SYSTÈMES D'EXPLOITATION

GESTION DES PROCESSUS ET DES THREADS

🎓 3A - Cursus Ingénieurs - Dominante Informatique et Numérique
🏛 CentraleSupelec - Université Paris-Saclay - 2024/2025

Idir AIT SADOUNE
idir.aitsadoune@centralesupelec.fr

PLAN

Retour au plan - Retour à l'accueil

PLAN

Retour au plan - Retour à l'accueil

DÉFINITIONS

- Un **programme informatique** : une suite statique d'instructions.
- Un **processeur** : un **automate** (électronique) de **traitement**.
  - Il peut exécuter un programme
  - Il modifie son état en fonction des instructions
- Un **processus** : un programme exécuté par un processeur.
  - capte le **caractère dynamique** d'un programme.

PROGRAMME vs PROCESSUS

- Un **processus** : un programme exécuté par un processeur.
  - capte le **caractère dynamique** d'un programme.
- **Un programme** peut donner naissance à plusieurs processus.
- **Un processus** est forcément créé par un autre processus
  (le système d'exploitation par exemple)

Exemples de processus

- Utilisation d'un logiciel de traitement de texte
- Compilation de code source
- Tâches système (envoi de données vers l'imprimante)

PROCESSUS

- 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.

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

COMMAND             PID     PPID    %CPU   STATE
com.docker.hyper    674     667     34.9   sleeping
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

Retour au plan - Retour à l'accueil

RÔLE DE L'OS

- **Création et suppression de processus**
  - Programme $\longrightarrow$ processus
  - **Munir** le programme des **informations** nécessaires pour son **exécution**
- **Suspension et reprise**
  - Multiprogrammation $\longrightarrow$ **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 $\longrightarrow$ processus
  - **Munir** le programme des **informations** nécessaires pour son **exécution**
- **Suspension et reprise**
  - Multiprogrammation $\longrightarrow$ **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

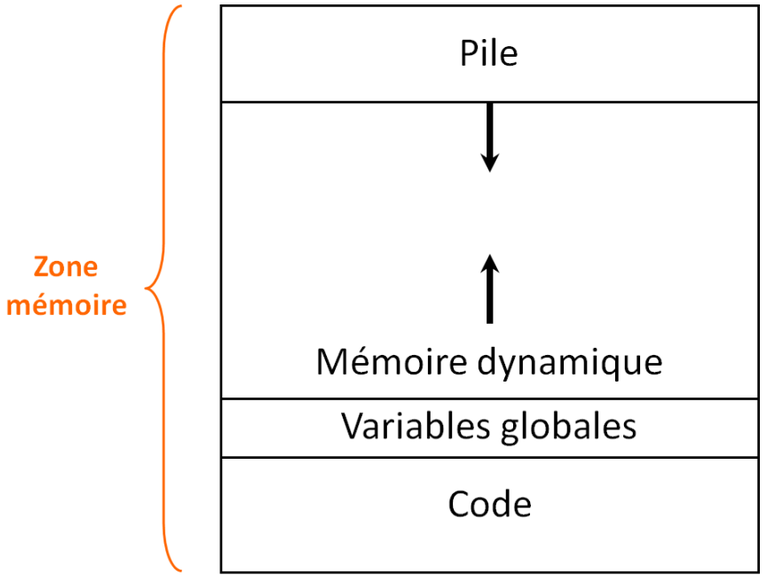
CRÉATION DE PROCESSUS

Rappel : un processus est forcément créé par un autre processus

- **Sous UNIX** $\longrightarrow$ 2 appels système
  - `fork` pour créer un processus à partir du processus courant
    - ➠ le processus courant est dupliqué
  - `exec` pour **remplacer** le processus courant par un autre processus

- **Sous WINDOWS**
  - `createprocess` pour créer un processus (**cf.** `exec` **Unix**)
    - ➠ le processus courant est **conservé**

L'ESPACE MÉMOIRE D'UN PROCESSUS

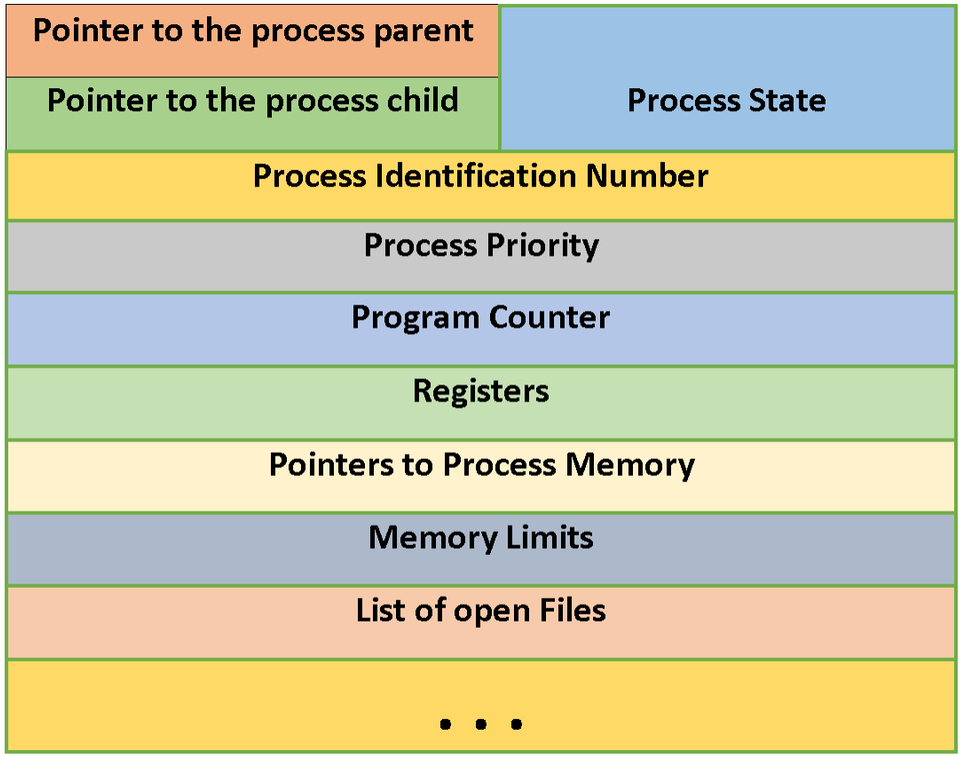


- **Code exécutable** en lecture seule (taille connue)
- **Variables/Constantes globales** (taille connue)
- **Pile** pour gérer les contextes et les variables temporaires (taille inconnue)
- **Le TAS** ou la **Zone d'allocation dynamique de mémoire** (taille inconnue)

BLOC DE CONTRÔLE DE PROCESSUS

Process Control Block - PCB

- **Structure de données** contenant les informations relatives à un processus utilisée par l'OS pour la **gestion des processus**.

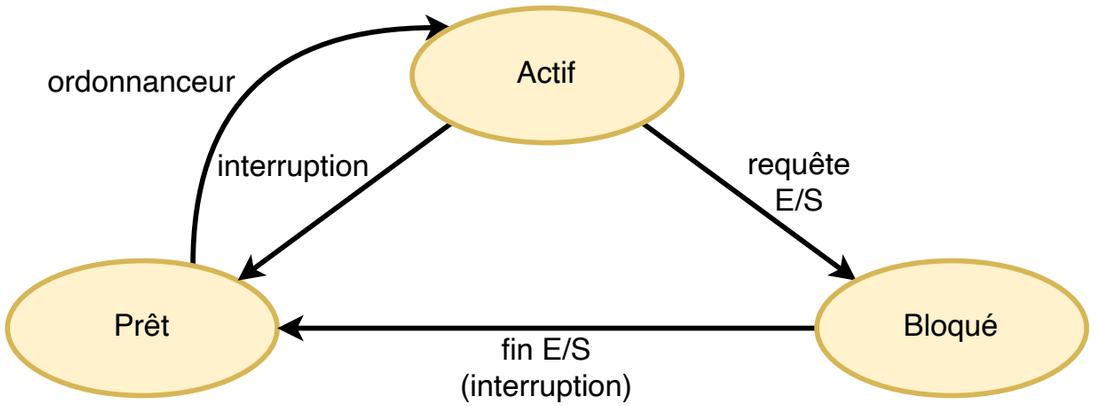


Tout ce qui doit être sauvegardé pour **interrompre** puis **reprendre** l'exécution d'un processus.

RÔLE DE L'OS

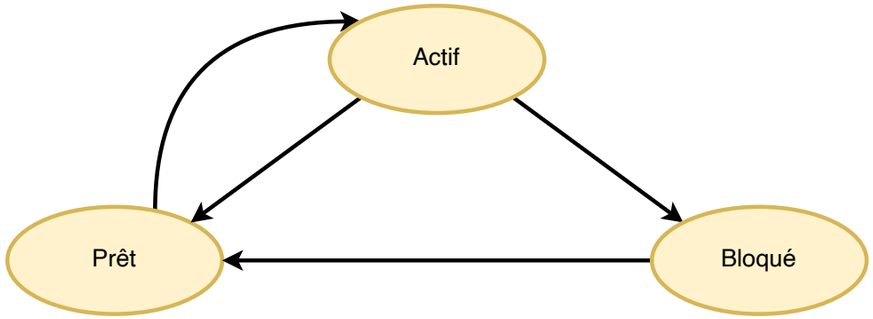
- **Création et suppression de processus**
  - Programme $\longrightarrow$ processus
  - **Munir** le programme des **informations** nécessaires pour son **exécution**
- **Suspension et reprise**
  - Multiprogrammation $\longrightarrow$ **interrompre et reprendre** les processus
  - **Gestion de la mémoire** 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

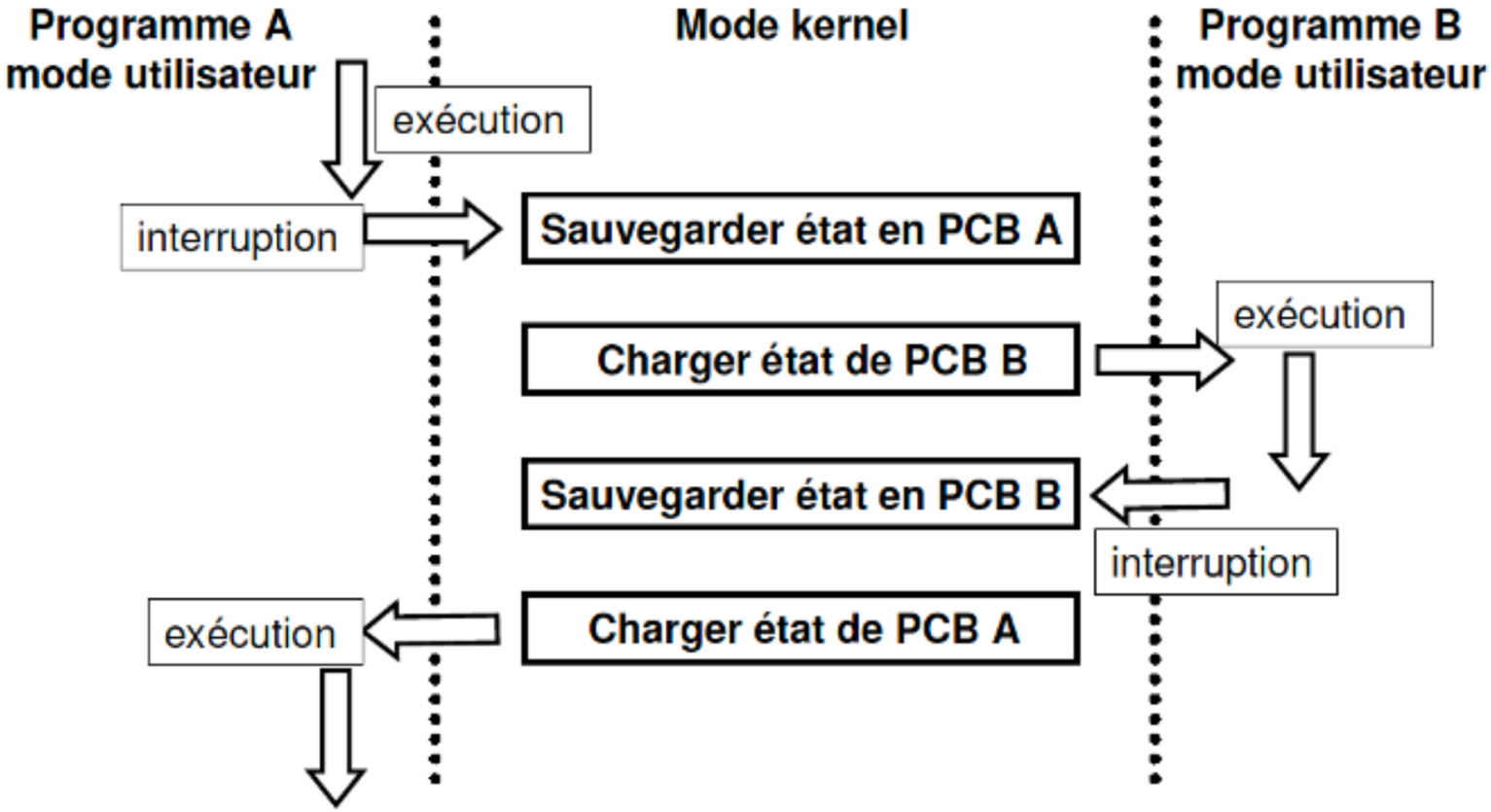
SUSPENSION DE L'EXÉCUTION



- Le processus en exécution laisse la main si:
  - son quantum a expiré → **Prêt**
  - crée un processus fils → **Prêt**
  - fait une demande d'E/S → **Bloqué**
  - exécute `wait` → **Bloqué**

COMMUTATION DE PROCESSUS

Changements de contexte



RÔLE DE L'OS

- **Création et suppression de processus**
  - Programme $\longrightarrow$ processus
  - **Munir** le programme des **informations** nécessaires pour son **exécution**
- **Suspension et reprise**
  - Multiprogrammation $\longrightarrow$ **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**: un processus peut verrouiller l'accès à une ressource
  - ➠ file d'attente pour l'accès à la ressource
- Outils et Algorithmes de **synchronisation**
  - ➠ voir cours Synchronisation des processus

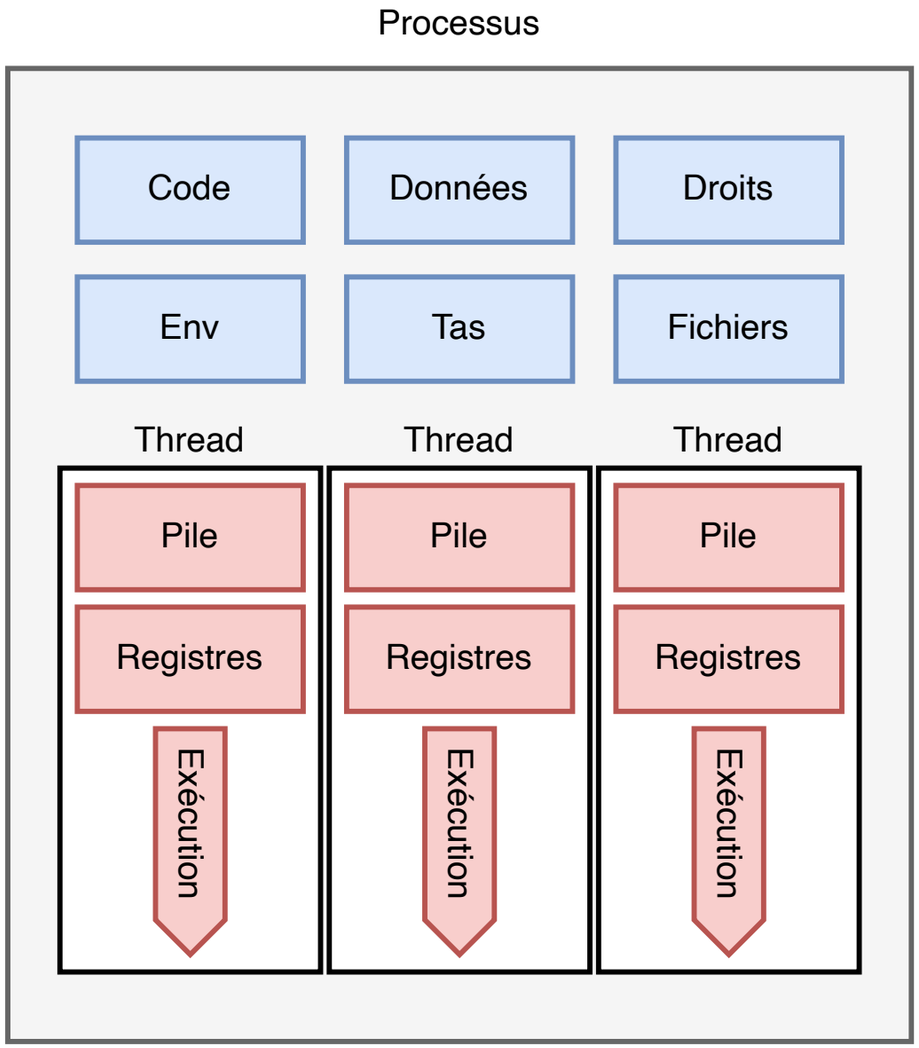
PLAN

Retour au plan - Retour à l'accueil

NOTION DE THREAD

- **Un thread** est l'unité d'exécution de base d'un processus, décrite par son point d'exécution et son état interne (registres, pile, ...).
  - les threads partagent le **même code et les mêmes données**
  - chaque thread a **sa propre pile**
  - **un processus** peut avoir **plusieurs threads**
- **Un thread** partage l'espace mémoire du processus qui l'a créé.
- **Un thread** est également appelé **processus léger**.

NOTION DE THREAD



UTILISATION DES THREADS

- Le **Thread** permet la gestion de **plusieurs traitements en parallèle** dans le même processus.
  - ➠ passage de ressources entre threads facilité.
  - ➠ les variables sont dans le contexte du même processus.

- **Performances améliorées** par rapport aux processus :
  - création plus rapide;
  - changement de contexte plus rapide;
  - partage du code $\longrightarrow$ gain de place en mémoire;
  - réactivité $\longrightarrow$ le processus s'exécute pendant qu'un thread est en attente.

PLAN

Retour au plan - Retour à l'accueil

MULTIPROGRAMMATION ET TEMPS PARTAGÉ

- **Les processus** sont répartis sur les ressources :
  - **Plusieurs processus** peuvent vouloir la **même ressource** en même temps
  - File d'attente de **PCB**
  - Choisir un processus parmi tous les processus dans la file d'attente

- **Exemple**
  - **le processeur** est **une ressource hautement critique**.
  - l'**OS** est en charge de sa répartition entre les processus
    ➠ **l'ordonnancement (scheduling)**

ORDONNANCEMENT

- On ne s'intéresse pas à la durée totale du processus ...
  mais au **temps** pendant lequel il va **garder le processeur** :
  - ➠ jusqu'à ce qu'il termine
  - ➠ jusqu'à ce qu'il fasse une E/S
  - ➠ jusqu'à ce que l'**OS** décide que ce n'est plus son tour

- **Le remplacement d'un processus** en exécution a un coût (**commutation de contexte**)
  - exécution de la routine d'ordonnancement
  - sauvegarde du contexte (registres + **PC**)
  - chargement d'un nouveau contexte

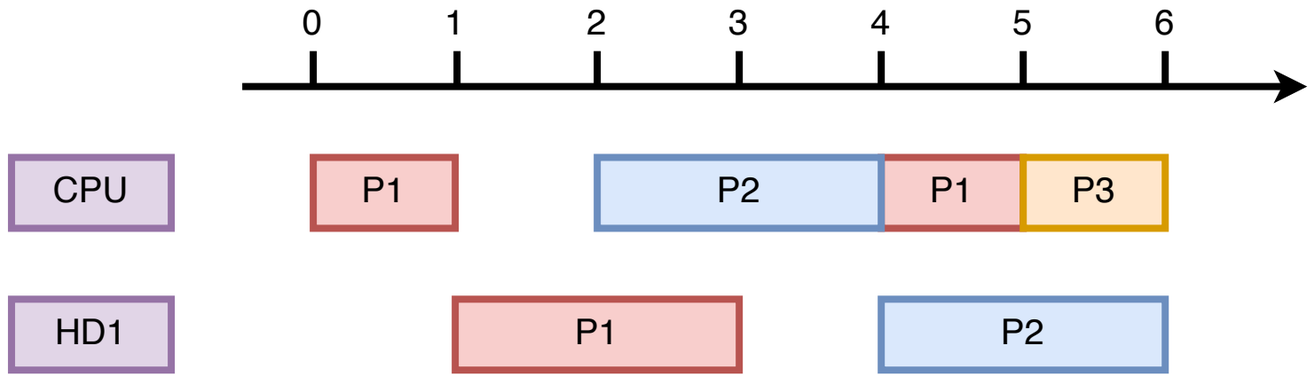
OBJECTIFS POSSIBLES DE L'ORDONNANCEMENT

- 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**) lier à la parallélisation ;
- assurer une **utilisation maximale** des ressources ;
- gérer convenablement les **priorités**.

Objectif $\longrightarrow$ choisir un **algorithme d'ordonnancement** qui minimise ou maximise **un critère**.

TAUX D'UTILISATION

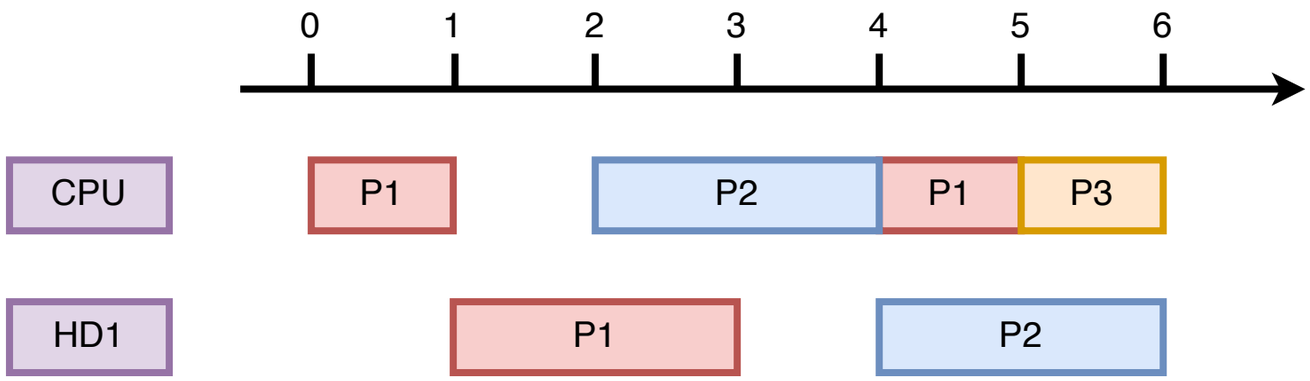
Proportion de temps pendant lequel la ressource (le **CPU**) est utilisée



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

DÉBIT

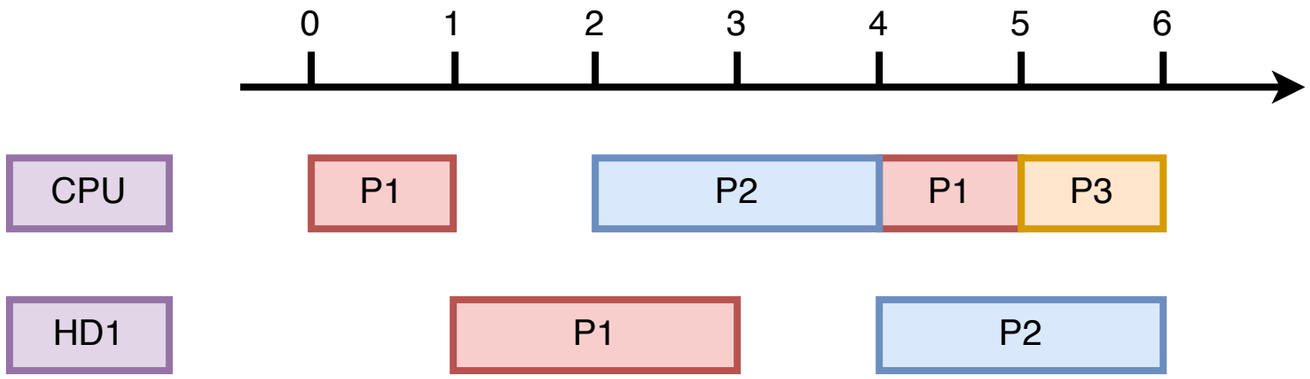
Nombre moyen de processus traités par unité de temps



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

TEMPS D'ATTENTE

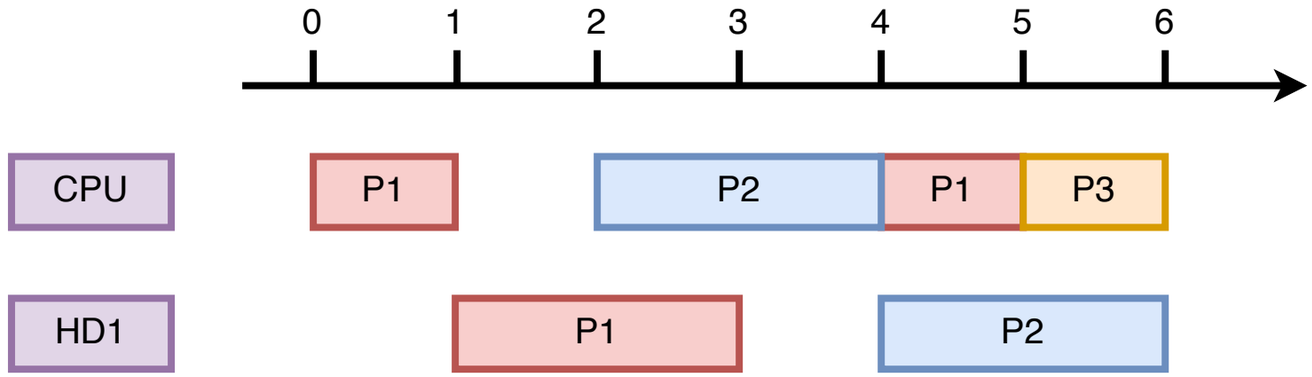
Temps total passé **par tous les processus** dans la file **prêt**



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

ROTATION

Durée d'un processus: $(date\_terminaison - date\_creation)$
$\rightarrow$ **temps de réponse** du processus



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

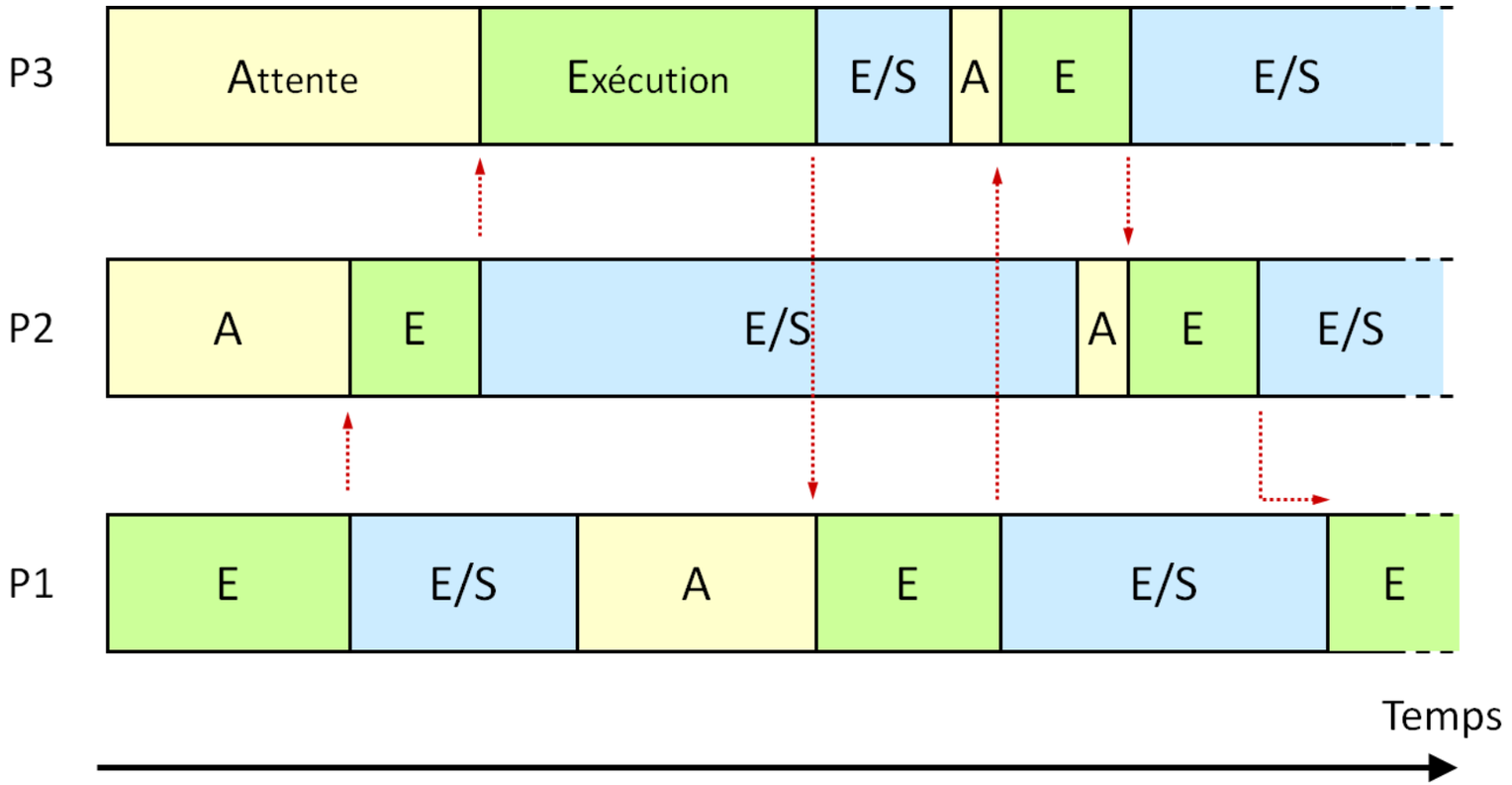
ORDONNANCEMENT

PRÉEMPTIF vs NON PRÉEMPTIF

- Ordonnancement **non préemptif** → après avoir donné le contrôle à un processus, l'OS ne peut pas l'interrompre
  - sauf si en attente d'une ressource
- Ordonnancement **préemptif** (**avec réquisition**) → l'**OS** peut interrompre un processus si :
  - le **quantum** (le temps d'utilisation maximum consécutif) est atteint
  - **un processus plus prioritaire** demande d'utiliser le processeur
- **L'ordonnancement préemptif** est indispensable pour gérer des **systèmes temps réel** ou des **systèmes interactifs**.

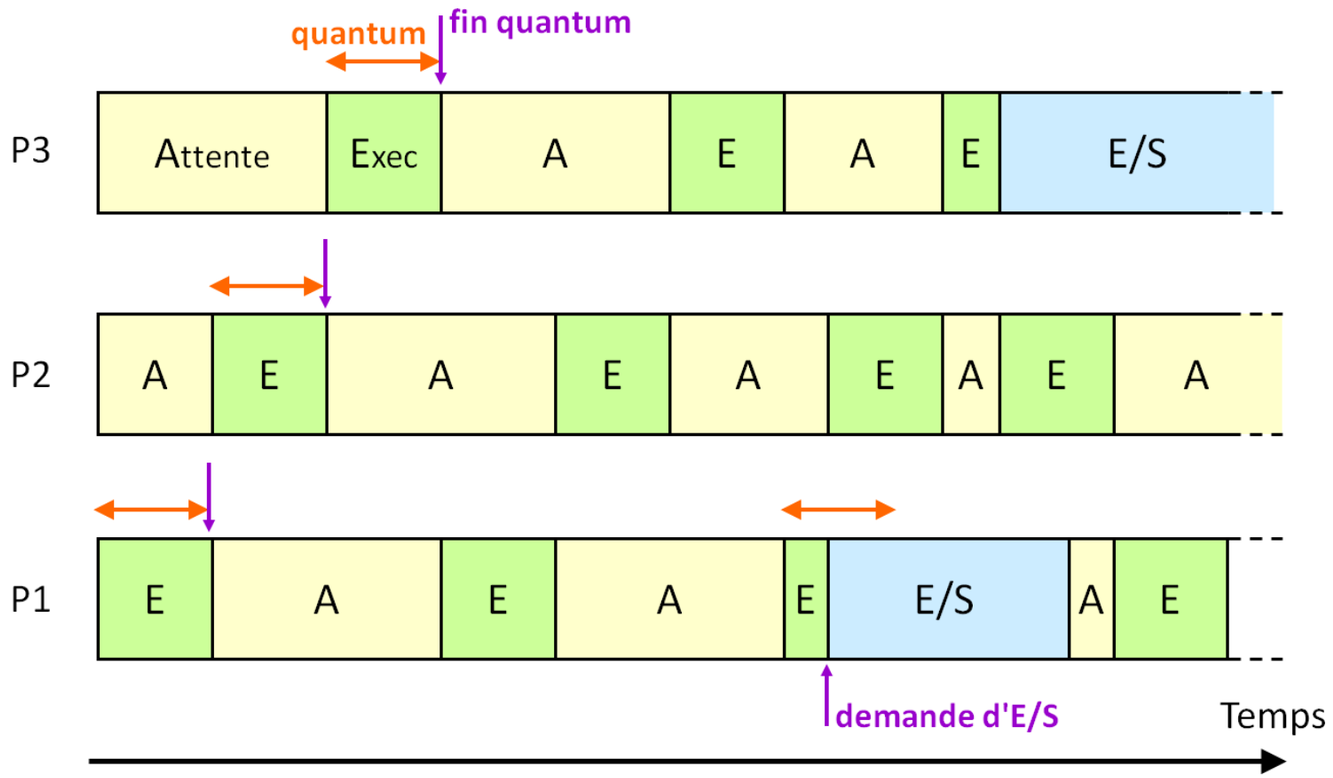
ORDONNANCEMENT

NON PRÉEMPTIF



ORDONNANCEMENT

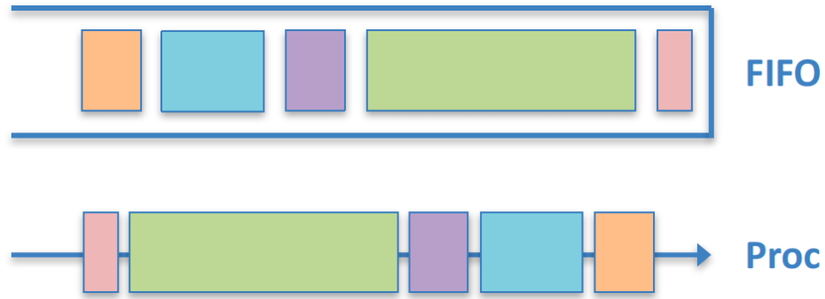
PRÉEMPTIF



DE NOMBREUSES STRATÉGIES

FIFO SANS PRÉEMPTION

- **Principe**:
  - premier arrivé, premier servi.



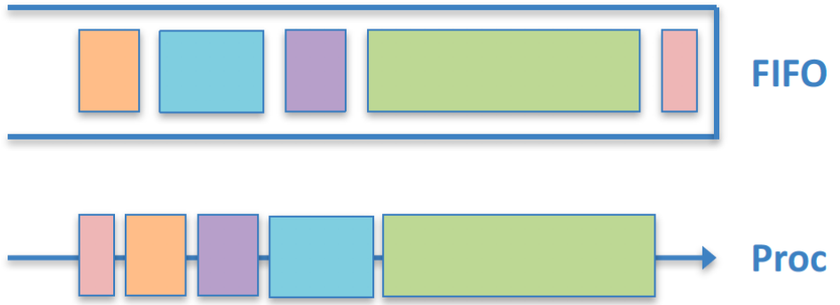
- **Avantages** :
  - ✔ Simple à implémenter
  - ✔ Équitable dans l'ordre d'arrivée

- **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

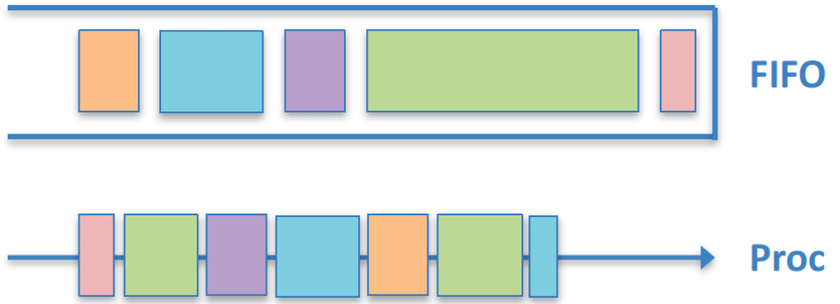
DE NOMBREUSES STRATÉGIES

ROUND-ROBIN (FIFO PRÉEMPTIF)

- **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 $\longrightarrow$ surcoût!



DE NOMBREUSES STRATÉGIES

FILES DE PRIORITÉS

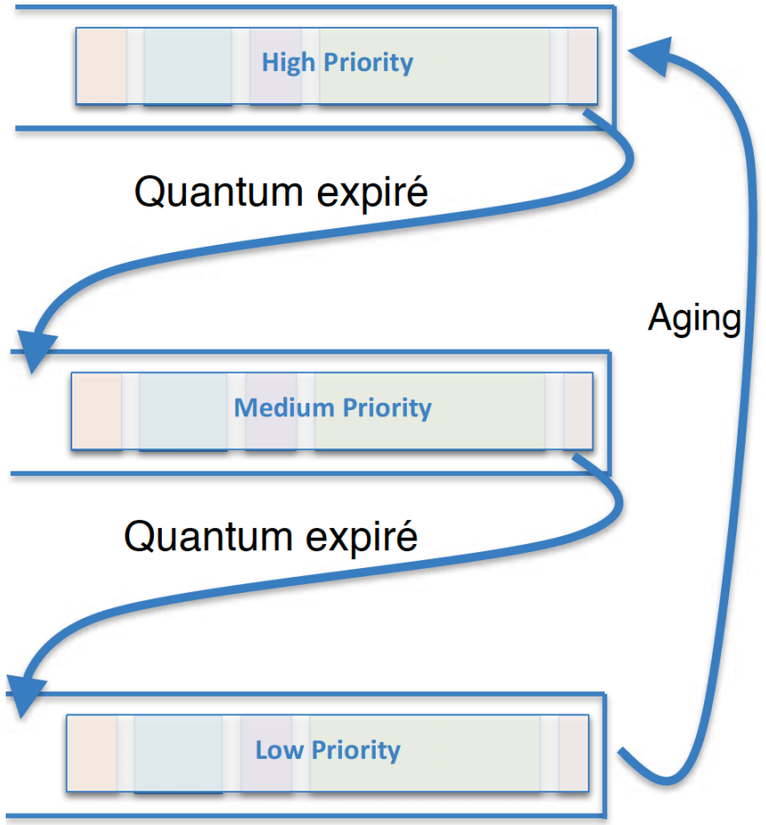
- Plusieurs files d'attente
- La durée des quantums peut dépendre de la file

✔ 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 modifié par l'OS
  - par exemple sortie d'attente I/O
  - attente longue en file basse priorité
- **Utilité** : Windows NT, OS 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 $\longrightarrow$ grand quantum
  - quantum consommé $\longrightarrow$ priorité augmentée
  - E/S $\longrightarrow$ 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 $\longrightarrow$ interface graphique plus réactive!

EXEMPLE : LINUX, MACOS X

Principe

- 2 algorithmes:
  1. Tâches **temps réel** $\longrightarrow$ préemptif selon priorité, **FIFO** ou **RR** par priorité
  2. Autres tâches $\longrightarrow$ 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
    $\longrightarrow credit' = credit/2 + credit_{init}$

PLAN

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