

LES SYSTÈMES D'EXPLOITATION

GESTION DES PROCESSUS ET DES THREADS

🎓 3A - Coursus Ingénieurs - Dominante Informatique et Numérique
🏛️ CentraleSupélec - Université Paris-Saclay - 2025/2026



Idir AIT SADOUNE
idir.aitsadoune@centralesupelec.fr

PLAN

- La notion du processus
- La gestion des processus par l'OS
- La notion du thread
- L'ordonnancement
- La synthèse

[Retour au plan](#) - [Retour à l'accueil](#)

PLAN

- > La notion du processus
- > La gestion des processus par l'OS
- > La notion du thread
- > L'ordonnancement
- > La synthèse

[Retour au plan](#) - [Retour à l'accueil](#)

PROGRAMME vs PROCESSUS

- Un **programme** est un **ensemble d'instructions** (code exécutable) écrit dans un langage de programmation et **stocké sur un support** (disque, SSD, etc.).
 - **le programme est passif** → c'est **un fichier** sur le disque tant qu'il n'est pas exécuté.
 - ne consomme aucune ressource système tant qu'il n'est pas lancé.
- Un **processus** est un programme **en cours d'exécution**.
 - c'est une **entité active**, créée et gérée par le système d'exploitation.
 - **Exemple** → lorsque on ouvre le programme Firefox, il devient un processus.
- **Un programme** peut donner **naissance à plusieurs processus**.
 - **Exemple** → si on ouvre plusieurs fenêtres de Firefox, on aura plusieurs processus.
- **Un processus** est forcément **créé par un autre processus**
 - le système d'exploitation par exemple
 - l'appel système **fork()** sous **UNIX/Linux**

PROGRAMME vs PROCESSUS

QUELQUES EXEMPLES ...

- Dans **un OS moderne**, plusieurs processus **s'exécutent en parallèle** :
 - **les processus de l'OS** (gestion du réseau, gestion des utilisateurs, ...)
 - le **shell** (toute **l'interface graphique** → plusieurs processus).
 - l'IDE **VSCode** avec lequel je tape ce cours.
 - le navigateur **Chrome** qui me permet de visualiser ce cours.
- Commande **top** sous **Unix/Linux** → afficher en temps réel la liste des processus actifs et l'utilisation des ressources du système (CPU, mémoire, etc.).

```
$ top -stats command,pid,ppid,cpu,pstate
```

COMMAND	PID	PPID	%CPU	STATE
launchd	1	0	8.5	sleeping
top	74562	34386	3.7	running
Terminal	34311	1	3.0	sleeping
WindowServer	169	1	2.5	sleeping
...				

PLAN

- La notion du processus
- La gestion des processus par l'OS
- La notion du thread
- L'ordonnancement
- La synthèse

[Retour au plan](#) - [Retour à l'accueil](#)

RÔLE DE L'OS

- **Création et suppression de processus**
 - Programme → processus
 - **Munir** le programme des **informations** nécessaires pour son **exécution**
- **Suspension et reprise**
 - Multiprogrammation → **interrompre et reprendre** les processus
 - **Gestion de la mémoire** où sont stockées les processus interrompus
- **Communication et synchronisation**
 - Partage de données entre plusieurs processus
 - **Consistance** de l'état de la mémoire

RÔLE DE L'OS

- **Création et suppression de processus**
 - Programme → processus
 - Munir le programme des **informations** nécessaires pour son **exécution**
- **Suspension et reprise**
 - Multiprogrammation → **interrompre et reprendre** les processus
 - **Gestion de la mémoire** où sont stockées les processus interrompus
- **Communication et synchronisation**
 - Partage de données entre plusieurs processus
 - **Consistance** de l'état de la mémoire

CRÉATION DE PROCESSUS

Rappel → un processus est forcément créé par un autre processus

L'**OS** crée un processus à la demande :

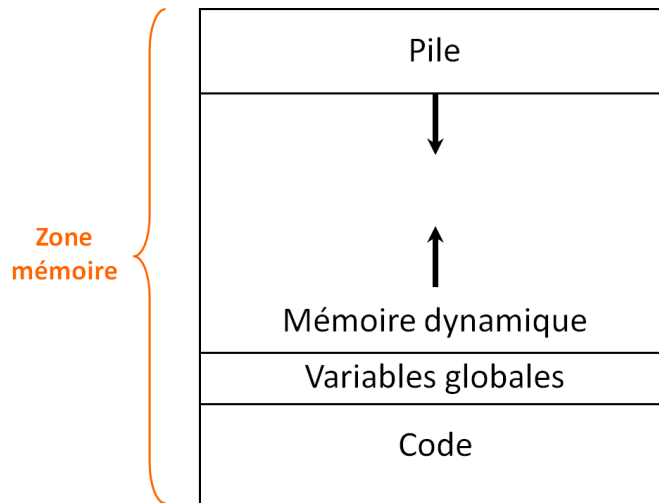
- Lorsqu'un **utilisateur** lance un programme.
- Lorsqu'un **processus** engendre un autre processus:
 - **sous UNIX/Linux** → 2 **appels système**
 - ➡ **fork** pour créer un processus à partir du processus courant (le processus est **dupliqué**)
 - ➡ **exec** pour **remplacer** le processus courant par un autre processus
 - **sous WINDOWS**
 - ➡ **createprocess** pour créer un processus (le processus courant est **conservé**)

Lorsque l'**OS** crée un **processus** → il charge le code **en mémoire**, initialise les **structures (PCB)**, et le met dans la file des **processus prêts** à exécuter.

L'ESPACE MÉMOIRE D'UN PROCESSUS

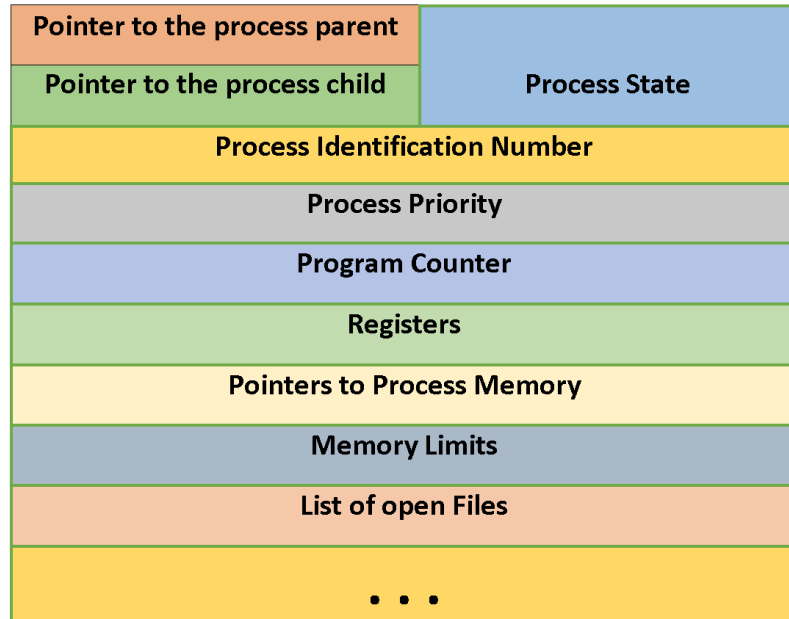
Le processus a **son propre espace mémoire**, son compteur ordinal (PC), ses registres, ses descripteurs de fichiers, etc.

- **Code exécutable** en lecture seule (taille connue)
- **Variables/Constantes globales** (taille connue)
- **Pile** pour gérer les contextes et les variables temporaires (taille inconnue)
- **Le TAS** ou la **Zone d'allocation dynamique de mémoire** (taille inconnue)



PROCESS CONTROL BLOCK - PCB

Structure de données contenant des informations utilisée par le système d'exploitation pour **définir** et **gérer un processus**.



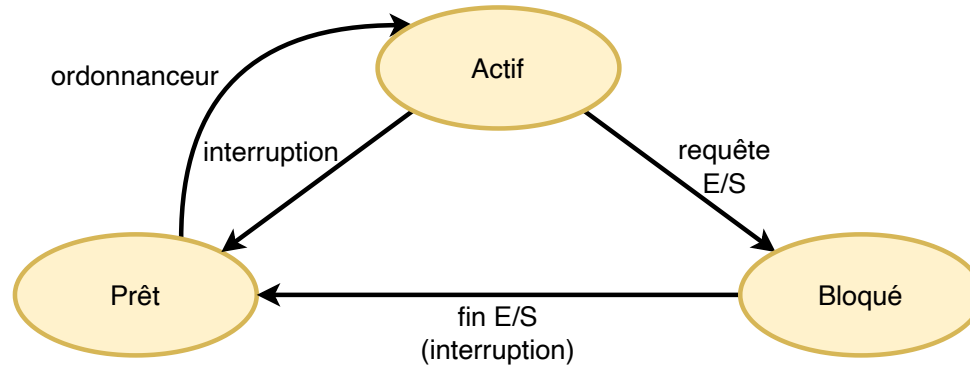
Created by Notes Jam

Tout ce qui doit être sauvegardé pour **interrompre** puis **reprendre** l'exécution d'un processus.

RÔLE DE L'OS

- **Création et suppression de processus**
 - Programme → processus
 - **Munir** le programme des **informations** nécessaires pour son **exécution**
- **Suspension et reprise**
 - Multiprogrammation → **interrompre et reprendre** les processus
 - **Gestion de la mémoire** où sont stockées les processus interrompus
- **Communication et synchronisation**
 - Partage de données entre plusieurs processus
 - **Consistance** de l'état de la mémoire

CYCLE DE VIE DU PROCESSUS

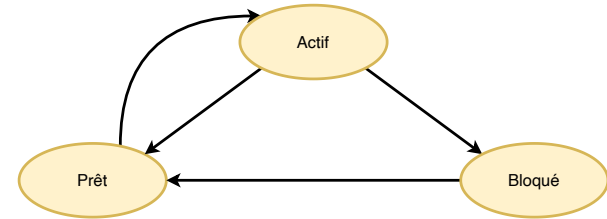


[0-1] processus en exécution, [0-n] processus prêts, [0-n] processus en attente

SUSPENSION DE L'EXÉCUTION

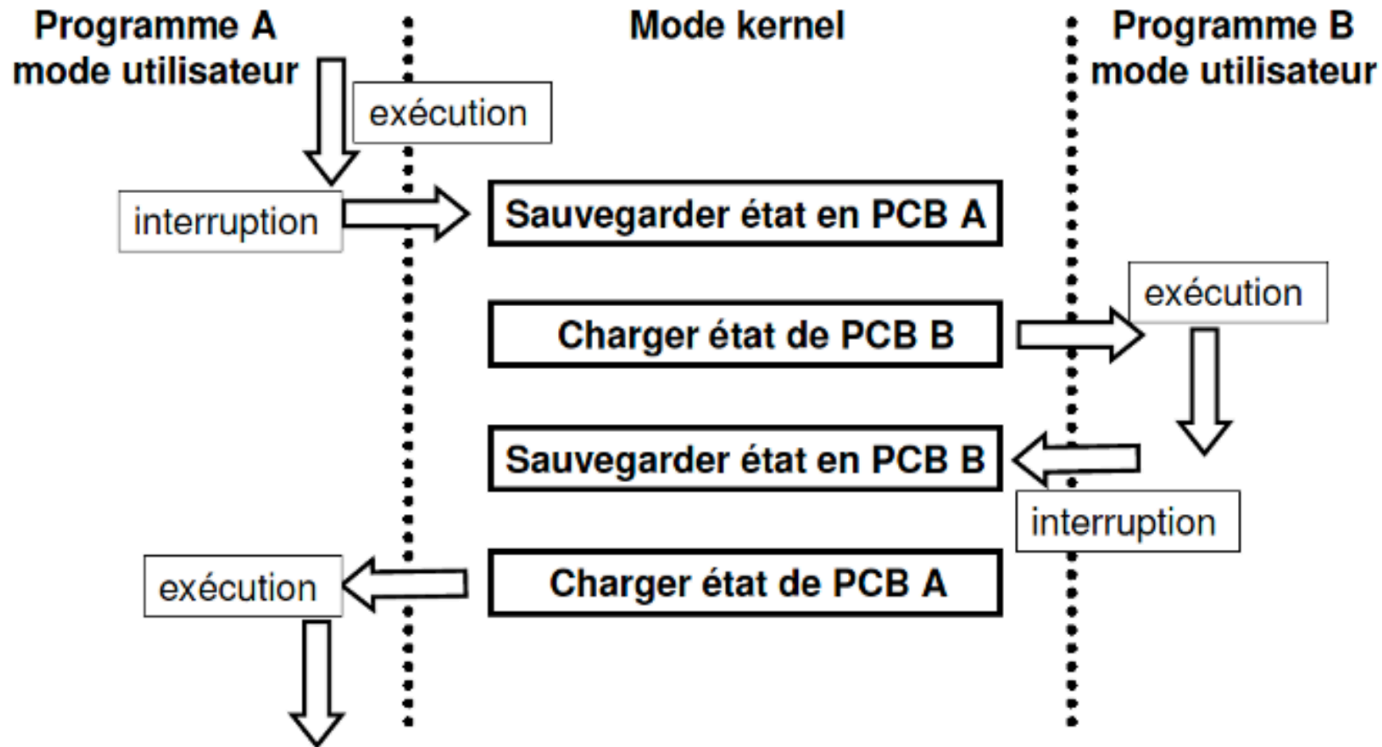
- Le processus en exécution laisse la main si:

- ▢ son quantum a expiré → état **Prêt**
- ▢ crée un processus fils (**fork**) → état **Prêt**
- ▢ fait une demande d'E/S → état **Bloqué**
- ▢ exécute **wait** → état **Bloqué**



COMMUTATION DE PROCESSUS

Changements de contexte



RÔLE DE L'OS

- **Création et suppression de processus**
 - Programme → processus
 - **Munir** le programme des **informations** nécessaires pour son **exécution**
- **Suspension et reprise**
 - Multiprogrammation → **interrompre et reprendre** les processus
 - **Gestion de la mémoire** où sont stockées les processus interrompus
- **Communication et synchronisation**
 - Partage de données entre plusieurs processus
 - **Consistance** de l'état de la mémoire

ACTIONS DE L'OS

- **Mémoire** → chaque processus a son propre espace mémoire
 - ▢ pas de problème de consistance mémoire/processeur
- **Verrous (locks)** → un processus verrouille l'accès à une ressource
 - ▢ gestion de l'accès à une **section critique**
 - ▢ file d'attente pour l'accès à la ressource
- **Outils** et Algorithmes de **synchronisation**
 - ▢ l'**OS** propose différents outils de synchronisation
 - ▢ files de messages, mémoire partagée, tubes (**pipes**), sockets, etc.
 - ▢ voir cours Synchronisation des processus (**prochain chapitre**)

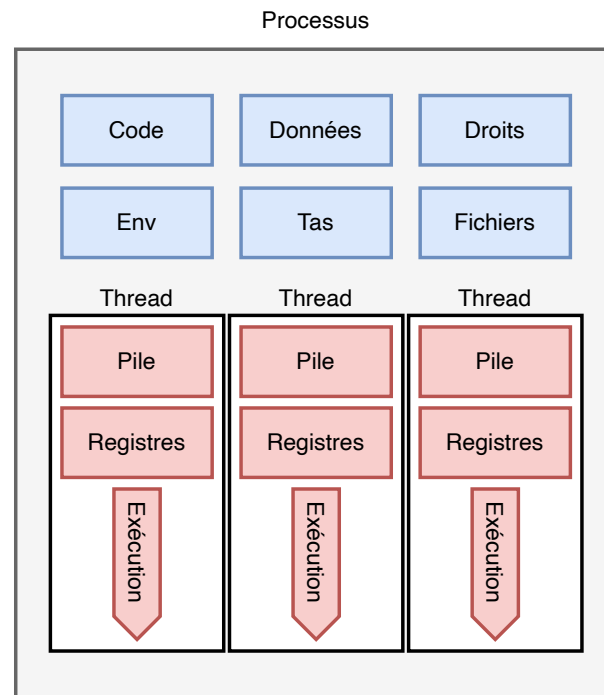
PLAN

- La notion du processus
- La gestion des processus par l'OS
- La notion du thread
- L'ordonnancement
- La synthèse

[Retour au plan](#) - [Retour à l'accueil](#)

THREAD vs PROCESSUS

- **Un thread** est l'unité d'exécution de base d'un processus, décrit par son point d'exécution et son état interne (registres, pile, ...).
 - ➡ un processus peut avoir plusieurs threads
 - ➡ les threads partagent le code et les données du processus (l'espace mémoire du processus)
 - ➡ chaque thread a sa propre pile et partage les mêmes variables globales avec les autres threads
- **Un thread** est également appelé processus léger.
- **Exemple** → un navigateur web
 - ➡ le navigateur entier = un processus.
 - ➡ chaque onglet ou tâche = un thread à l'intérieur du processus.



UTILISATION DES THREADS

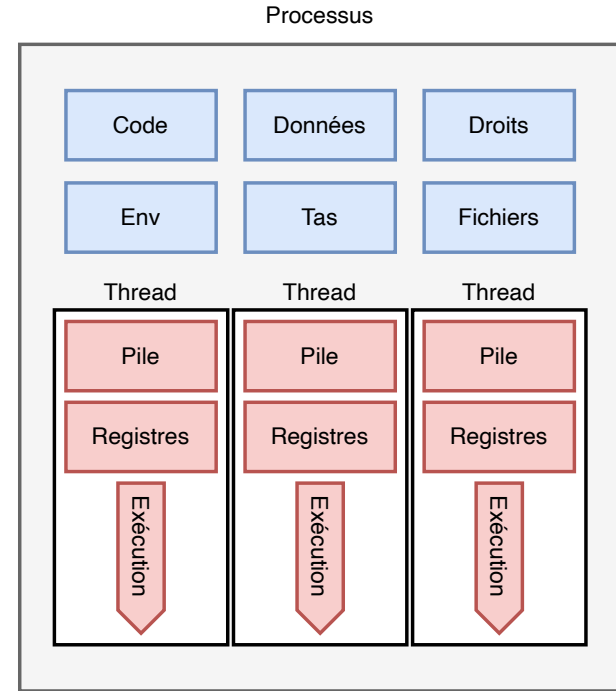
- Le **Thread** permet la gestion de **plusieurs traitements en parallèle** dans le même processus.

➡ **création** d'un thread \equiv exécuter une fonction en parallèle par **un appel système** :

sous **Unix** \rightarrow `pthread_create()`

sous **Windows** \rightarrow `CreateThread()`

- ➡ le passage de ressources entre threads facilité.
 - ➡ les variables dans le contexte du même processus.
- Performances améliorées** par rapport aux processus :
 - ➡ création plus rapide;
 - ➡ changement de contexte plus rapide;
 - ➡ partage du code \rightarrow gain de place en mémoire;
 - ➡ réactivité \rightarrow le processus s'exécute pendant qu'un thread est en attente.



PLAN

- La notion du processus
- La gestion des processus par l'OS
- La notion du thread
- L'ordonnancement
- La synthèse

[Retour au plan](#) - [Retour à l'accueil](#)

MULTIPROGRAMMATION vs TEMPS PARTAGÉ

- **Multiprogrammation**

- le **but** → maximiser l'utilisation du **CPU** (éviter qu'il reste inactif)
- la capacité d'un **OS** à garder plusieurs processus en mémoire.

- **Temps partagé (time sharing)**

- le **but** → améliorer la réactivité et le partage équitable du **CPU**.
- le **CPU** est partagé entre plusieurs processus.
- chaque processus utilise le **CPU** pendant un **quantum de temps**.
- l'**OS** alterne rapidement entre processus
→ donner l'illusion qu'ils s'exécutent simultanément

- L'**OS** gère ces mécanismes via :

- un **ordonnanceur** (**scheduler**) décidant quel processus obtient le **CPU**
- des **files d'attente** de processus (**prêt**, **actif**, et **bloqué**),
- des **interruptions** permettant de reprendre le contrôle à la fin d'un quantum.

L'ORDONNANCEMENT

- L'**ordonnement** est le mécanisme par lequel l'**OS** décide quel **processus** obtient le **CPU** et pendant combien de temps (**quantum de temps**).
- Dans l'**analyse d'un ordonnancement**, on ne s'intéresse pas à la durée totale d'un processus ... mais au **temps** pendant lequel il va **garder le CPU** :
 - jusqu'à ce qu'il **termine** (fin d'exécution)
 - jusqu'à ce qu'il **fasse une E/S** (→ à l'état **bloqué**)
 - jusqu'à ce que l'**OS** décide que **ce n'est plus son tour** (fin de **quantum**, → à l'état **actif**)
- **Le remplacement d'un processus** en cours d'exécution a un coût (**commutation de contexte**)
 - exécution de la **routine d'ordonnement**
 - sauvegarde du **contexte** (registres + **PC**)
 - chargement d'un nouveau **contexte**

LES CRITÈRES D'UN ORDONNANCEMENT

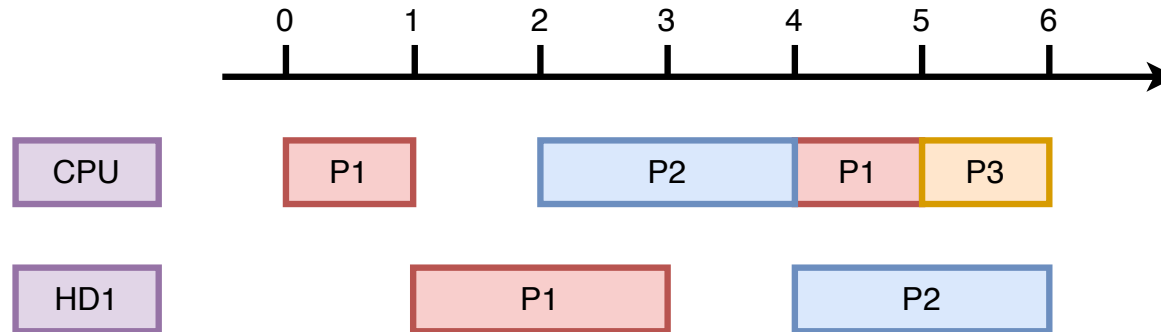
- être **équitable** (**fairness**) vis-à-vis des processus ;
- maximiser l'utilisation globale du processeur (**efficace**) ;
- avoir un comportement le plus **prévisible** possible ;
- permettre un maximum d'utilisateurs interactifs (**réactif**) ;
- **minimiser le surcoût** (**overhead**) lié à la parallélisation ;
- assurer une **utilisation maximale** des ressources ;
- gérer convenablement les **priorités**.

Objectif → choisir un **algorithme d'ordonnancement** qui **minimise** ou **maximise un critère**.

LES CRITÈRES D'UN ORDONNANCEMENT

LE TAUX D'UTILISATION

Proportion de temps pendant lequel le CPU est utilisée

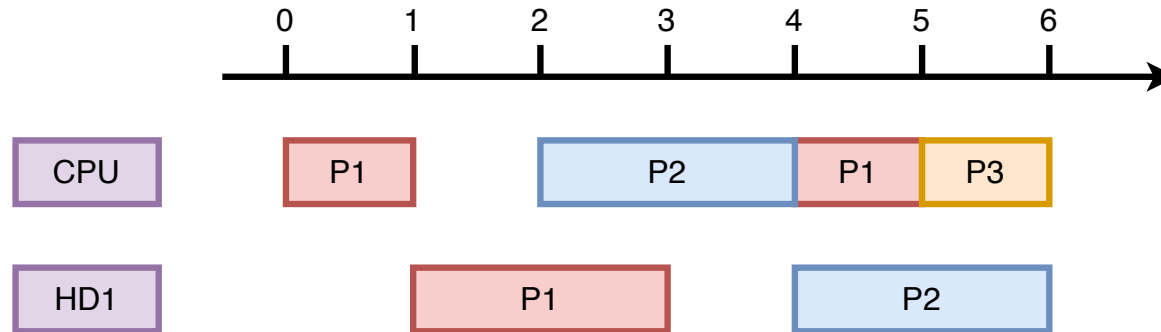


$$taux = \frac{5}{6} = 83\% \rightarrow \text{à maximiser}$$

LES CRITÈRES D'UN ORDONNANCEMENT

LE DÉBIT

Nombre moyen de processus traités par unité de temps

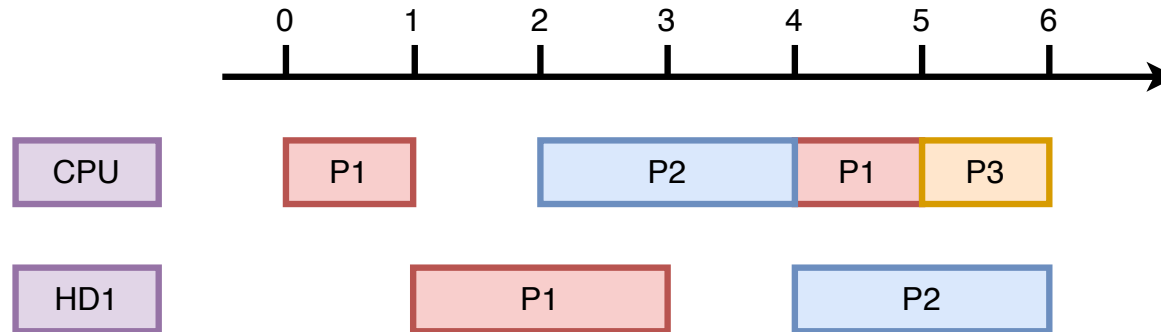


$$nb = \frac{3}{6} = 0.5 \rightarrow \text{à maximiser}$$

LES CRITÈRES D'UN ORDONNANCEMENT

LE TEMPS D'ATTENTE

Temps total passé par tous les processus dans la file prêt

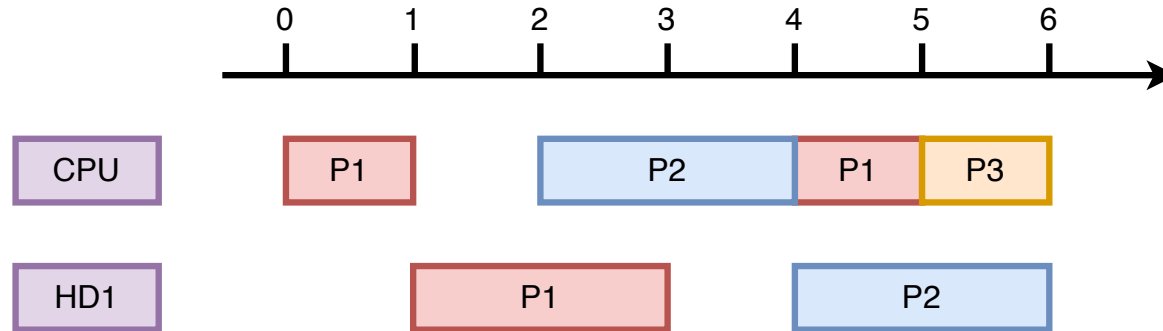


$$\text{moyenne} = \frac{1+2+5}{3} = 2.66 \rightarrow \text{à minimiser}$$

LES CRITÈRES D'UN ORDONNANCEMENT

LA ROTATION

Durée d'un processus → **temps de réponse** du processus
(*date_termination* – *date_creation*)



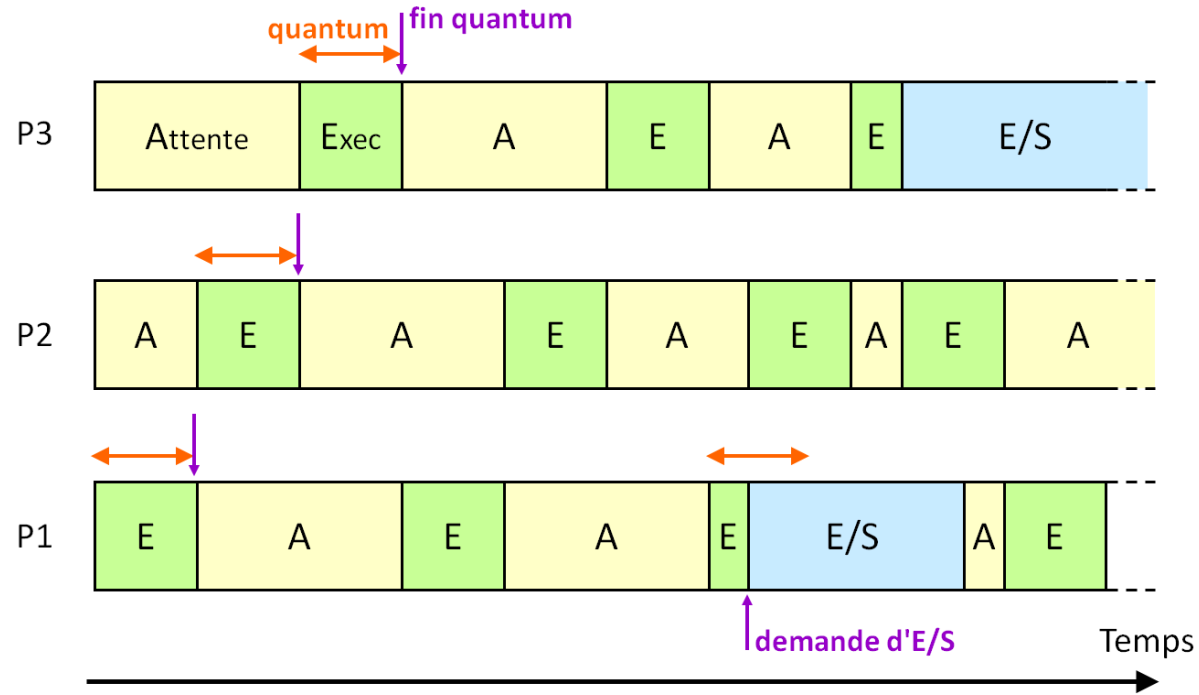
$$moyenne = \frac{5+6+6}{3} = 5.66 \rightarrow \text{à minimiser}$$

ORDONNANCEMENT

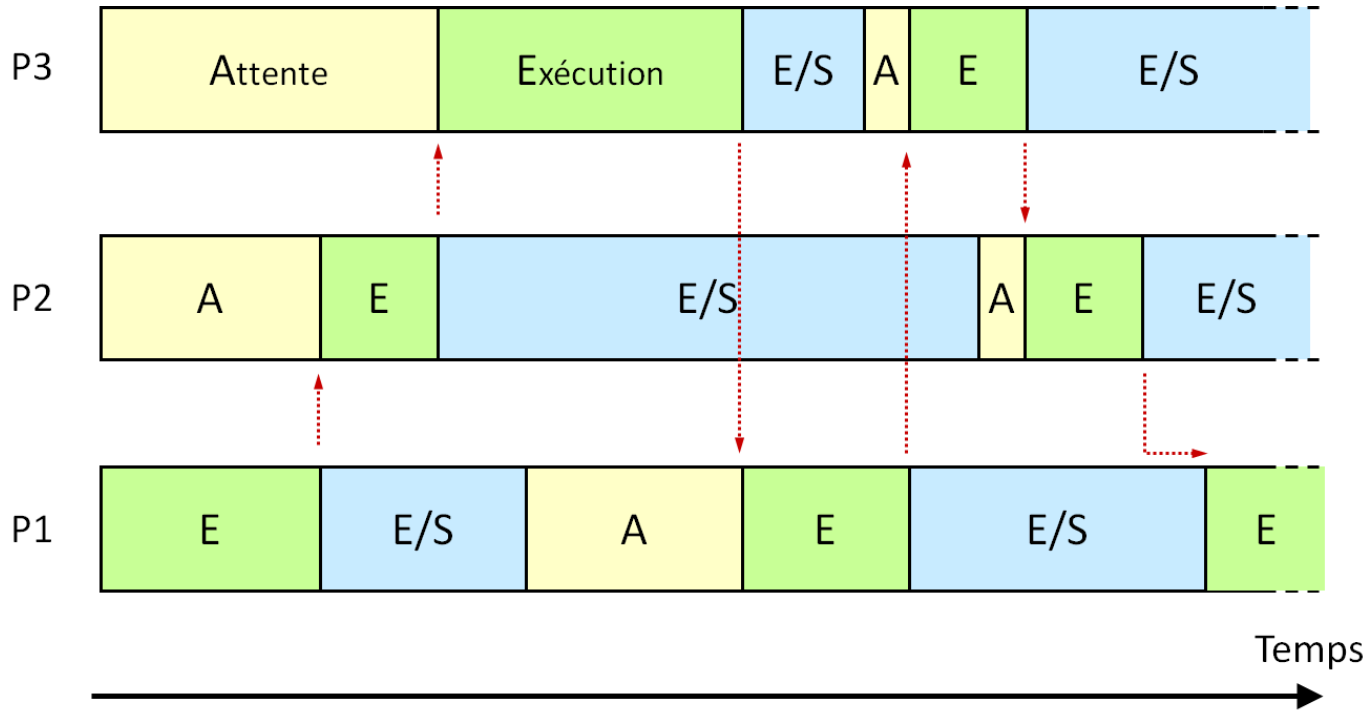
PRÉEMPTIF vs NON PRÉEMPTIF

- Ordonnancement **préemptif** (**avec réquisition**)
 - l'**OS** peut interrompre un processus en cours pour en exécuter un autre :
 - quand le **quantum** est écoulé
 - lorsqu'**un processus plus prioritaire** demande à utiliser le **CPU**
 - assure une **meilleure réactivité**
 - indispensable pour les **systèmes temps réel** ou les **systèmes interactifs**.
- Ordonnancement **non préemptif**
 - un processus **garde le CPU** jusqu'à ce que :
 - il se termine
 - il se bloque (en attente d'une ressource)
 - plus **simple**, mais moins réactif.

ORDONNANCEMENT PRÉEMPTIF



ORDONNANCEMENT NON PRÉEMPTIF



DE NOMBREUSES STRATÉGIES

FIFO SANS PRÉEMPTION

- **Principe:**

➡ premier arrivé, premier servi.

- **Avantages :**

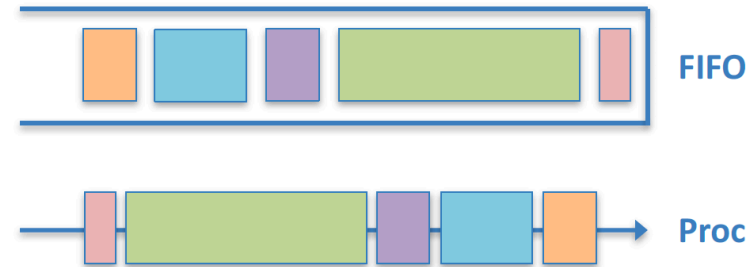
✓ Simple à implémenter

✓ Équitable dans l'ordre d'arrivée

- **Inconvénients :**

✗ Peu efficace → des processus ont "longtemps" le processeur

✗ Peu réactif → des processus peuvent attendre longtemps



DE NOMBREUSES STRATÉGIES PLUS COURT D'ABORD

- **Principe :**

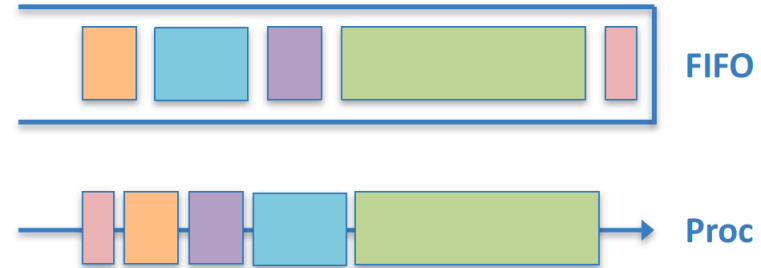
➡ priorité au processus le plus court

- **Avantages :**

- ✓ Réactif → avantage aux petits processus
- ✓ Optimal sur le temps d'attente moyen

- **Inconvénients :**

- ✗ Pas efficace → les processus ont moins le processeur s'il y a beaucoup d'E/S
- ✗ Non équitable → on peut avoir une **famine** des gros processus



DE NOMBREUSES STRATÉGIES

ROUND-ROBIN (FIFO PRÉÉMPTIF)

- **Principe :**

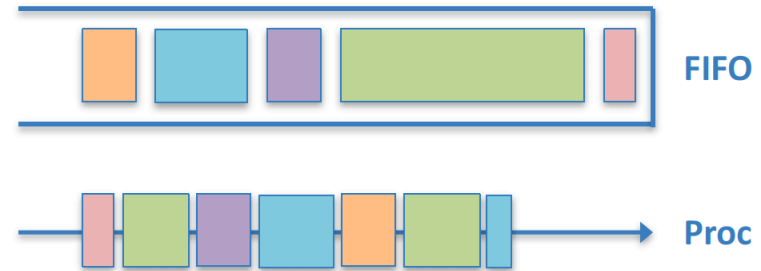
➡ **FIFO** avec **quantum** de temps

- **Avantages :**

- ✓ Équitable → tout le monde a le processeur
- ✓ Réactif → les processus n'attendent pas

- **Inconvénients :**

- ✗ Temps d'attente moyen plus élevé
- ✗ Beaucoup de commutations → surcoût!



DE NOMBREUSES STRATÉGIES

FILES DE PRIORITÉS

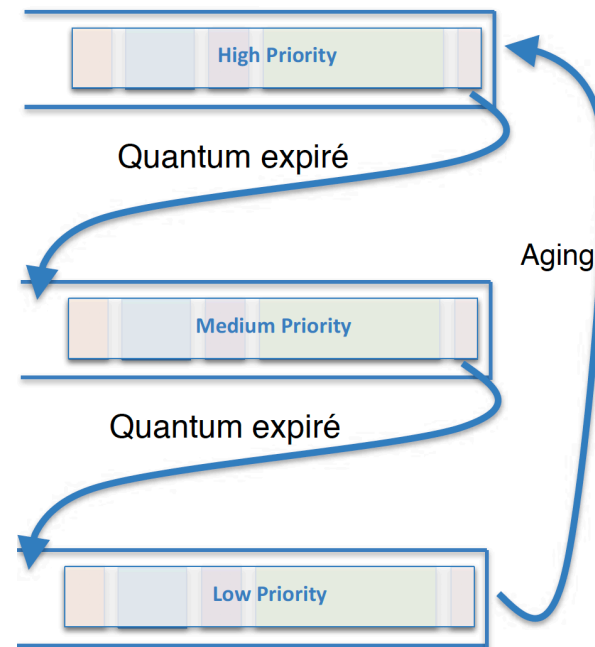
- Plusieurs files d'attente
- La durée des quantum peut dépendre de la file
- ✓ favorise le temps de réponse des processus systèmes
- ✗ Comment choisir la priorité ?



DE NOMBREUSES STRATÉGIES

FILES DE PRIORITÉS DYNAMIQUES

- La priorité d'un processus peut être modifiée par l'OS
 - par exemple sortie d'attente I/O
 - attente longue en file basse priorité
- **Utilité** → Windows NT, MacOS X, Linux
- Notion de **classe de priorités**



EXEMPLE - SOLARIS

OS DES MACHINES SUN ENTRE 1993 ET 2000

Principe

- Quantum de temps selon priorité (0 = priorité max)
- Priorité modifiée à la fin du quantum ou après une E/S
 - ▢▶ prioritaire → grand quantum
 - ▢▶ quantum consommé → priorité augmentée
 - ▢▶ E/S → priorité diminuée

EXEMPLE - WINDOWS XP ET APRÈS

Principe

- Priorité + Round Robin
 - ▢➡ gérée au niveau des threads uniquement
 - ▢➡ 32 niveaux de priorité
 - ▢➡ ordonnancement préemptif par niveau de priorité
- Priorité dynamique
 - ▢➡ baissée à la fin du quantum
 - ▢➡ remontée après chaque E/S → interface graphique plus réactive!

EXEMPLE - LINUX, MACOS X

Principe

- 2 algorithmes:
 1. Tâches **temps réel** → préemptif selon priorité, **FIFO** ou **RR** par priorité
 2. Autres tâches → temps partagé équitable
- Système de crédits
 - ▢▶ chaque processus dispose d'un **crédit** = sa priorité
 - ▢▶ le processus le plus riche l'emporte (préemptif)
 - ▢▶ perte de 1 crédit à la fin du quantum
 - ▢▶ si aucun processus **prêt** n'a de crédit, **tous** les processus sont re-crédités
→ $credit' = credit/2 + credit_{init}$

PLAN

- La notion du processus
- La gestion des processus par l'OS
- La notion du thread
- L'ordonnancement
- La synthèse

[Retour au plan](#) - [Retour à l'accueil](#)

SYNTHÈSE

- Un **processus** est un programme exécuté par un processeur
- L'OS stocke les informations sur les processus dans un **PCB**
- L'OS assure la **consistance** des données en mémoire
- Les **threads** partagent le code, l'environnement et le tas
- **Ordonnancement** = choix d'un processus à exécuter
- Algorithmes d'ordonnancement :
 - FIFO
 - Plus court d'abord
 - Round-Robin

MERCI

[Version PDF des slides](#)

[Retour à l'accueil](#) - [Retour au plan](#)