Université Ibn Tofail
جامعة ابن طفيل
ⵜⴰⵙⴷⴰⵡⵉⵜ ⵉⴱⵏ ⵟⴼⵓⵢⵍ
Ibn Tofail University

Système d'Exploitation -Ordonnancement-

Med. AMNAI

Filière SMI-S4

Département d'Informatique

2023-2024

Plan

Plan

Plan

Plan

Plan

Plan

Plan

Plan

Plan

Plan

Plan

Plan

Plan

Plan

Contexte

- **Objectif** : Gérer l'allocation du CPU aux processus
- **Problématique** : Quelle politique d'allocation du CPU au processus à choisir pour utiliser le processeur "au mieux" ?
- **Solution** : L'**Ordonnanceur** (**scheduler**) choisit (l'ordre) des processus qui vont pouvoir accéder au CPU.

Ordonnanceur

- L'**ordonnanceur** est un algorithme qui va choisir le processus qui a accès au CPU. Ce processus a le « privilège » d'accéder au CPU pendant un intervalle de **temps**.

- L'Ordonnancement se fait à deux niveaux :

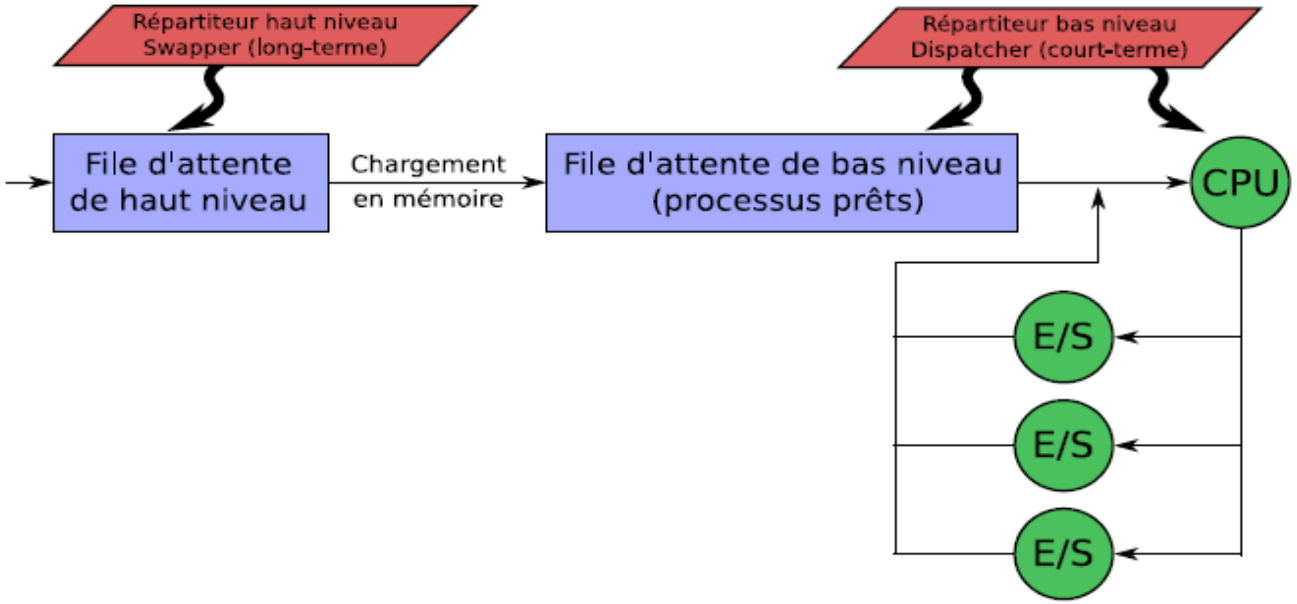
  - Une répartition de **haut niveau** qui sélectionne le prochain processus à charger en mémoire (mémoire virtuelle) ;
  - Une répartition de **bas niveau** (**dispatcher**) qui sélectionne, à chaque fois que le CPU devient inactif, un processus parmi tous ceux qui sont prêts.

File d'Attente

- Idée : gestion des accès par file d'attente.
- Une file d'attente pour chaque périphérique (ressource) et par type d'E/S (lecture, écriture, ...).
- Questions :
  - Ordre d'insertion des processus ?
  - Mode de passage entre file et CPU ?
  - Gestion E/S et CPU identiques ?

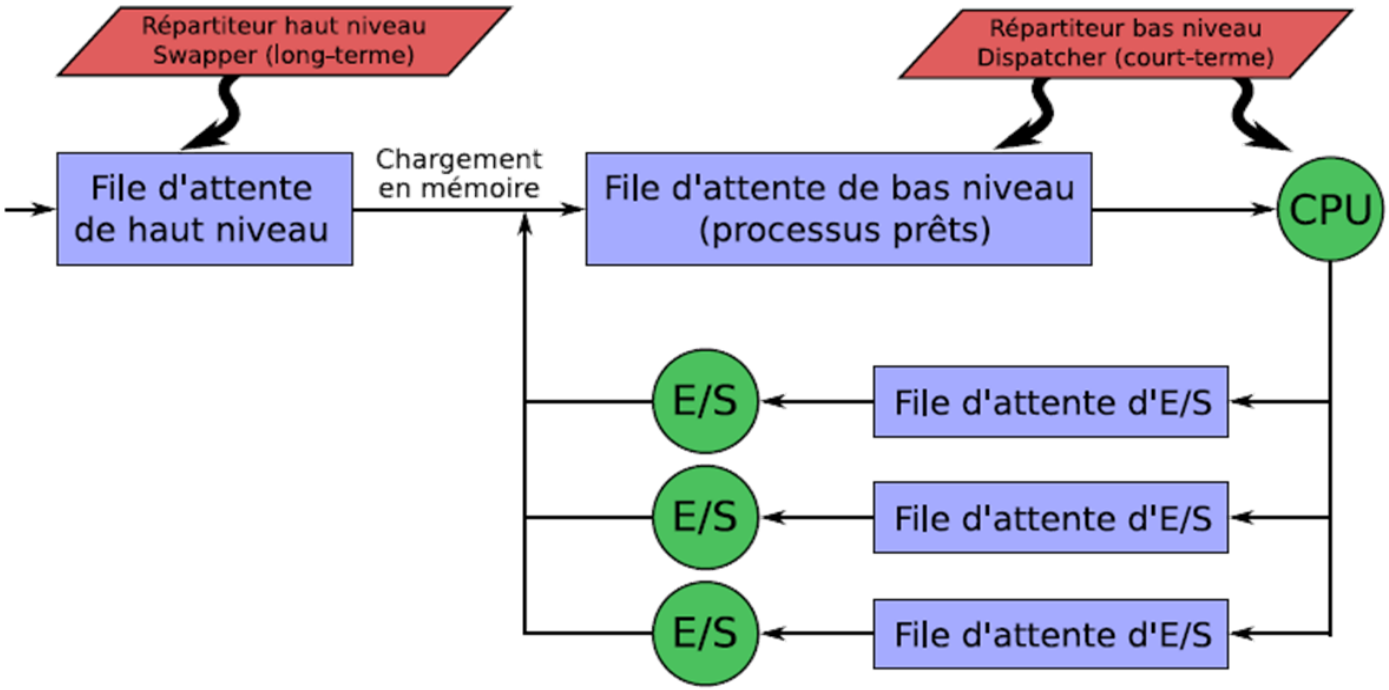
Traitement par lots (Batch mode)

- Pas d'ordonnancement de bas niveau ;
- En cas de demande d'E/S, le **CPU reste inactif** durant le traitement.



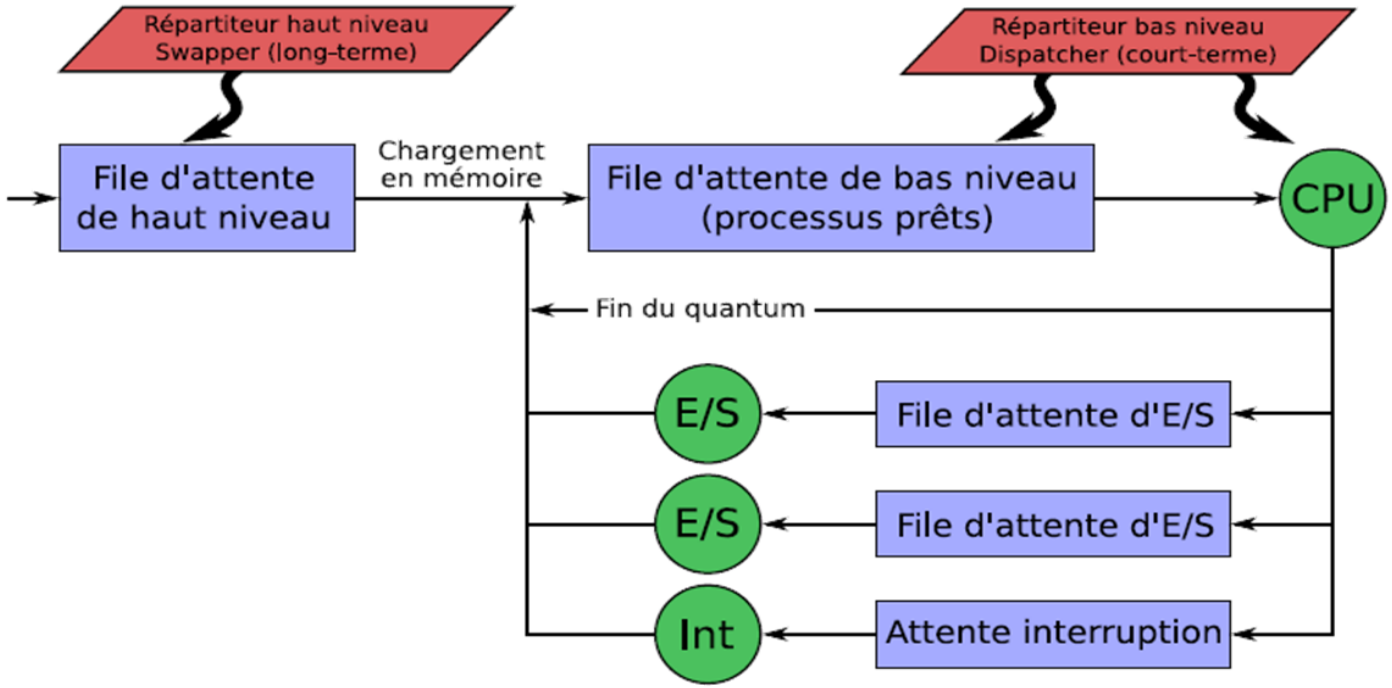
Multiprogrammation

- En cas de demande d'E/S, le **CPU est libéré** pour traiter un autre processus.



Temps Partagé

- Quantum de temps : durée de **temps maximale** allouée à chaque processus.



Objectifs d'Ordonnanceur

- S'assurer que chaque processus en attente d'exécution obtienne sa part de temps processeur (**Equité**).
- Minimiser le temps de réponse.
- Utiliser le processeur à 100% (**Efficacité**).
- **Utilisation équilibrée** des ressources.
- Prendre en compte des **priorités**.
- Être **prédictibles**.

Critères d'Evaluation des Performances

- **Rendement** : Nombre de travaux exécutés par unité de temps.
- **Temps de service (TE) ou d'exécution (séjour)** :
  - Temps qui s'écoule entre le moment où un travail est soumis et où il est exécuté (temps d'accès en mémoire + temps d'attente en file des éligibles + temps d'exécution dans le processeur et le temps de E/S).
- **Temps d'attente (TA)** : Temps passé dans la file des processus éligibles.
- **Temps de réponse (TR)** : Temps qui s'écoule entre la soumission d'une requête et la première réponse obtenue

Classes d'Ordonnanceurs

- **Non préemptif (sans réquisition)** : un processus **ne relâche le CPU** que quand il a fini ou quand il passe en E/S.

- **Préemptif (avec réquisition)** : le **CPU** n'est **utilisé** que pendant un **quantum** de temps par un processus.

Ordonnanceurs Non Préemptifs

Dans un système à **ordonnancement non préemptif** ou sans **réquisition**, un processus est **exécuté jusqu'à la fin sans suspension** ou il se bloque (en attente d'un événement) :

- Le premier arrivé est le premier servi (**PAPS**) (First Come First Served (**FCFS**) ou (**FIFO**)) ;
- Le plus court d'abord (short job first **SJF**).

Ordonnanceurs (FCFS)

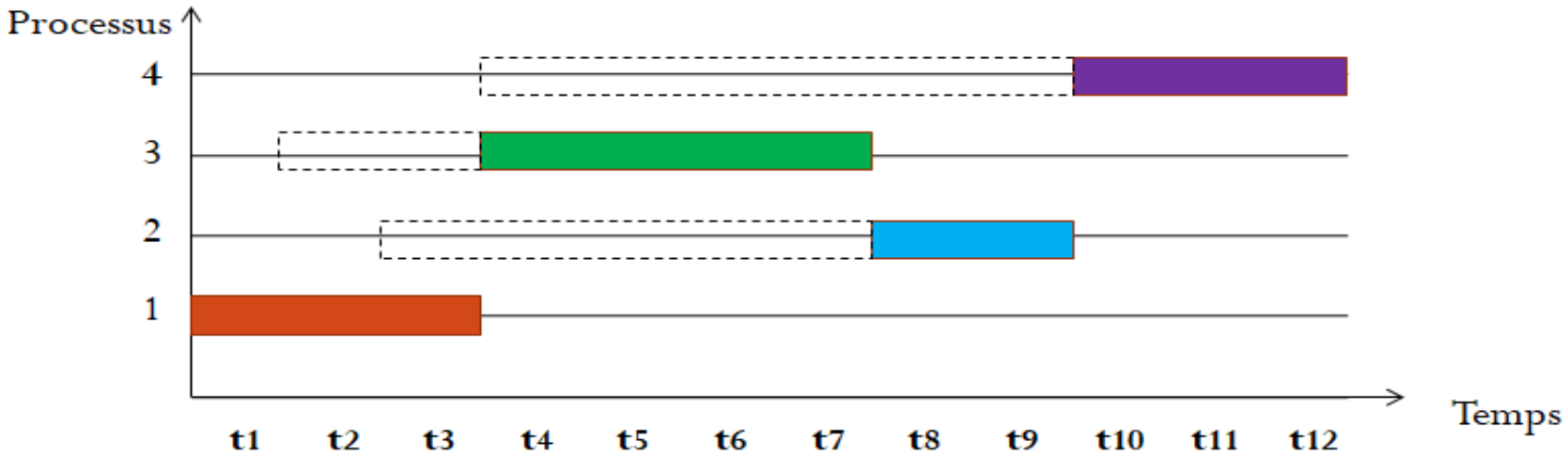
Les processus sont **ordonnés** suivant leur **date d'arrivée** au système.

- **Propriétés** :
  - Non préemptif (pas de réquisition) ;
  - Pas de notion de priorité sur les processus ;
  - Pas de connaissance sur la durée des processus ;
  - Croissance rapide du temps d'attente.
- **Avantage** : Algorithme facile à mettre en oeuvre ;
- **Inconvénient** : Un **processus** avec un temps d'exécution (**TE**) **minimum risque d'attendre longtemps** avant d'être exécuté.

Exemple (FCFS)

Processus	Durée	Temps d'arrivage
1	3	0
2	2	2
3	4	1
4	3	3

Processus	Temps de d'attente	Temps séjour
1	0	3
2	5	7
3	2	6
4	6	9



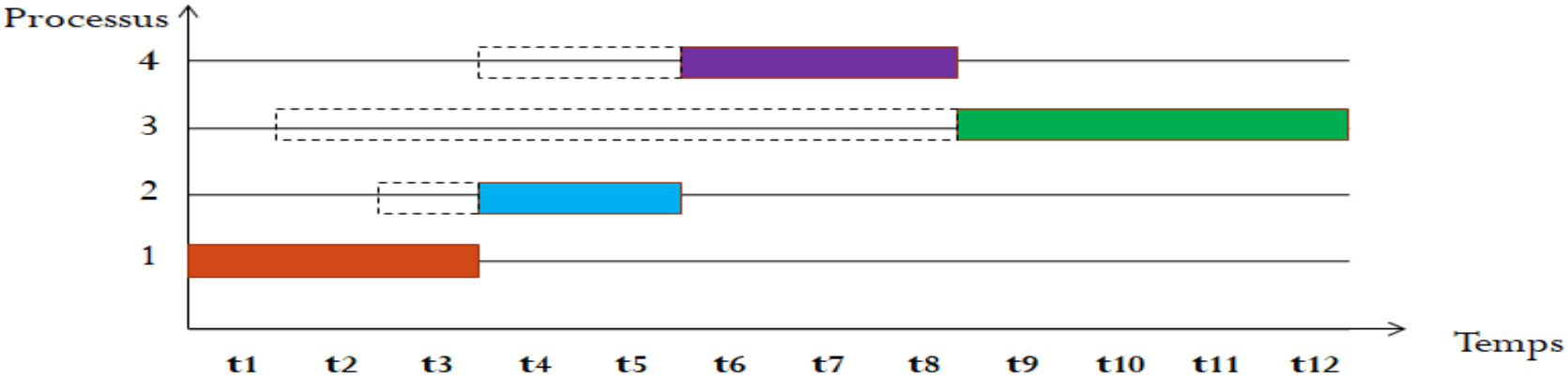
Ordonnanceurs (SJF)

- La file d'attente est ordonnée selon le **TE**. Le processus le **plus court** est exécuté en premier.
- **Proriétés** :
  - Non préemptif ;
  - Les processus ayant un minimum d'unités CPU à consommer sont mis en tête de la liste ;
  - Nécessité de connaître la durée des processus ;
  - En cas d'**égalité** utiliser **FCFS**.
- **Inconvénients** :
  - Les processus ayant un **TE très grand risquent de ne jamais être exécutés**.
  - Comment connaître le temps CPU des processus ?

Exemple (SJF)

Processus	Durée	Temps d'arrivage
1	3	0
2	2	2
3	4	1
4	3	3

Processus	Temps de d'attente	Temps séjour
1	0	3
2	1	3
3	7	11
4	2	5



Exemple (TME)

- Avec l'ordre 1 :
  - Le temps écoulé après l'exécution de **A** est de **8min**, de **12min** pour **B**, de **16min** pour **C** et de **20min** pour D. Le temps moyen d'exécution **(TME) = 14 min**.
- Avec l'ordre 2 :
  - Les temps écoulés sont **4**, **8**, **12** et **20min** et le **(TME) = 11min**. Il est plus optimal.

1)

8 min	4 min	4 min	4 min
A	B	C	D

Tête Queue

2)

4 min	4 min	4 min	8 min
B	C	D	A

Tête Queue

Ordonnanceurs Préemptifs

- Un ordonnanceur préemptif est appelé aussi **avec réquisition** ;
- Afin de partager le temps processeur, les ordinateurs ont une **horloge** qui génère périodiquement une **interruption** ;
- Le SE reprend la main à **chaque interruption** et décide si le processus en cours d'exécution **a consommé son quantum** (unité de temps) et **alloue le processeur (CPU) à un autre processus**.

Shortest Remaining Time (SRT)

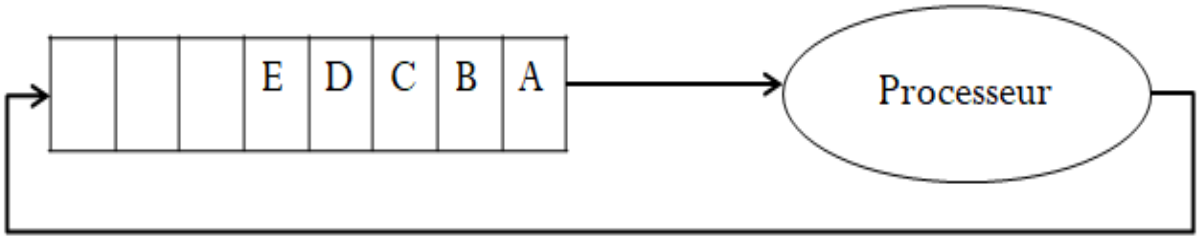
Le plus court temps restant est le prmier :

- **SRT** est la version préemptive de l'algorithme **SJF** ;
- Un processus arrive dans la file de processus,
  - l'ordonnanceur **compare la valeur espérée** pour ce processus contre **la valeur estimée du temps restant du processus actuellement en exécution**.
  - **Si le temps** du nouveau processus est **plus petit**, il rentre **en exécution** immédiatement.

Round Robin (RR)(tourniquet, circulaire)

- Dans le **(RR)**, un **quantum est assigné à chaque processus**. Si l'exécution du processus n'est pas terminée à la fin du quantum, il est **replacé en fin de file**.
- **Propriétés** :
  - Pas de notion de priorité sur les processus Préemptif.
  - Pas de connaissance d'avance sur la durée des processus.
- **Un processus libère le CPU si** :
  - Demande d'E/S.
  - Terminaison.
  - Fin du quantum de temps alloué

Round Robin (RR)(suite)



- Un **quantum trop petit** provoque **trop de commutations** de processus et abaisse l'efficacité du processeur.
- Un **quantum trop élevé augmente le temps de réponse des courtes** commandes en mode interactif.
- Un quantum entre **20** et **50** *ms* est souvent un compromis **raisonnable**.

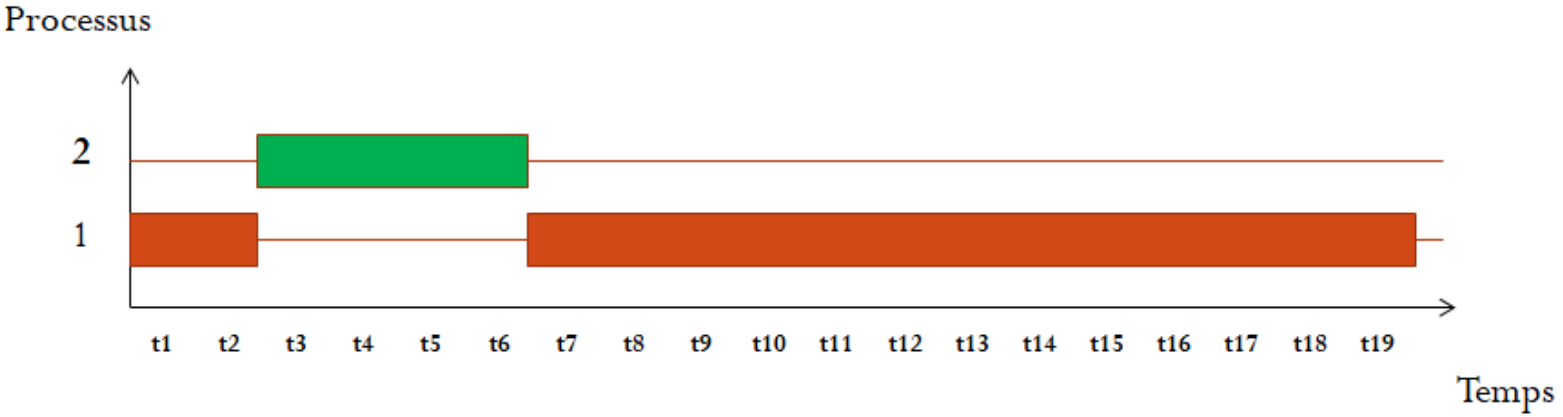
Exemple

- Soient deux processus **A** et **B** prêts tels que **A** est arrivé en premier, suivi de **B** à **2** unités de temps après. Les temps de UCT nécessaires pour l'exécution des processus **A** et **B** sont respectivement **15** et **4** unités de temps. Le temps de commutation est supposé **nul**.
- Calculer le **temps de séjour** de chaque processus **A** et **B**, le temps **moyen de séjour**, le temps d'**attente**, le temps **moyen d'attente**, et le **nombre de changements de contexte** pour :
  1. SRT.
  2. Round robin (quantum = 10 unités de temps).
  3. Round robin (quantum = 3 unités de temps).

Exemple : Solution SRT

Processus	Durée	Temps d'arrivage
1	15	0
2	4	2

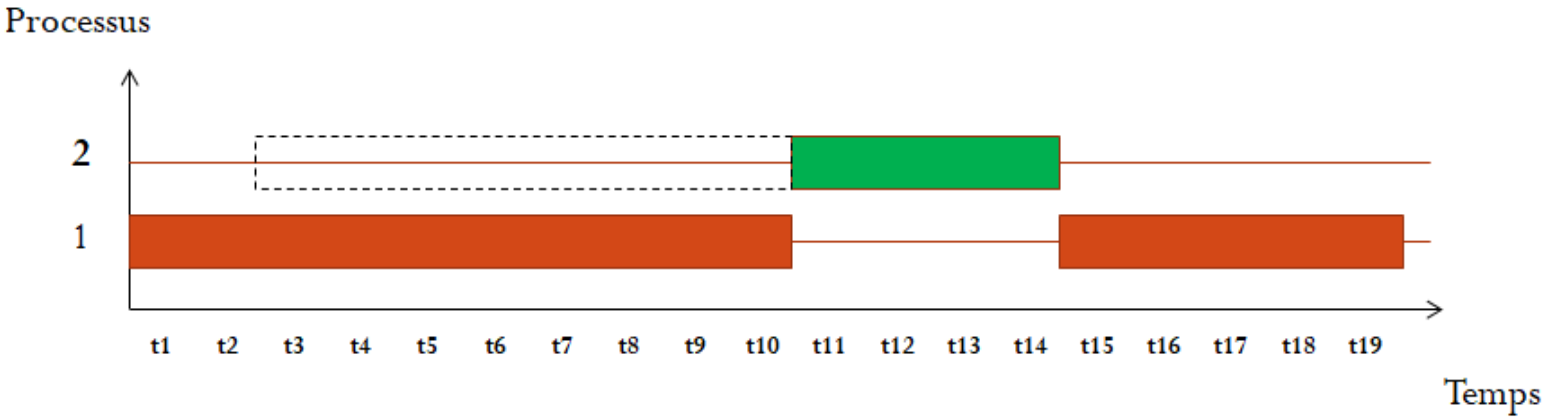
Processus	Temps de séjour	Temps d'attente
1	19	4
2	4	0
Temps moyen de séjour : 11,5		
Temps moyen d'attente : 2		



Exemple : Solution Round Robin (quantum = 10)

Processus	Durée	Temps d'arrivage
1	15	0
2	4	2

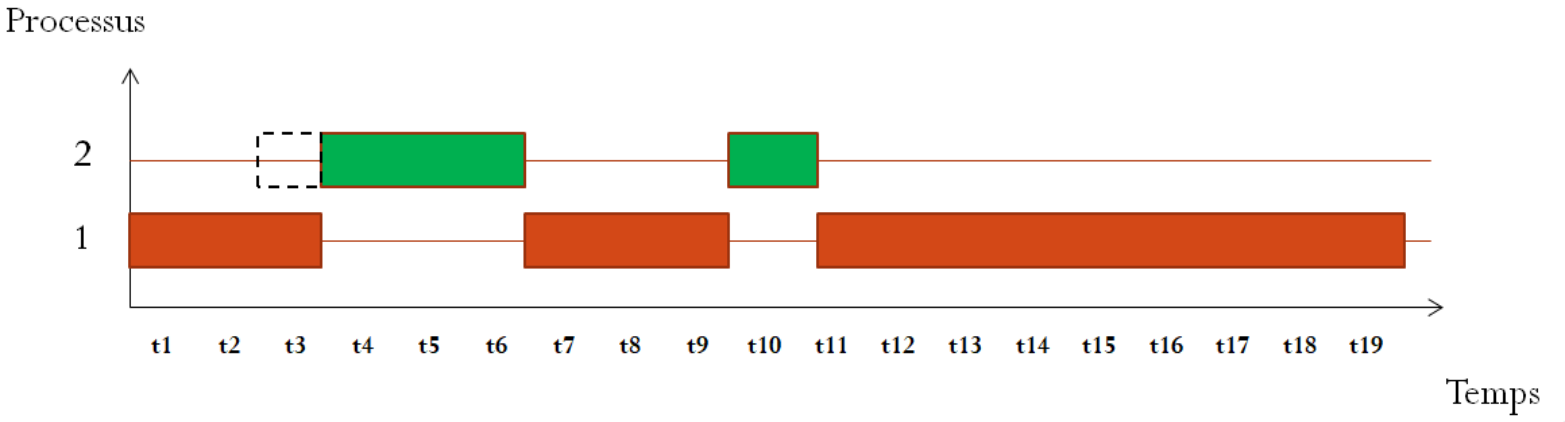
Processus	Temps de séjour	Temps d'attente
1	19	4
2	12	8
Temps moyen de séjour : 15,5		
Temps moyen d'attente : 6		



Exemple : Solution Round Robin (quantum = 3)

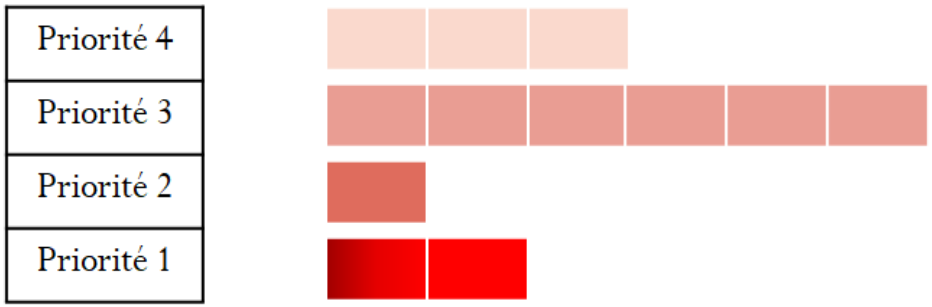
Processus	Durée	Temps d'arrivage
1	15	0
2	4	2

Processus	Temps de séjour	Temps d'attente
1	19	4
2	8	4
Temps moyen de séjour : 13,5		
Temps moyen d'attente : 4		



Ordonnancement à Priorité : Principe

- On associe une **priorité** à chaque processus.
- Il y a **autant de files** qu'il y a de **niveaux** de priorité.
- Les processus de **même priorité** sont regroupés dans une file du type **FIFO**.
- Le **choix** du processus à élire dépend des **priorités** des processus **prêts**.
- L'ordonnanceur **choisit** le processus le plus **prioritaire** qui se trouve **en tête** de file.
- Les processus de **même priorité** sont ordonnancés selon l'algorithme du **tourniquet**.



Evolution des Priorités

Pour empêcher les processus de priorité élevée de s'exécuter indéfiniment, l'ordonnanceur **diminue régulièrement la priorité** du processus en cours d'exécution.

- La **priorité du processus en cours est comparée** régulièrement à celle du processus **prêt le plus prioritaire** (en tête de file) ;
- Lorsqu'elle **devient inférieure**, la commutation a lieu.
- Le processus **suspendu est inséré en queue** de file correspondant à sa **nouvelle priorité** ;
- L'attribution et l'évolution des priorités dépendent des objectifs fixés et de beaucoup de paramètres.

Exemple

- Les processus qui font beaucoup d'E/S (qui sont souvent en attente) doivent acquérir le processeur dès qu'ils le demandent.
- Lorsqu'un processus passe de l'état **élu** à l'état **bloqué**, sa **priorité est recalculée**. Sa nouvelle valeur est le rapport : **quantum/temps réellement** utilisé par le processus.
- Les processus qui ont le **plus grand rapport sont les plus prioritaires** :
  - Si le quantum = 100 ms et le temps utilisé = 2 ms, la nouvelle priorité est 50.
  - Si le quantum = 100 ms et le temps utilisé = 50 ms, la nouvelle priorité est 2.

Files Multiples (quantum variable)

- Pour éviter qu'il y ait beaucoup de commutations pour les processus consommateurs de temps UCT, il est préférable d'allouer un plus grand quantum à ces processus.
- Lorsqu'un processus passe à l'état élu :
  - **Pour la première fois**, le processeur lui est alloué pendant **un** quantum.
  - **Pour la seconde fois**, le processeur lui est alloué pendant **2** quantum.
  - Pour la **n** fois le processeur lui est alloué pendant 2 n-1 quantum.
- Processus dont le **nombre** de quantum est le plus **petit** sont les plus **prioritaires**.
- Les processus prêts sont répartis selon leur priorité dans des files (FIFO).

Files de Priorité

- Il existe des classes de processus qui sont rangés dans différentes files de priorité.
  - **processus systèmes**,
  - **processus temps réel**,
  - **processus interactifs**,
  - **processus d'édition interactive**,
  - **processus « batch »**.
- Chaque classe est absolument **prioritaire sur celles du niveau inférieur**.
- À l'intérieur de chaque classe, les processus sont rangés dans un système de **files de priorité multi-niveaux**

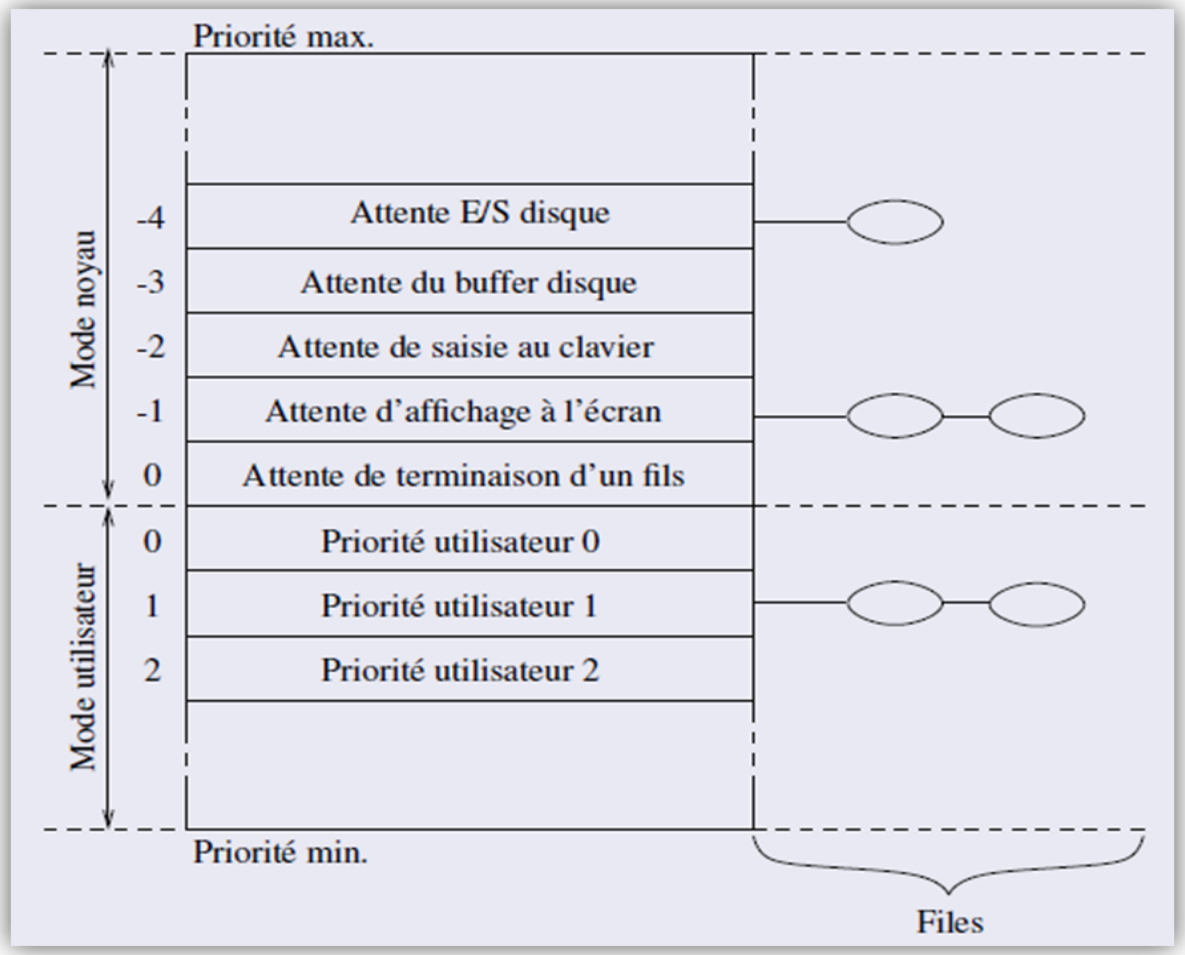
Cas d'étude :MS-DOS

- Mono-utilisateur :
  - Il peut y avoir, à un moment donné, plusieurs processus en mémoire mais un seul est à l'état prêt,
- Très simple :
  - Il exécute un processus jusqu'à ce qu'il se termine ou crée un ls. Le processus créateur est suspendu jusqu'à ce que le processus créé se termine.

Cas d'étude : Unix

- Utilisation de files à plusieurs niveaux, chaque file correspondant à un ensemble de priorités disjoint des autres ;
- Les processus sont séparés en deux parties :
  - Les processus qui s'exécutent en **mode noyau** (i.e. processus qui font des appels système).
  - Les processus **utilisateurs** (i.e. qui ne font pas d'appel système).
- Les processus **utilisateurs** ont des priorités à **valeurs positives**.
- Les processus qui s'exécutent en **mode noyau** ont des priorités à **valeurs négatives**.
- Les priorités **négatives** sont **prioritaires** par rapport aux valeurs **positives**.

Cas d'étude : Unix (suite)



Cas d'étude : Unix Description

- L'ordonnanceur parcourt les files en partant de la **priorité la plus haute(valeur la plus basse)** ;
- Dès qu'une **file non vide** est trouvée : le premier processus de la file est démarré ;
- Le processus s'**exécute** pendant un **quantum** de temps (typiquement **100ms**) sauf s'il se bloque avant ;
- Une fois le quantum terminé, le processus est placé en fin de file ;
- Les processus de même classe partagent donc le CPU suivant l'algorithme **Round Robin**.