Système d'Exploitation

Cycle de vie des processus
Ordonnanceur

Juan Angel Lorenzo del Castillo
jlo@cy-tech.fr

Contributions de :
Thierry Garcia
Florent Devin

ING1 Informatique - Mathématique appliquée
2022-2023

Plan

Cycle de vie des processus

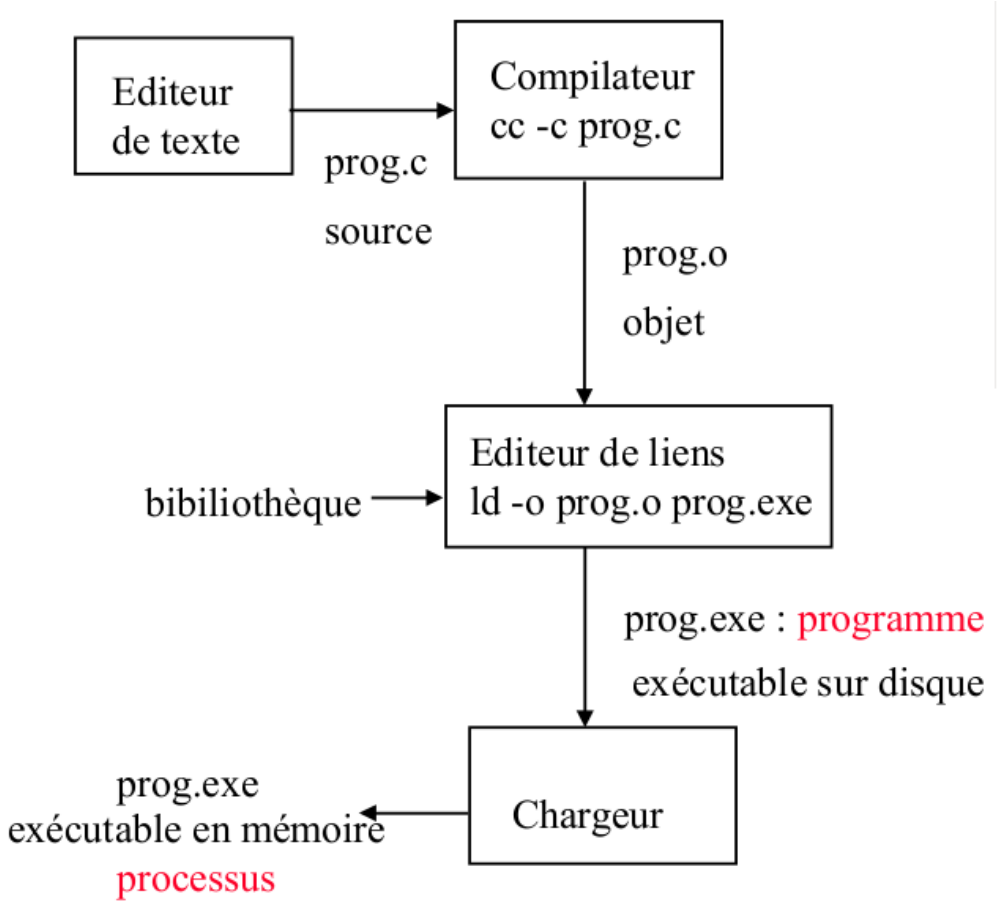
Rappel : Processus

- Un processus est une instance de programme en cours d'exécution.
- Il possède un certain nombre de propriétés et ressources :
  - un espace d'adressage privé (adresses virtuelles)
  - un numéro d'identification : le pid (*process identifier*)
  - un numéro de propriétaire (propriétaire de l'exécutable)
  - un numéro d'utilisateur
  - une table de descripteur des fichiers
  - le pid du processus parent (père)
- **Objectifs** des processus :
  - Séparer les différentes tâches du système
  - Gestion des fichiers et des applications
  - Permettre le multitâche (plusieurs activités "en même temps")
  - Permettre à plusieurs utilisateurs de travailler sur la même machine et donner l'illusion à l'utilisateur d'avoir la machine pour lui tout seul.

Chaque processus doit tenir comptes des autres.

Rappel : Processus

- Un processus est un programme en cours d'exécution auquel est associé un environnement processeur (PSW, registres géneraux, etc.) et un environnement mémoire appelés **contexte du processus**.
- Un processus est l'instance dynamique d'un programme et incarne le fil d'exécution de celui-ci.
- Un processus évolue dans un espace d'adressage protégé



Composantes d'un processus

- Des données (variables globales, etc.) stockées dans une zone de la mémoire qui a été allouée au processus;
- La valeur des registres du processeur lors de l'exécution ;
- Les ressources qui lui ont été allouées par le système d'exploitation (mémoire principale, fichiers ouverts, périphériques utilisés, etc.).

L'ensemble de ces composantes forme le **contexte d'exécution** d'un processus.

Caractéristiques des processus

- Un identificateur ou numéro (PID, *Process IDentification*, en UNIX)
- Un état opérationnel : activ/endormi/prêt, en mémoire central / en swap, mode système /utilisateur.
- Son contexte
- Des informations comme les priorités, la date de démarrage, la filiation
- Des statistiques calculées par le système telles que le temps d'exécution cumulé, le nombre d'opérations d'E/S, etc.

Le **PCB (*Process Control Block*)**, qui représente chaque processus dans le système, contient ces informations :

Identificateur du processus
Etat opérationnel du processus
Contexte
???
Mémoire, priorités, date de démarrage, filiation
Informations comptabilisation, E/S, fichiers ouverts

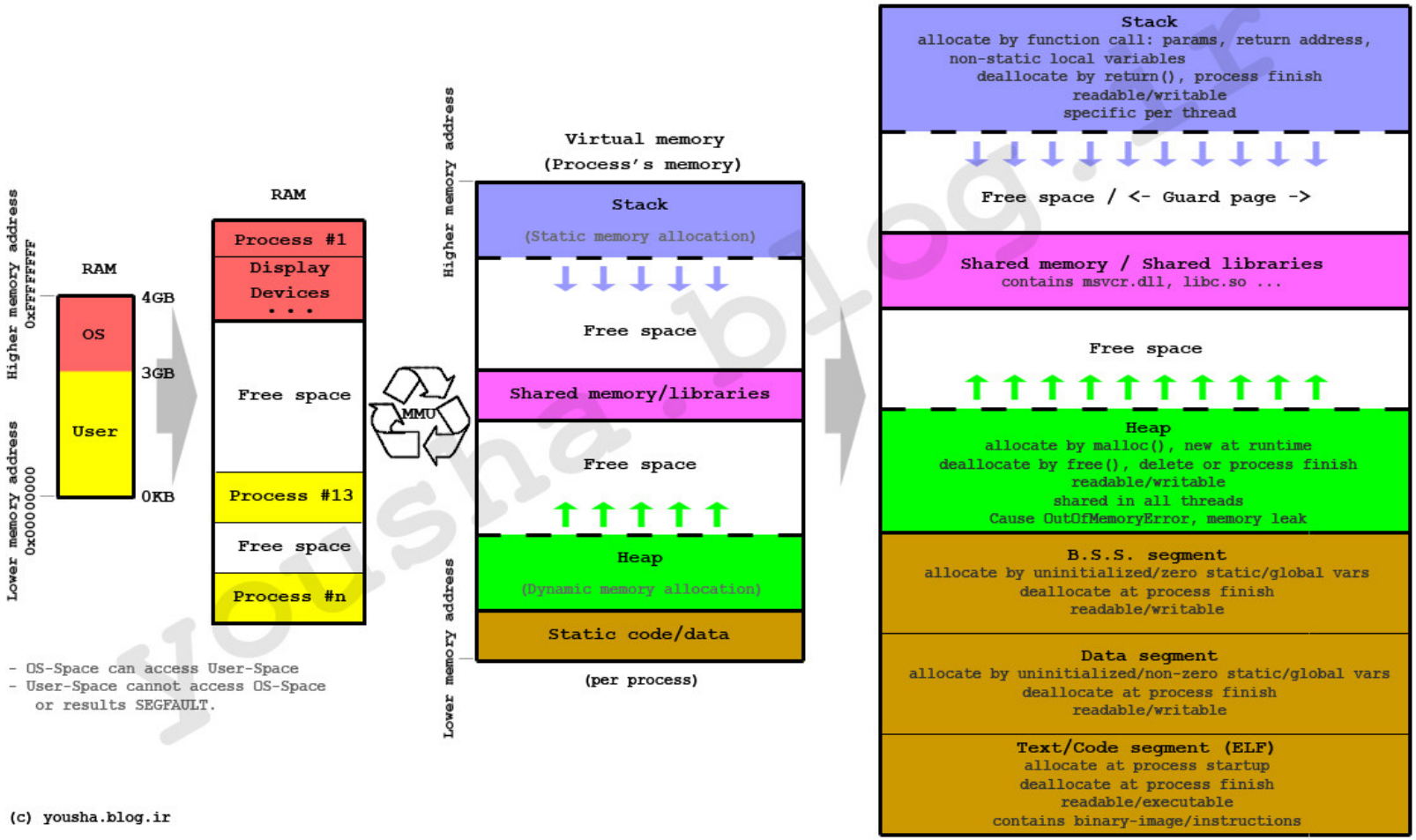
Structure d'un processus

Format d'un fichier exécutable :

- En-tête pour décrire l'ensemble du fichier (attributs, etc.)
- La taille à allouer pour les variables non initialisées.
- Section TEXT qui contient le code à exécuter (en langage machine).
- Section DATA qui contient les données initialisées (en langage machine).
- D'autres sections.

Structure d'un processus

Chargement en mémoire d'un exécutable (UNIX) :



Source : stackoverflow.com/questions/79923/what-and-where-are-the-stack-and-heap

Structure d'un processus

Chargement en mémoire d'un exécutable (UNIX) :

- 4 régions sont allouées en mémoire :
  - Région du **code** (***text*** *segment*) : pour la section TEXT. Taille fixée.
  - Région des **données** (***data*** *segment*) : pour la section DATA. Taille fixée.
    - ★ Section `.data` : variables globales ou statiques initialisées.
    - ★ Section `.bss` (*Block Started by Symbol*) : données qui ne sont pas initialisées.
  - La **Pile** (***Stack***) :
    - ★ Croissance **automatique** dans le sens décroissant des adresses.
    - ★ Ses éléments sont empilés et dépilés (croissance ou décroissance du Stack) lors de l'appel ou retour de fonction.
    - ★ Taille limitée. Variables **locales** et de taille fixée.
    - ★ Pointeur de pile pour indiquer la profondeur courante de la pile.
    - ★ Un processus UNIX pouvant s'exécuter en mode noyau et/ou utilisateur, une pile privée sera utilisée dans chaque mode.
  - Le **Tas** (***Heap***) :
    - ★ Allocation **manuelle** par l'utilisateur : `malloc()`, `calloc()` mais aussi `free()` sont nécessaires.
    - ★ Taille non limitée. Variables accessibles globalement avec une taille qui peut varier (`realloc()`).

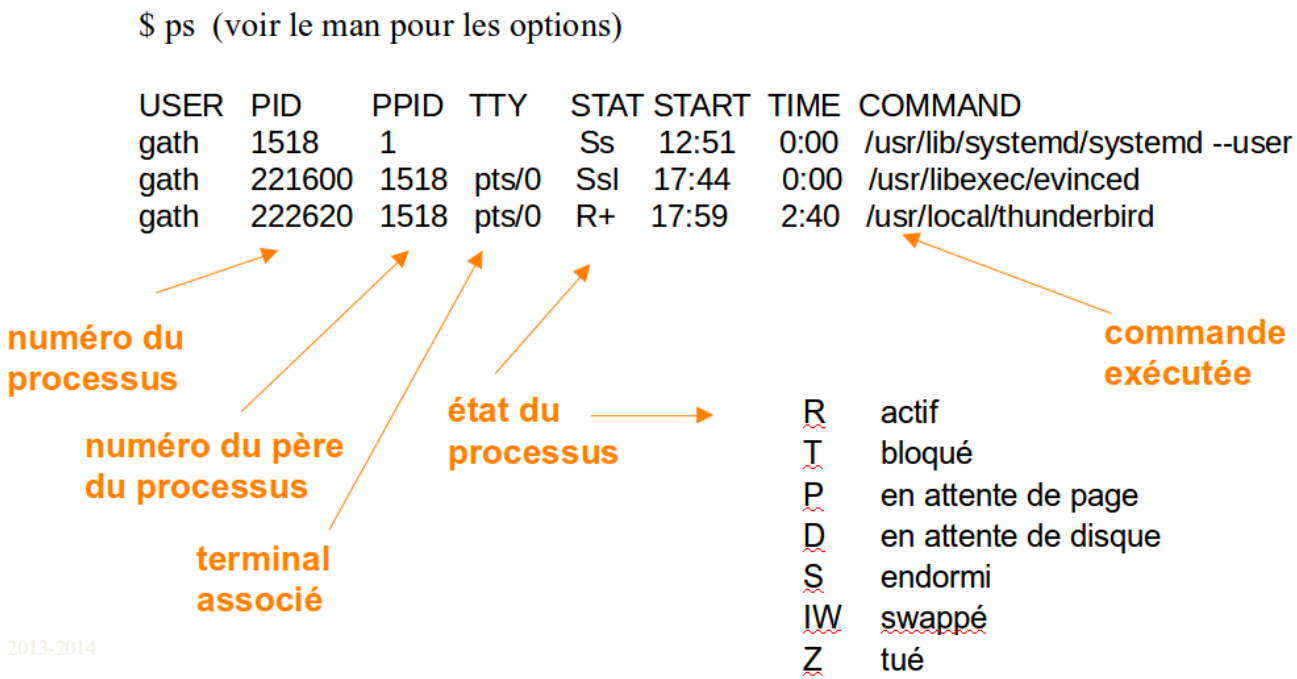
Plus d'information : https://en.wikipedia.org/wiki/Data_segment

Contexte d'un processus

- Environnement matériel
  - compteur ordinal
  - pointeur de pile
  - registres de travail
  - registres de données
  - registres pour la mémoire virtuelle
  - ...

Contexte d'un processus

- Environnement système
  - Table des processus :
    - ★ Le SE maintient une table des processus où chaque processus a une entrée unique : le PID (process ID).
    - ★ Interne au noyau.
    - ★ Pour afficher la table des processus, on peut utiliser dans un terminal les commandes `ps`, `pstree`, `top` :



Contexte d'un processus

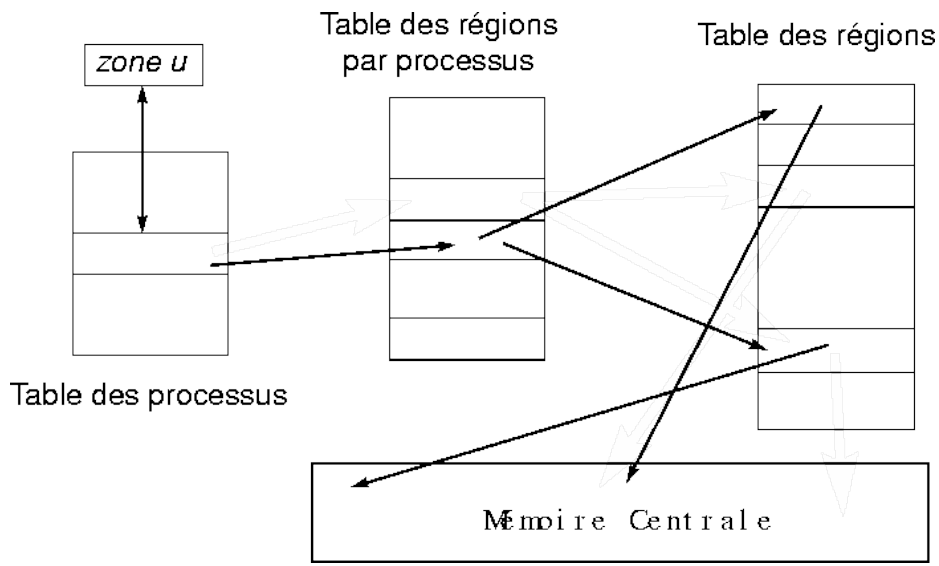
- Environnement système
  - Table des processus :
    - ★ Sortie de la commande 'ps l'

```
[15:02:12] jlo@Multivac> ~:$ ps l
F   UID   PID  PPID PRI  NI    VSZ   RSS WCHAN  STAT TTY        TIME COMMAND
0  1000   502     1  20   0 450316  4372 poll_s Sl+  tty2       0:01 /usr/bin/python3 /usr/share/ap
4  1000  2758  2698  20   0 193448  2128 poll_s Ssl+ tty2       0:00 /usr/lib/gdm3/gdm-x-session --
4  1000  2760  2758  20   0 1321472 253528 poll_s Sl+ tty2    119:44 /usr/lib/xorg/Xorg vt2 -displa
0  1000  2772  2758  20   0  44168  3672 ep_pol S+   tty2       0:11 dbus-daemon --print-address 4
0  1000  2775  2758  20   0 566828  3076 poll_s Sl+  tty2       0:02 /usr/lib/gnome-session/gnome-s
0  1000  2871     1  20   0 274584  1932 poll_s Sl+  tty2       0:00 /usr/lib/gvfs/gvfsd
```

- F : localisation du processus
  - 0 : Hors de la mémoire centrale
  - 1 : En mémoire
  - 2 : Processus système
  - 4 : Verrouillé (en attente d'E/S)
  - 8 : Vidage
- `UID` : User IDentifier
- `PID` : Process IDentifier
- `PPID` : Parent Process IDentifier
- `PRI` : Priorité du processus. Une valeur plus élevée veut dire une priorité plus basse.
- `NI` : Valeur de Nice
- `VSZ` : Taille du processus
- `RSS` : Mémoire vive utilisée par le processus (en kiloBytes)
- `WCHAN` : Attente du processus
- `STAT` : État du processus
  - O : non existant
  - S : sleeping
  - W : waiting
  - R : running
  - Z : zombie
  - X : en évolution
  - P : pause
  - I : attente d'entrée au terminal ou input
  - L : attente d'un fichier (verrouillé ou locked)
- `TTY` : Terminal associé
- `TIME` : Temps d'exécution cumulé
- `COMMAND` : Commande lancée

Contexte d'un processus

- Environnement système
  - Table des processus
  - Structure U (*U Area*) :
    - ★ Structure allouée par le noyau pour chaque processus.
    - ★ Privée au processus et uniquement manipulable par le noyau.
    - ★ Contient information spécifique du processus (fichiers ouverts, répertoire courant, etc.)
    - ★ Contient l'adresse de la *Table des Régions par processus* qui permet d'accéder à la *Table des Régions*. Ce double niveau d'indirection permet de faire partager des régions.

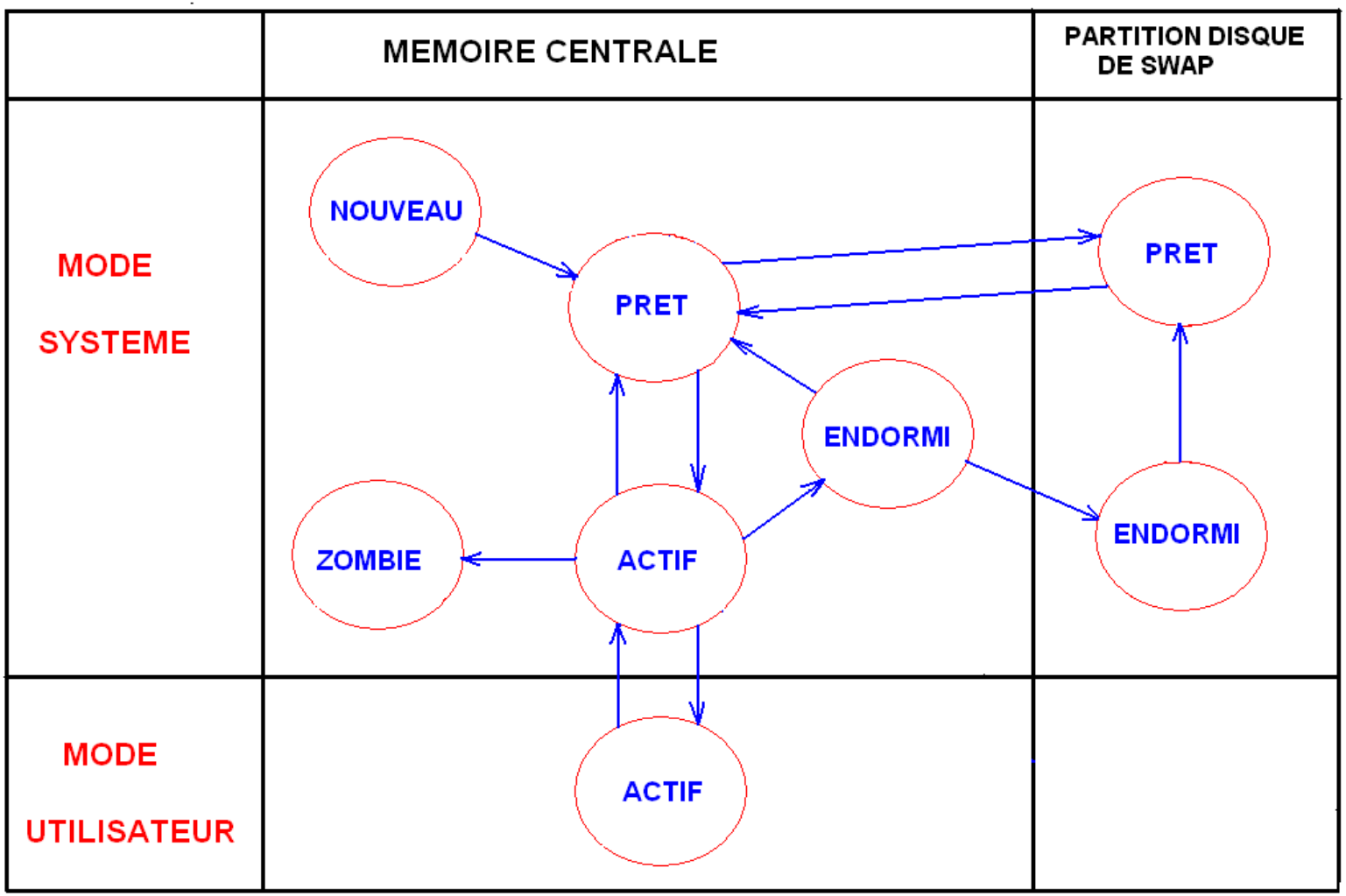


Source : http://www-igm.univ-mlv.fr/~dr/cs/node67.html

Cycle de vie d'un processus

- Quand un processus s'exécute, il change d'état.
- Le processeur va basculer constamment d'un processus à l'autre.
- Un seul processus peut être en exécution sur n'importe quel processeur à tout moment.

Cycle de vie d'un processus



État d'un processus sous Unix

1. **Nouveau:** création. Ni prêt, ni endormi; état initial de tous les processus.
2. **Prêt** (*waiting*) **:** mis en attente jusqu'à que la CPU soit libérée.
   - Endormi en mémoire centrale ou
   - Endormi en zone de swap (sur disque par exemple)
3. **Actif** (*running*) **:**
   - Si le processus épuise le temps qui lui est alloué par l'ordonnanceur, il est remis en file d'attente des prêts.
   - S'il a besoin d'une ressource non disponible (op. E/S, par exemple) il est mis en état bloqué.
   - S'il se termine, il passe à l'état Zombie.
4. **Endormi :** Bloqué. Le processus est en attente d'une ressource. Dès sa libération il repasse à l'état Prêt.
   - Endormi en mémoire centrale ou en zone de swap
5. **Terminé** (*Zombie*) **:** réalisation d'un exit.
   - Il est conservé uniquement dans la table des processus le temps nécessaire pour que son processus père puisse récupérer le code de retour et d'autres informations de gestion (coût de l'exécution sous forme de temps, et d'utilisation des ressources).
   - Si un processus est orphelin de père, c'est le PID 1 qui l'accueillera.

L'espace d'échange (swap)

- Utilisé lorsque la mémoire physique (RAM) arrive à saturation.
- Situé sur le disque dur (donc temps d'accès **plus lent** que la RAM).
- Il peut être une partition de swap consacrée, un fichier de swap ou une combinaison des deux.

Gestion du swap

- Géré par le processus 0
- Transfert **hors** de la mémoire
  - Transfert en priorité les processus bloqués, puis les prêts
  - Aucun transfert de processus Zombie, ni verrouillé par le SE
  - Transfert d'un processus prêt : uniquement si il est présent en mémoire depuis un certain temps.
  - Si pas de possibilité : "re-scan" toutes les secondes
- Transfert **vers** la mémoire
  - Examen des différents processus prêt présents sur le disque
  - Choix du plus ancien
  - Transfert de celui-ci : allocation de mémoire, lecture depuis le disque, libération de l'espace utilisé en swap
  - Exécution jusqu'à
    - ★ Plus de processus prêt sur le disque
    - ★ Plus de place en mémoire

Exécution interactive d'un programme

Exécution en avant plan :

1. Lecture de la commande (ex : `./bidon`)
2. Duplication du shell via la commande `fork`. Existence de deux shells
3. Recouvrement du segment de texte (programme), par le code approprié
4. Récupération du numéro du processus fils par le shell initial
5. Attente de la fin d'exécution du processus fils (`wait (etat)`)

Exécution interactive d'un programme

Exécution en arrière plan :

1. Lecture de la commande (ex : `./bidon`)
2. Duplication du shell via la commande `fork`. Existence de deux shells
3. Recouvrement du segment de texte (programme), par le code approprié
4. Récupération du numéro du processus fils par le shell initial
5. Émission d'un prompt, puis reprise de l'activité

Plan

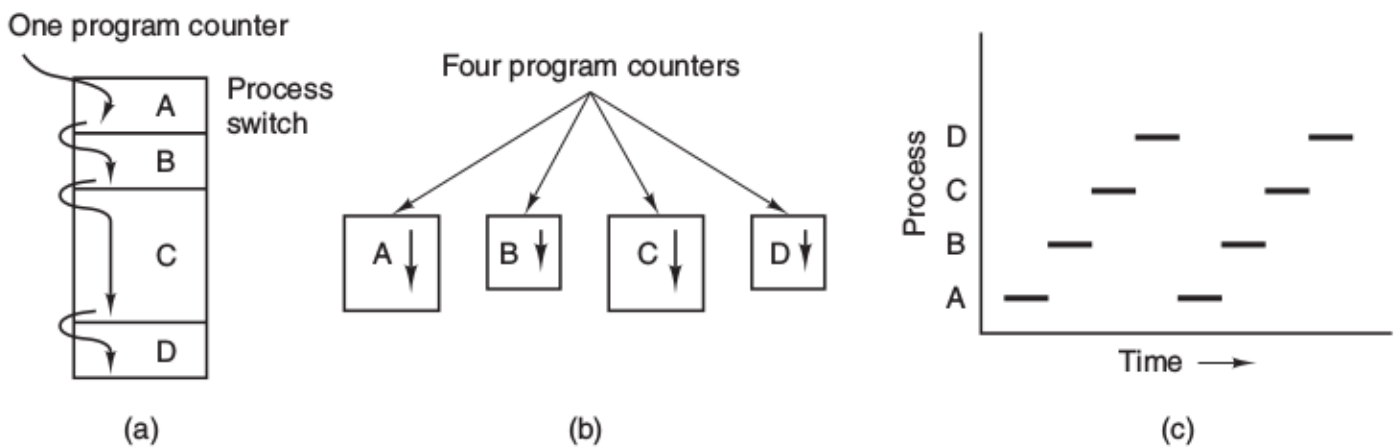
Ordonnanceur

Problématique

- Rappel : Le processeur (CPU) est chargé d'exécuter les instructions des programmes **placés en mémoire centrale**.
- Considérons qu'un processeur ne peut exécuter qu'une seule instruction à la fois.
- Nous sommes dans un système multitâche = plusieurs processus s'exécutent simultanément avec un processeur ne pouvant exécuter qu'une instruction (d'un programme) à la fois.
- Comment exécuter plusieurs programmes en même temps et travailler à plusieurs utilisateurs sur la même machine afin que tout le monde ait l'impression d'avoir la machine à lui tout seul ?

Simultanéités

- Activation de plusieurs processus en même temps : **Multiprogrammation**
- *Simultanéité totale ou vraie* : nombre de processus $\leq$ nombre de processeurs
- *Simultanéité partielle* : sinon
  - Exécution enchevêtrée de plusieurs processus sur un seul processeur.
  - Obtenue par commutation temporelle d'un processus à l'autre sur le processeur : **pseudo-parallélisme**



(a) Multiprogramming 4 processus
(b) Modèle conceptuel de 4 processus séquentiels.
(c) Simultanéité partielle : un seul processus actif en même temps.

Source : Tanenbaum

Mécanismes de commutation

- La **commutation de processus** va sauvegarder le contexte d'un processus pour restaurer/reprendre (à la place) le contexte d'un processus précédemment interrompu dans le cadre d'un partage des ressources d'un processeur.
  1. P1 s'exécute, on stoppe P1 et on mémorise le contexte de P1
  2. P2 s'exécute, on stoppe P2 et on mémorise le contexte de P2
  3. On restaure le contexte de P1 et on refait 1) à partir de l'endroit où P1 a été stoppé
  4. On restaure le contexte de P2 et on refait 2) à partir de l'endroit où P2 a été stoppé ...
- Qui effectue la commutation ? Le système d'exploitation
  - il stocke l'état du processus en RAM
  - il choisit le nouveau processus
  - il récupère son état et le relance
- Mais comment décider quel nouveau processus exécuter ?

Ordonnancement

- L'*ordonnancement* des processus va permettre de planifier l'exécution des processus.
- Dans le système d'exploitation il existe un ordonnanceur qui a pour rôle de choisir, parmi tous les processus existants, lequel va pouvoir s'exécuter en fonction d'une politique d' ordonnancement.
- Pour savoir quel est l'état des processus, s'ils existent dans le système, le système d'exploitation utilise les informations du processus sur le PCB (*Process Control Block*).

Critères d'ordonnancement

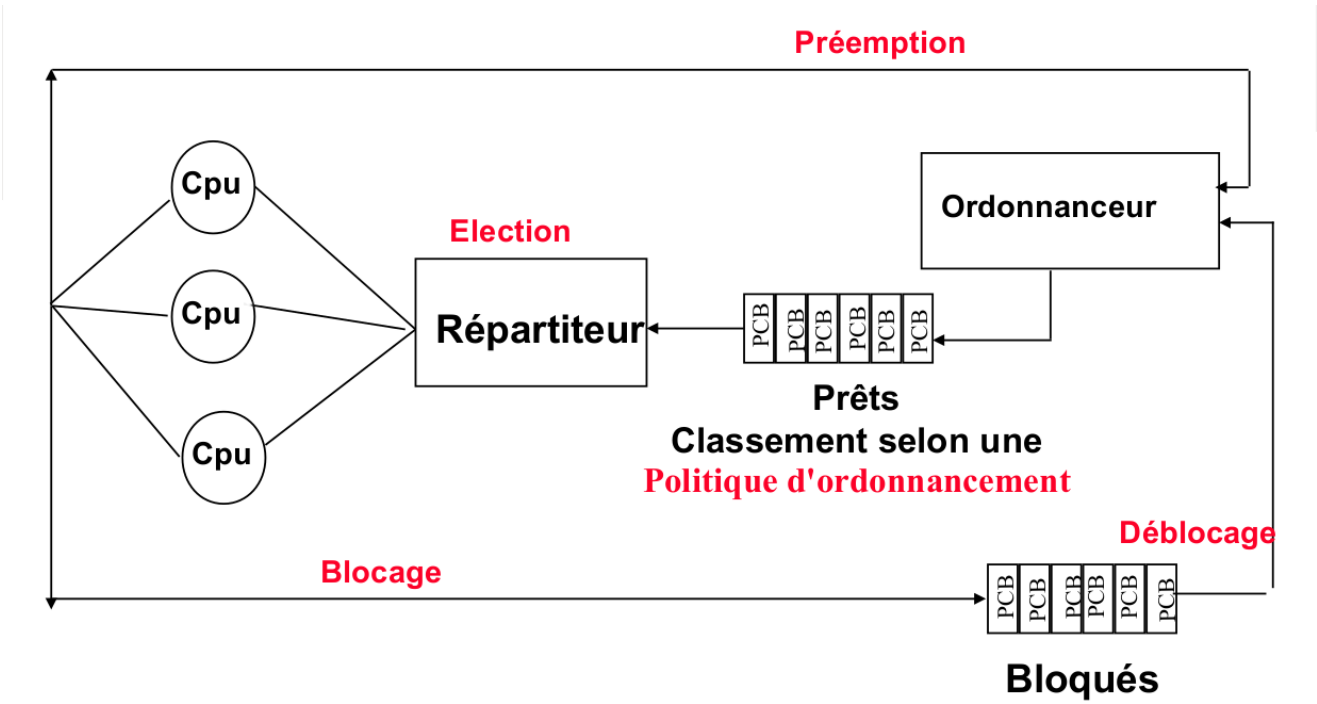
- *Équité* : chaque processus doit pouvoir s'exécuter
- *Efficacité* : utilisation des processeurs maximales
- *Temps de réponse* : minimiser les temps de réponses pour les processus interactifs
- *Temps d'exécution* : minimiser le temps d'exécution
- *Rendements* : Nombre de travaux réalisés par unités de temps doit être maximal

Problèmes

- Favorise une catégorie : défavorise une autre
- Comment connaître à l'avance les demandes de l'entrée/sortie ?
- Nécessité de mettre en œuvre un mécanisme pour "reprendre la main"
- Obligation d'effectuer un ordonanncement avec réquisition : *Préemption*
- Possibilité de conflits : utilisation d'un mécanisme comme les sémaphores

Familles d'ordonnanceurs

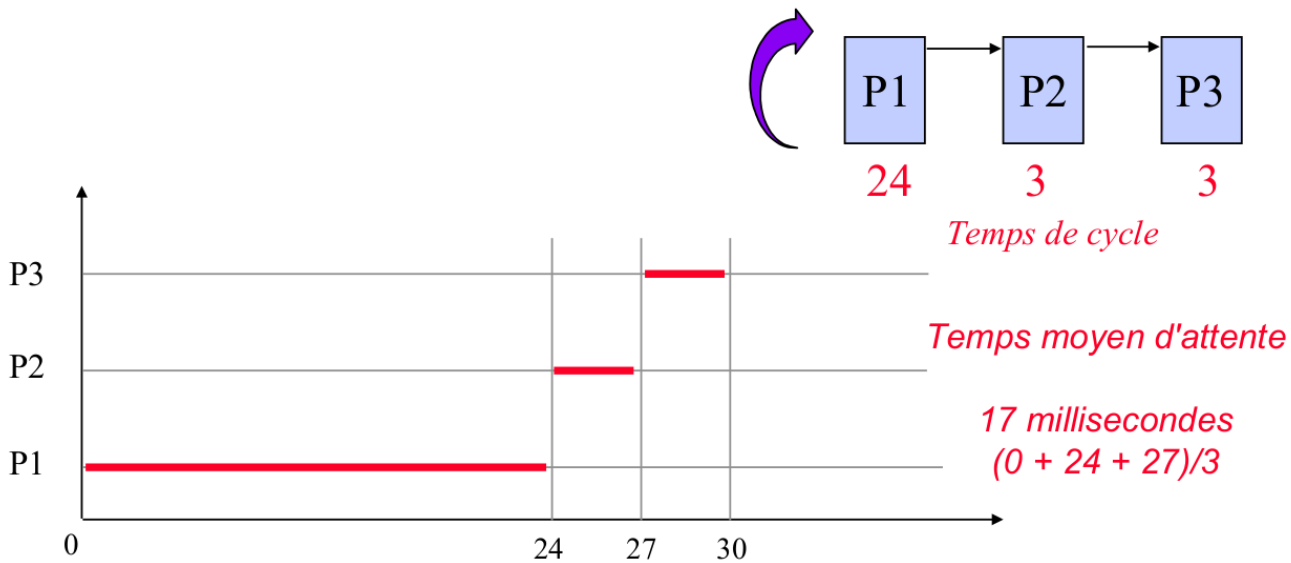
- **Non préemptif** ou **sans réquisition** (OSR) : l'algorithme du SE choisit un nouveau processus sur blocage ou terminaison du processus courant.
- **Préemptif** ou **avec réquisition** (OAR) : Les ordinateurs ont une horloge électronique qui génère périodiquement une interruption. À chaque interruption (après un temps fixé, ou **quantum**) l'ordonnanceur reprend la main et élit un nouveau processus actif.



OSR : Ordonnanceur Sans Réquisition

Politiques d'ordonnancement :

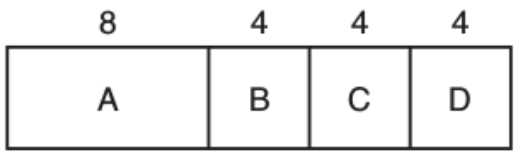
- **FCFS (*First Come First Served*) ou FIFO (First In First Out) :** Ordre d'arrivée en gérant une file des processus.
  - Premier processus de la queue commence à s'exécuter jusqu'à ce qu'il se termine ou qu'il se bloque. En retournant du blocage, il est mis à la fin de la file $\rightarrow$ **Tourniquet** ou **Round Robin**.
  - Algorithme facile à implanter.
  - Loin d'optimiser le temps de traitement moyen pour des processus avec temps d'exécution très différents.



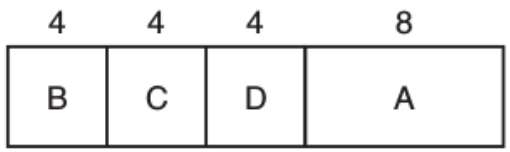
OSR : Ordonnanceur Sans Réquisition

Politiques d'ordonnancement :

- **PCTE (*Plus Court Temps d'exécution*)** ou **SJF (*Shortest Job First*) :** inverse du temps d'exécution :
  - Plusieurs travaux d'égale importance se trouvent dans une file
  - Élection du plus court d'abord : il faut connaître leurs temps d'exécution à l'avance (bien adapté au traitement par lots)
  - Temps d'attente moyen minimal SI toutes les tâches sont présentes dans la file d'attente au moment où débute l'assignation.
  - Meilleur temps moyen de séjour ($t_{séjour}$: intervalle de temps entre la soumission du processus et son achèvement)



Avant SJF



Après SJF

$t_{séjourA} = 8$
$t_{séjourB} = 8 + 4$
$t_{séjourC} = 8 + 4 + 4$
$t_{séjourD} = 8 + 4 + 4 + 4$
$\bar{t}_{séjour} = \frac{4*8+3*4+2*4+4}{4} = 14$

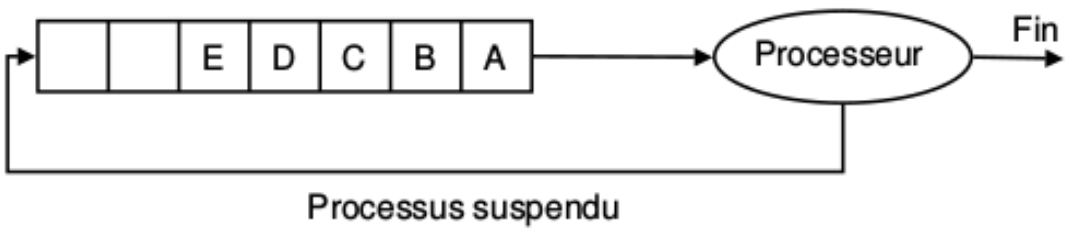
$t_{séjourB} = 4$
$t_{séjourC} = 4 + 4$
$t_{séjourD} = 4 + 4 + 4$
$t_{séjourA} = 4 + 4 + 4 + 8$
$\bar{t}_{séjour} = \frac{4*4+3*4+2*4+8}{4} = 11$

OAR : Ordonnanceur Avec Réquisition

Politiques d'ordonnancement :

Round-Robin ou **Ordonnancement circulaire**:

- Un des plus simples et des plus robustes
- Attribution d'un **quantum** de temps
- Temps d'exécution maximal par processus
- Deux cas :
  - Exécution pas achevée : Processeur réquisitionné par l'ordonnanceur et attribué à un autre processus
  - Exécution terminée ou bloquée : Processeur attribué automatiquement à un autre processus

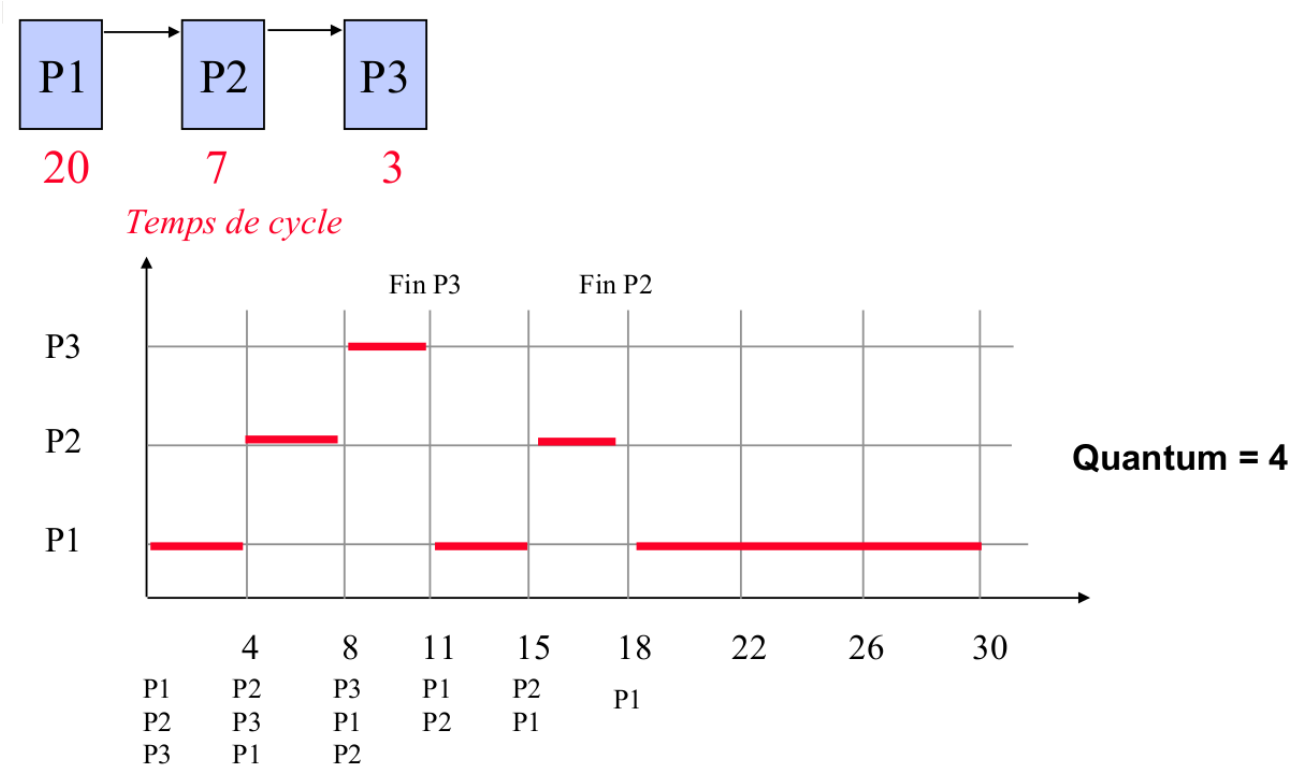


OAR : Ordonnanceur Avec Réquisition

Politiques d'ordonnancement :

Round-Robin ou **Ordonnancement circulaire**:

- *Quantum de temps trop petit* : trop de commutations de processus et abaisse l'efficacité du processeur.
- *Quantum de temps trop grand* : peu d'interactivité, augmente le temps de réponse des commandes courtes.
- Quantum acceptable : entre 20 et 50 ms.



OAR : Ordonnanceur Avec Réquisition

Politiques d'ordonnancement :

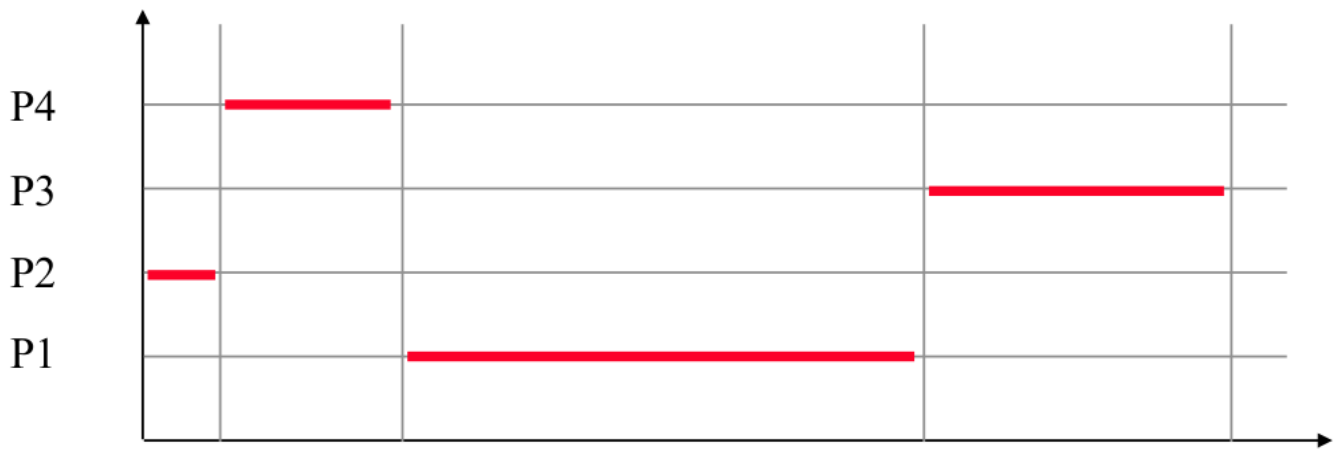
Shortest Remaining Time (SRT) : version préemptive de SJF.

1. Processus arrive dans la file de processus.
2. L'ordonnanceur compare la valeur espérée pour ce processus avec la valeur du processus actuellement en exécution.
3. Si le temps du nouveau processus est plus petit, il rentre en exécution immédiatement.

Ordonnancement par priorités constantes

- Le processus de plus forte priorité est élu.
- La plus petite valeur correspond à la plus forte priorité.

Processus	Temps d'exéc.	Priorité
P1	10	3
P2	1	1
P3	5	4
P4	2	2



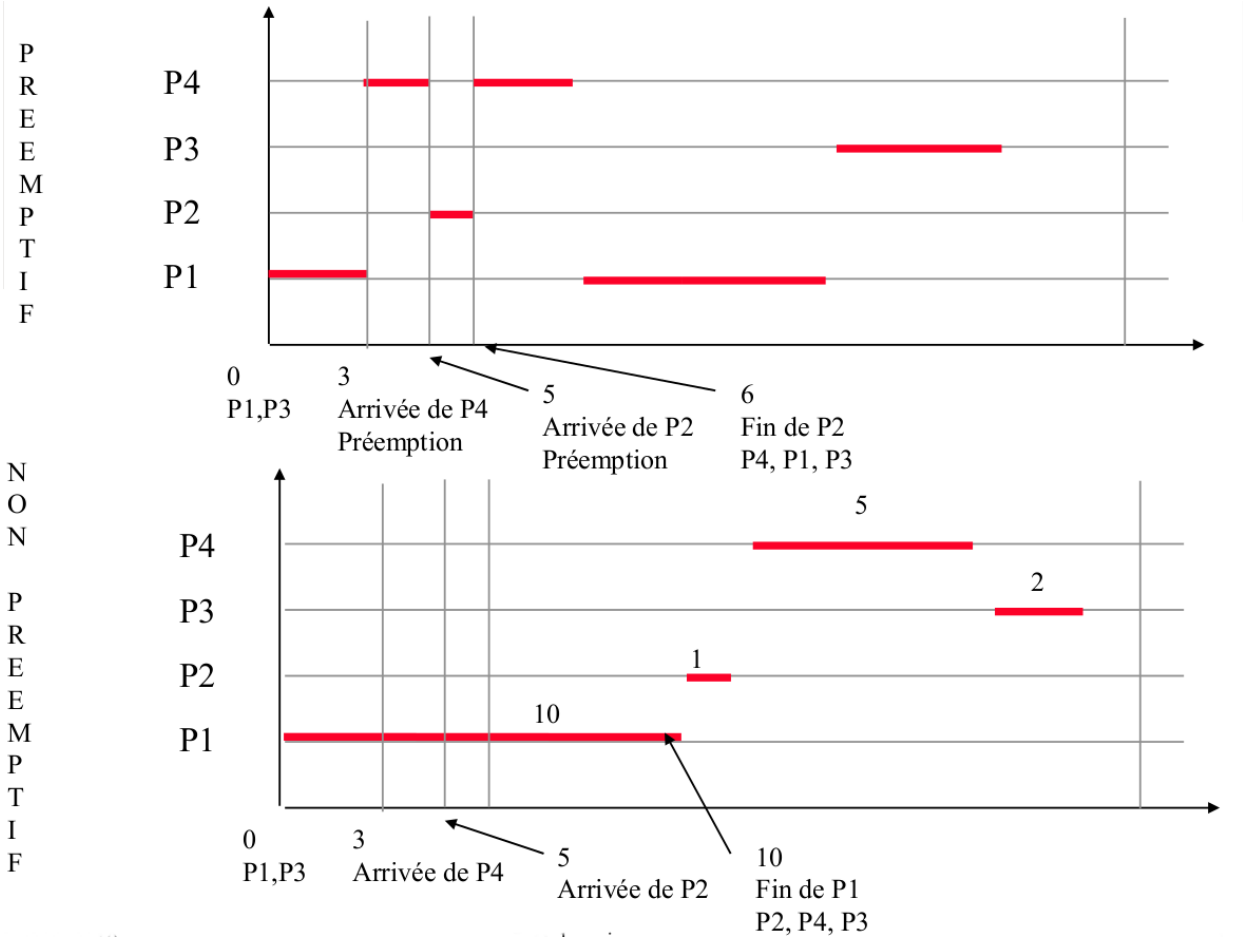
Temps moyen d'attente

4,25 millisecondes

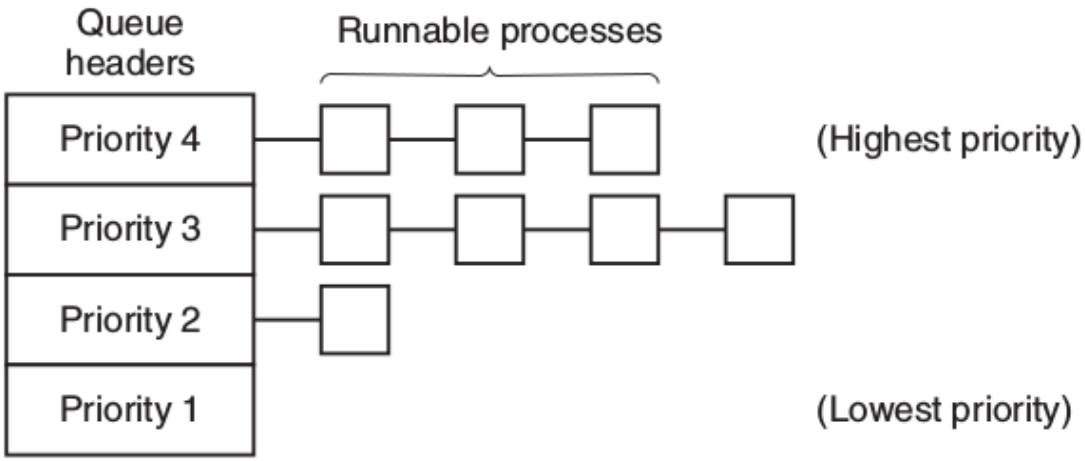
Ordonnancement par priorités constantes

- Avec un temps d'arrivée différent

Processus	Temps d'exéc.	Priorité	Temps d'arrivée
P1	10	3	0
P2	1	1	5
P3	5	4	0
P4	2	2	3



Ordonnancement par priorités constantes



- Les processus sont classifiés en plusieurs classes de priorité.
- Sélection d'un processus, l'ordonnanceur parcourt successivement les queues dans l'ordre décroissant.
- Autre possibilité : partager les quantums de temps sur les différentes queues.
- Autre possibilité : réaliser différents algorithmes d'ordonnancement sur les différentes queues. Risques :
  - Risque de famine pour les priorités faibles
    $\Rightarrow$ Solution : augmenter la priorité avec le temps d'attente
  - Risque de dégradation importante du système pour schedulers non préemptifs $\Rightarrow$ Solution : préemption

Ordonnancement à deux niveaux

- La taille de la mémoire centrale peut être insuffisante pour contenir tous les processus prêts à être exécutés.
- Certains processus doivent résider sur disque.
- Ordonnanceur de bas niveau (*CPU scheduler*) :
  - utilisation de l'un des algorithmes précédents aux processus résidant en mémoire centrale.
- Ordonnanceur de haut niveau (*medium term scheduler*) :
  - retire de la mémoire les processus qui y sont resté assez longtemps.
  - transfère en mémoire des processus résidant sur disque.

Ordonnancement à deux niveaux

- Possibilité d'existence d'un ordonnanceur à long terme(*job scheduler*)
  - détermine si un processus utilisateur peut effectivement entrer dans le système
  - au besoin diffère l'entrée : temps de réponse se dégradent
- Prise en compte par l'ordonnanceur de haut niveau :
  - Temps du processus en mémoire ou sur disque
  - Temps d'utilisation du processeur par le processus
  - Priorité du processus
  - Taille du processus

Simulateur d'ordonnancement sur :
https://solver.assistedcoding.eu/process_scheduling

Cas d'étude : Unix

- Plusieurs file d'attente.
- Une priorité per file.
- Les processus prêts qui sont en mémoire sont répartis dans les files selon leur priorité.
- Les priorités des processus d'utilisateur sont ≥ 0.
- Les priorités des processus du noyau sont < 0 (plus élevées).
- Pour chaque niveau de priorité : système de tourniquet (*Round-Robin*).
- Les priorités des processus prêts en mémoire sont recalculées périodiquement.

Cas d'étude : Unix