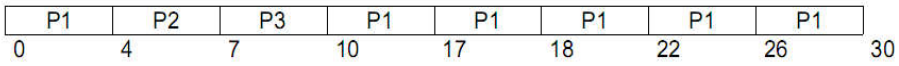
② Si un processus élu se termine ou se bloque (pour des raisons de synchronisation par exemple) avant de consommer son quantum de temps, le processeur est immédiatement alloué au prochain processus se trouvant en tête de la file d'attente des prêts.

③ Si le processus élu continue de s'exécuter au bout de son quantum, dans ce cas le processus sera interrompu et mis en queue de la file d'attente des prêts et le processeur est réquisitionné pour être réalloué au prochain processus en tête de cette même file d'attente.

NB : 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 :



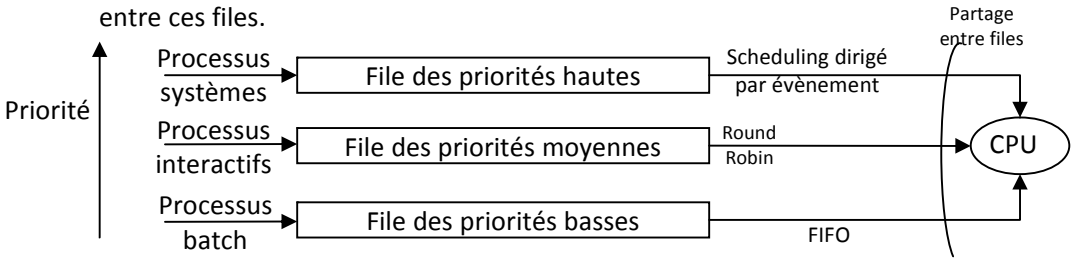
Le temps moyen d'attente est de : (6+4+7)/3 = 17/3 = 5.66 ms

Remarque : 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, il y'aura une surcharge due au changement des contextes.

Exercice : 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.

iv. **Scheduling avec files d'attente multi-niveaux** : Cette stratégie s'adapte à des situations où l'on peut classifier les processus en groupes selon des critères bien déterminés.

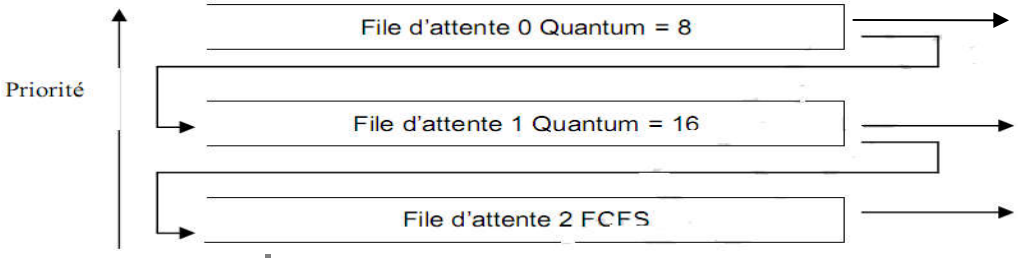
Dans ce cas, les processus prêts sont répartis à travers plusieurs files selon les types de processus. Chaque file est alors gérée par son propre algorithme de scheduling. En plus, on doit disposer une politique de scheduling entre ces files. Elle peut être réalisée par priorité ou par partage de temps de l'unité centrale entre ces files.



Exemple de scheduling multi-niveaux.

v. **Scheduling avec files d'attente multi-niveaux et feedback** : Dans la stratégie précédente, un processus ne change pas de file tout au long de son exécution. Maintenant, un processus peut basculer entre files. Ceci est réalisé dans le but d'isoler les processus consommateurs de temps d'unité centrale d'une part, et de relancer les processus de faible priorité d'autre part.

Exemple : Un système est doté de 3 files d'attentes multi-niveaux : 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 file 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.

Remarque : *Cette technique permet de favoriser les petits travaux sans avoir besoin de savoir à l'avance combien de temps CPU ceux-ci vont utiliser.*