Cours 05: Ordonnancement des processus

Plan du cours

Introduction

Mécanisme et Critères de l'ordonnancement

Algorithmes d'ordonnancement

Exercice: Implémentation en C++

Conclusion

Section 1 : Mécanisme de l'ordonnancement (*scheduling*)

Introduction

- Qualité de Service QoS (**ordonnancement**, sécurité, performance, fiabilité, etc.)

guichets de services

En système d'exploitation

- En informatique, **l'ordonnancement** est la méthode par laquelle les ***threads***, les ***processus*** ou ***les flux de données*** ont accès aux **ressources du système** (par exemple, le temps du processeur (cycle CPU), la bande passante des communications).

- Cela se fait habituellement pour atteindre une qualité de service (Qos) cible.

Rappel

- Dans un système uniprocessus, un seul processus est exécuté

  à la fois, d'autres processus doivent attendre le temps de CPU ☹

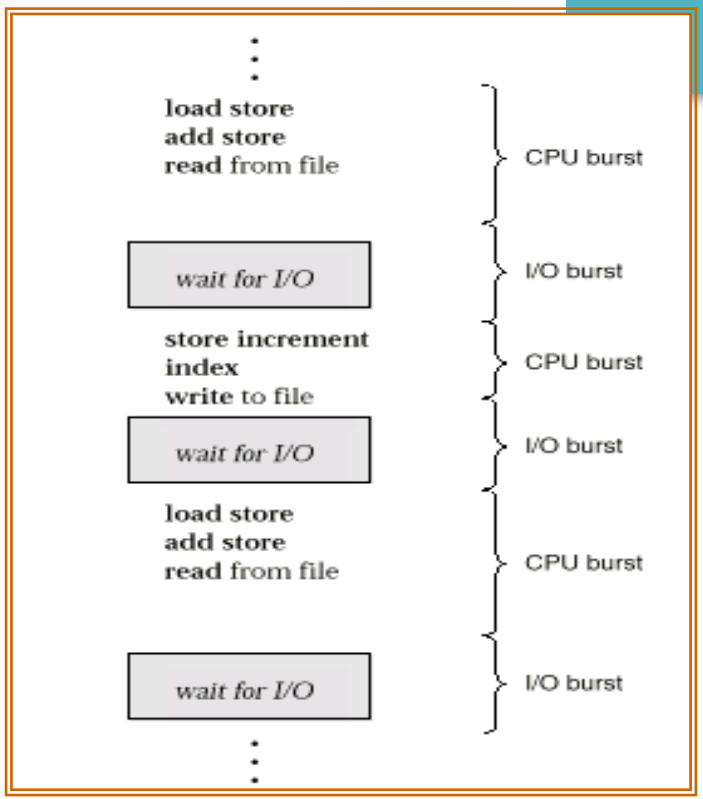
- L'exécution d'un processus consiste en des cycles d'utilisation CPU

  et d'attente sur E/S (I/O).

- Comment continuer à exécuter des processus tout le temps ?

  - pour une utilisation maximale de l'UC

- Solution : Multi programmation ☺
  - La présence simultanée, en mémoire principale, de plusieurs programmes.
  - Affectation de processeur



➲ La multiprogrammation nécessite le mécanisme d'**ordonnancement**

CPU Scheduling

- Le système d'exploitation gère une collection de processus.
- Si un système a plus de processus et un seul CPU
  - il faut diviser le temps CPU entre différents processus.

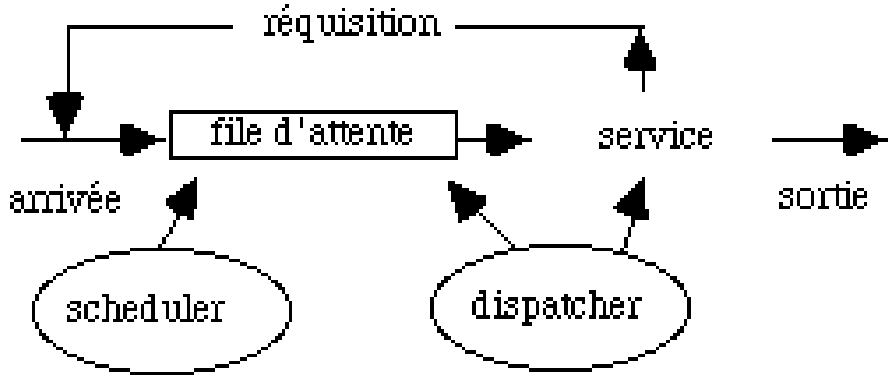
➲C'est ce qu'on appelle CPU Scheduling

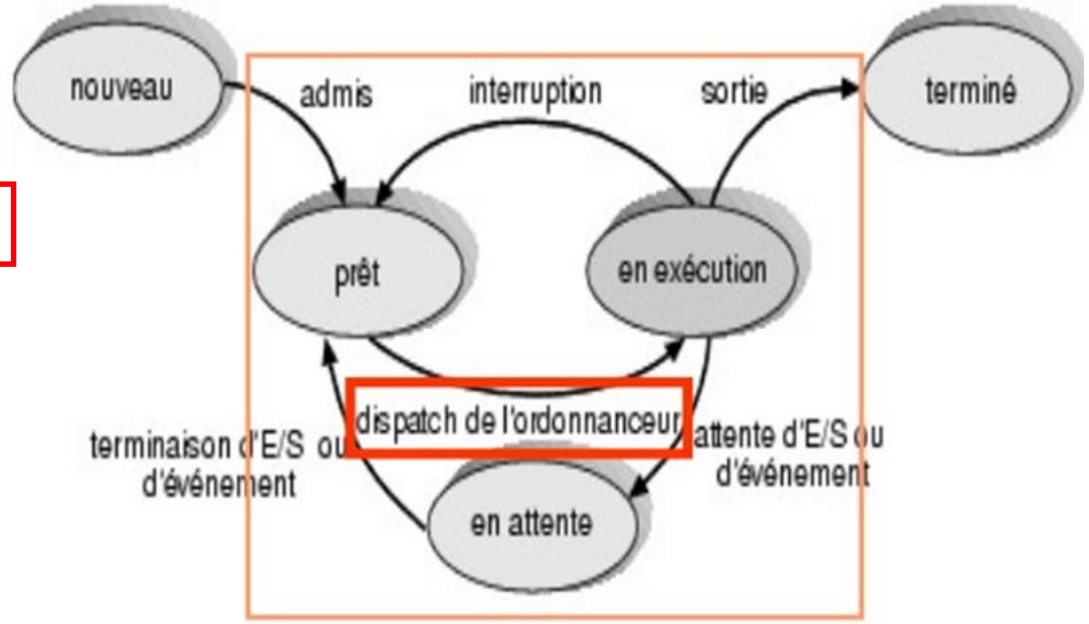
Le *scheduling* de l'UC permet de décider quel processus dans la file d'attente des processus prêts doit être alloué à l'UC.

Gestion du processeur par le scheduler et le dispatcher

- Le scheduler gère la file d'attente. Il analyse les demandes qui arrivent et les place dans la file.
- Le dispatcher suit l'activité du processeur, il choisit la requête à traiter et l'extrait de la file. Il gère donc l'allocation du processeur
- Il peut le réquisitionner en fonction de la priorité des processus demandeurs et du temps déjà consommé par celui qui est en cours d'exécution.

le scheduler et le dispatcher





Type d'Ordonnancement: non préemptif /préemptif

- L' ordonnancement non préemptif:
  - un processus conservait le contrôle de l'UC jusqu'à ce qu'il se bloque ou qu'il se termine.
  - correspond parfaitement aux systèmes de traitements par lots.
- L' ordonnancement préemptif
  - L'ordonnanceur (dispatcher de ordonnancement) peut préempter (**interrompre**) un processus avant qu'il se bloque ou se termine, afin d'attribuer l'UC à un autre processus.

Les critères de scheduling

Utilisation CPU –Maintenir le CPU le plus utilisé possible.

Débit –(*throughput*) C'est le nombre de processus qui terminent leur exécution par unité de temps (1000 par heure, ou 10 par seconde?).

Temps de retournement (Restitution) –(*turnaround time*) Temps nécessaire pour exécuter un processus (depuis la soumission jusqu'`a la terminaison).

Temps d'attente –(waiting time) Temps qu'un processus passe dans la file ready.

Temps de réponse –(response time) Temps entre la soumission d'une requête et la première réponse (et non la sortie finale).

- On cherche à maximiser **l'utilisation du CPU** et **le débit**, et à minimiser les temps de **retournement**, **attente** et **réponse**.
- On peut chercher à optimiser la moyenne ou bien le minimum/maximum

Section 1: Mécanisme de scheduling

Question

- Si n processus doivent être ordonnancés sur une unité centrale, combien d'ordonnancements différents peut-on avoir ?
  - Donner une formule en fonction de **n**.

$$n*(n-1)*(n-2)*..*2*1=n!$$

- Comment le scheduler sélectionne un processus à exécuter?

Section 2 : Scheduling Algorithms

Scheduling Algorithms

- Premier arrivé, premier servi (**First Come, First Served)**
- Algorithme du travail le plus court d'abord (**Shortest Job First)**
- Algorithme avec priorité
- Algorithme du tourniquet (**Round Robin)**
- Algorithme temps restant le plus court (SRT : **Shortest Remaining Time**)
- Algorithme avec files multiniveaux
- Algorithme avec files multiniveaux avec recyclage (**MFQ, Multilevel Feedback Queues**)

Les algorithmes de scheduling

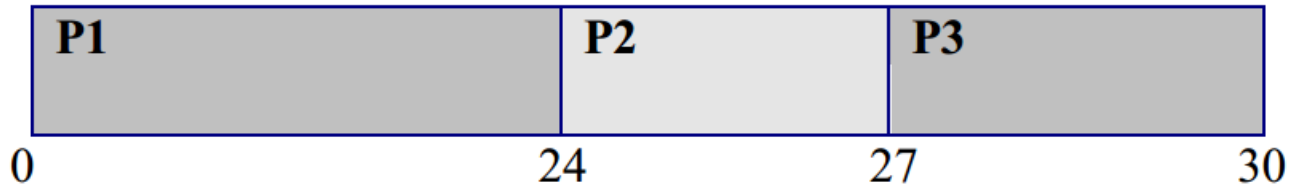
On utilise généralement un **diagramme de Gantt** pour représenter le *scheduling*.

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 :

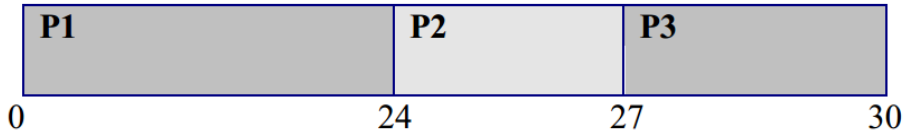
Processus	Temps du cycle (ms)
P1	24
P2	3
P3	3

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



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

- 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

Temps de retournement

- Le temps d'attente moyen sera : **Tma= (0+24+27)/3 = 17 ms**
- Si l'ordre d'arrivée était **p2,p3,p1**, on aurait eu un temps moyen
- **Tma= (0+3+6)/3 = 3 ms** seulement **!**
- Ainsi, le temps moyen d'attente est variable dans la politique **FCFS**.

Débat 2mn

- Quels sont les *avantages* et les *inconvénients* de l'algorithme FCFS?

FCFS

-Facile à comprendre

-Facile à programmer des files d'attente simples

-Le cas de bloquer ou d'ajouter de nouveau processus nécessite simplement de l'attacher à la queue de la file d'attente

- Le temps d'attente moyen est souvent assez long

-Petit processus attend un grand processus

-Ne convient pas au partage du temps

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

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

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 : **Tma= (0+3+9+16)/4 = 7 ms**

Avec un algorithme FCFS, ⇨ temps d'attente: **Tma= 10,25 ms.**

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

Débat 2mn

- Quels sont les *avantages* et les *inconvénients* de l'algorithme FCFS?

SJF

- On favorise les processus courts ce qui augmente le débit de processus traités et donc la réactivité
- …

- Risques de famine (un processus trop long pourrait ne jamais être exécuté s'il y a toujours un plus court)

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.

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.

Algorithme avec priorité

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 : **Tma= 8,2 ms**.

Débat 2mn

- Quels sont les *avantages* et les *inconvénients* de l'algorithme FCFS?

Algorithme avec priorité

- Simplicité
- le temps augmente -> augmenter la priorité d'un processus
- Adapté aux applications avec des temps et des ressources variables

-Si le système se bloque finalement, tous les processus de faible priorité se perdent

-Blocage indéfini ou famine...

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.

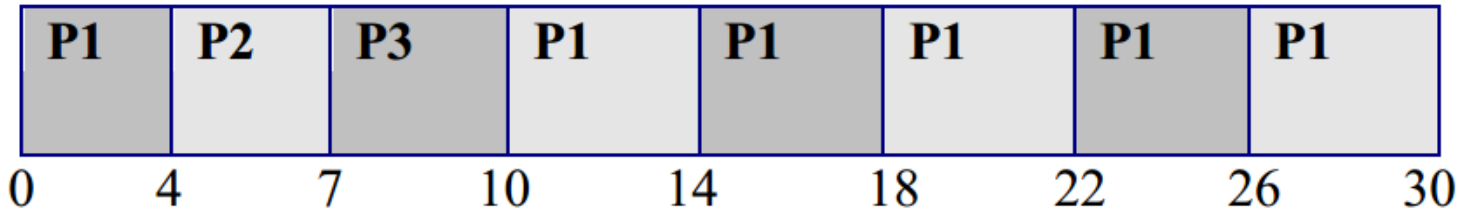
Algorithme du tourniquet (Round-Robin ou RR)

Exemple:

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

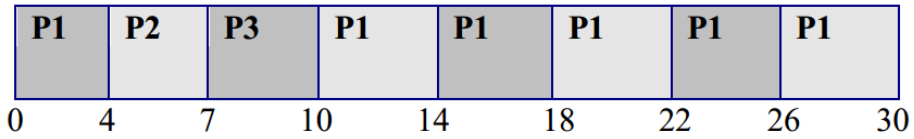
Si le quantum = 4 ms, alors le premier processus obtient les 4 premières ms. Puisqu'il demande 20 ms supplémentaires, il est interrompu après la première tranche de temps et l'UC est allouée au processus p2. puisque p2 n'a pas besoin de 4 ms, il part avant l'expiration du quantum. L'UC est ensuite allouée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



Algorithme du tourniquet (Round-Robin ou RR)

- **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 : **Tma = 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).

Algorithme du tourniquet (Round-Robin ou RR)

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

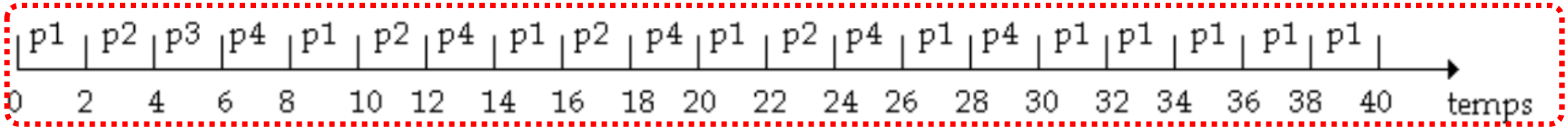
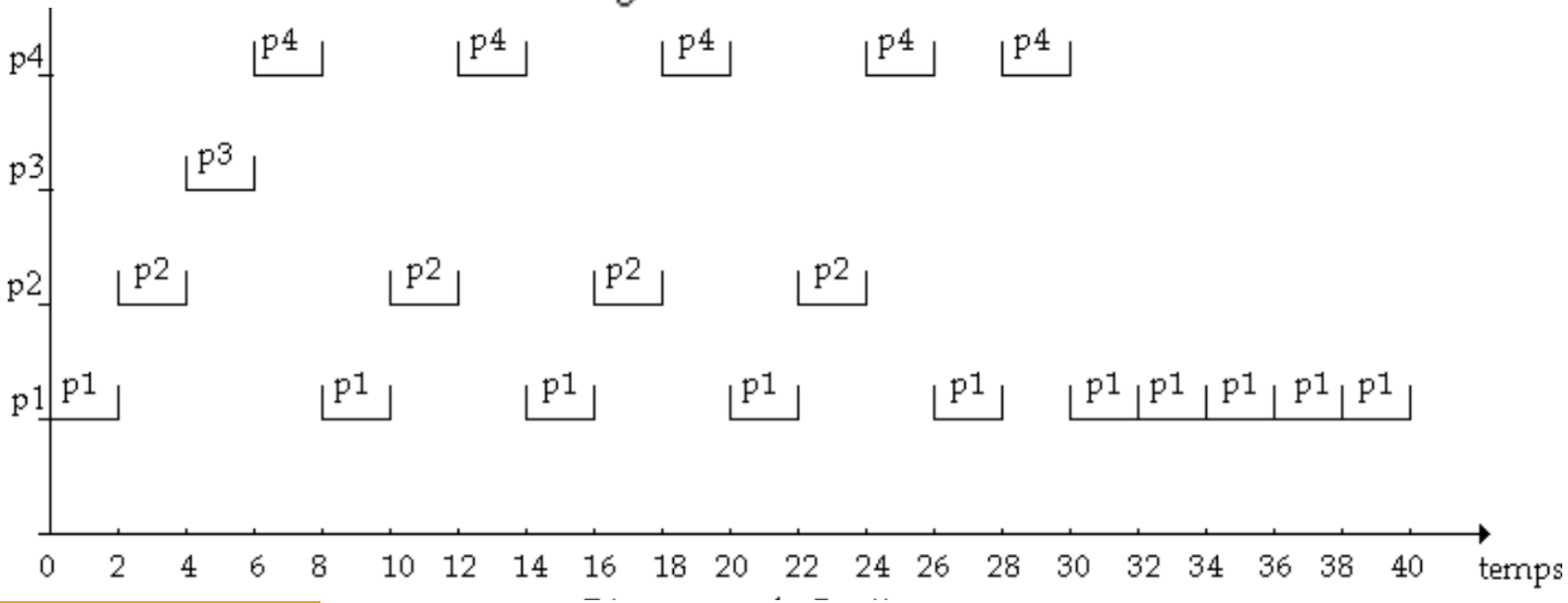


Diagramme de Gantt



Débat 2mn

- Quels sont les avantages et inconvénients du choix d'un quantum petit pour l'algorithme de scheduling Round Robin ?
- Quels sont les *avantages* et les *inconvénients* de l'algorithme FCFS?

Round-Robin ou RR

- Simple et facile à implémenter
- Chaque processus a la même chance d'exécution
- Gestion de tous les processus sans priorité
- Sans famine

- Dépend de la longueur de la tranche de temps
- même FCFS, si la tranche de temps est infiniment grande
- la fréquente commutation de contexte

Exercice

Envisager l'ensemble suivant de processus, avec la durée du cycle d'UC donnée en millisecondes :

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

Tous les processus sont arrivés à l'instant 0 dans l'ordre du tableau.

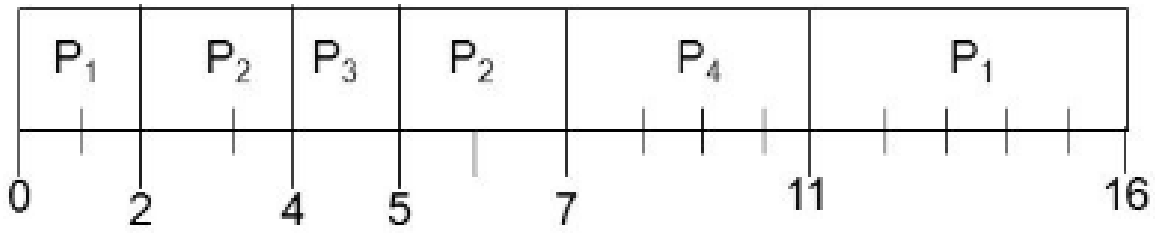
1. dessiner les diagrammes de Gantt illustrant l'exécution de ces processus selon les algorithmes FCFS, SJF, priorité (avec et sans préemption) et RR (quantum=1ms).
2. quel est le temps de restitution de chaque processus pour chaque algorithme ?
3. quel est le temps d'attente de chaque processus pour chaque algorithme ?
4. lequel des schedulings obtient-il le temps moyen d'attente minimal sur tous les processus ?

Algorithme SRT: temps restant le plus court (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.

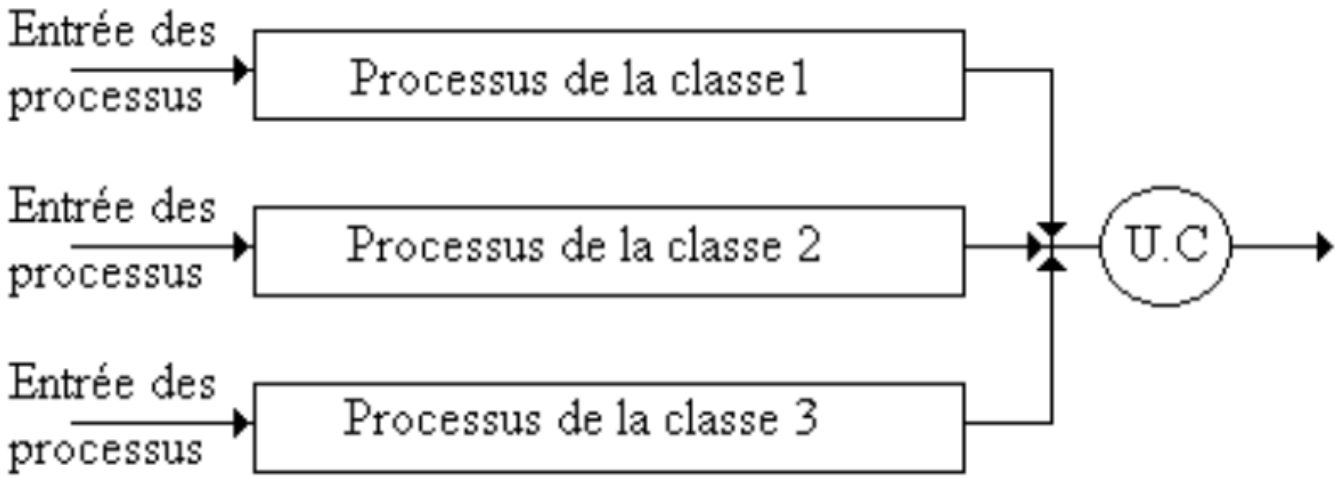
Process	Arrival Time	Burst Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4



- **Waiting time** for $P_1 = 11\text{-}2 = 9$; $P_2 = 5\text{-}4 = 1$; $P_3 = 4\text{-}4 = 0$; $P_4 = 7\text{-}5 = 2$

Algorithme avec files multiniveaux

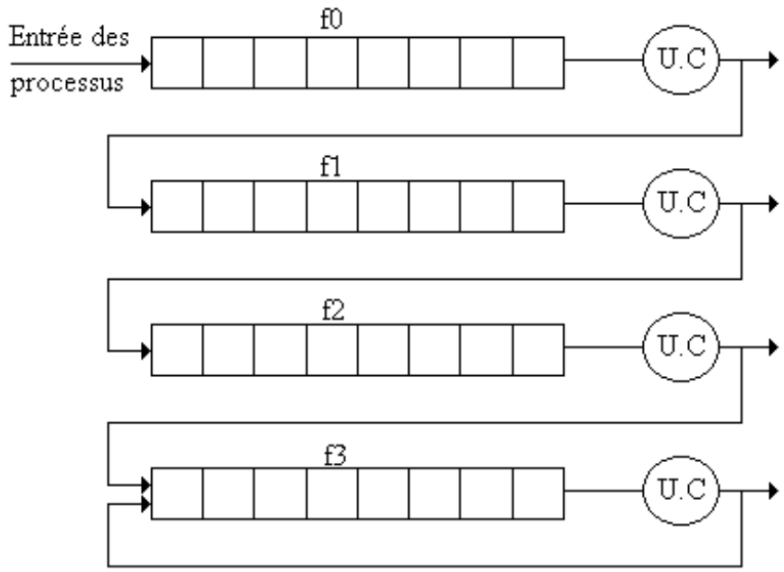
On définit des classes de processus et on associe à chacune des classes une priorité et une stratégie d'ordonnancement. Le processeur est alloué aux processus de la classe la plus prioritaire. On ne change de file que si la file la plus prioritaire est vide. Un processus ne peut pas changer de classe.



- Ce type de scheduling est intéressant dans le cas où les processus ne sont pas tous identiques et ont des besoins en ressources différents (par exemple : les processus système, les processus utilisateurs, ...etc).

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

- dans un scheduling multiniveau et feedback, un processus peut changer de file.
- Il est intéressant d'utiliser le feedback dans le cas où on doit suivre le comportement des processus et décider si on doit changer le niveau d'un processus soit à la baisse , soit à la hausse, en fonction de l'utilisation des ressources. Par exemple, il est recommandé de "dégrader" un processus qui utilise trop longtemps le processeur pour éviter qu'il pénalise le reste des processus



Section 2: Contexte et états des processus

Quelles sont les informations nécessaires pour implémenter un processus ?

- Une mémoire virtuelle
- Un contexte d'exécution

Section 3 : Implémentation des processus

Implementation in C++

- **Class:** cpuschedule
- **Attributes:**
  - n – number of processes
  - Bu[ ] – Array to store Burst Time
  - A[ ] – Array to store Arrival Time
  - Wt[ ] – Array to store Waiting Time
  - Twt – Total Waiting Time
  - Awt – Average Waiting Time

- **Operations:**
  - Getdata() – To get number of processes and Burst Times from the user
  - Fcfs() – First Come, First Served Algorithm
  - Sjf() – Shortest Job First Algorithm
  - Priority() – Priority Algorithm
  - RoundRobin() – Round Robin Algorithm

Section 4 : Conclusion

Conclusion

Les éléments mentionnés ci-dessus sont les différentes *CPU scheduling* utilisées dans le présent.

Ces algorithmes sont hérités en fonction de l'exigence du processeur.

Cours 05: Ordonnancement des processus