Système d'Exploitation

Cycle de vie des processus
Ordonnanceur

Juan Angel Lorenzo del Castillo
Florent Devin

ING1 GI-GM
2020-2021

Plan

Cycle de vie des processus

Rappel : Processus systèmes

- Ne sont pas contrôlés par un terminal
- Propriétaire *root*
- Résident en mémoire centrale en attente de demande
- Assure des services pour tous les utilisateurs
- Peuvent être lancés au démarrage, ou par l'administrateur

Rappel : Création d'un processus

- Création : appel système `int fork ();`
- Crée tous les processus
- Notion de père/fils
- Notion de PID (unique)
- PID 0 : créé au démarrage. Appel à `fork()` pour créer le processus PID 1.
- PID 1 (appelé *init* ou *systemd*): l'ancêtre de tous les autres processus.

Format d'un fichier exécutable

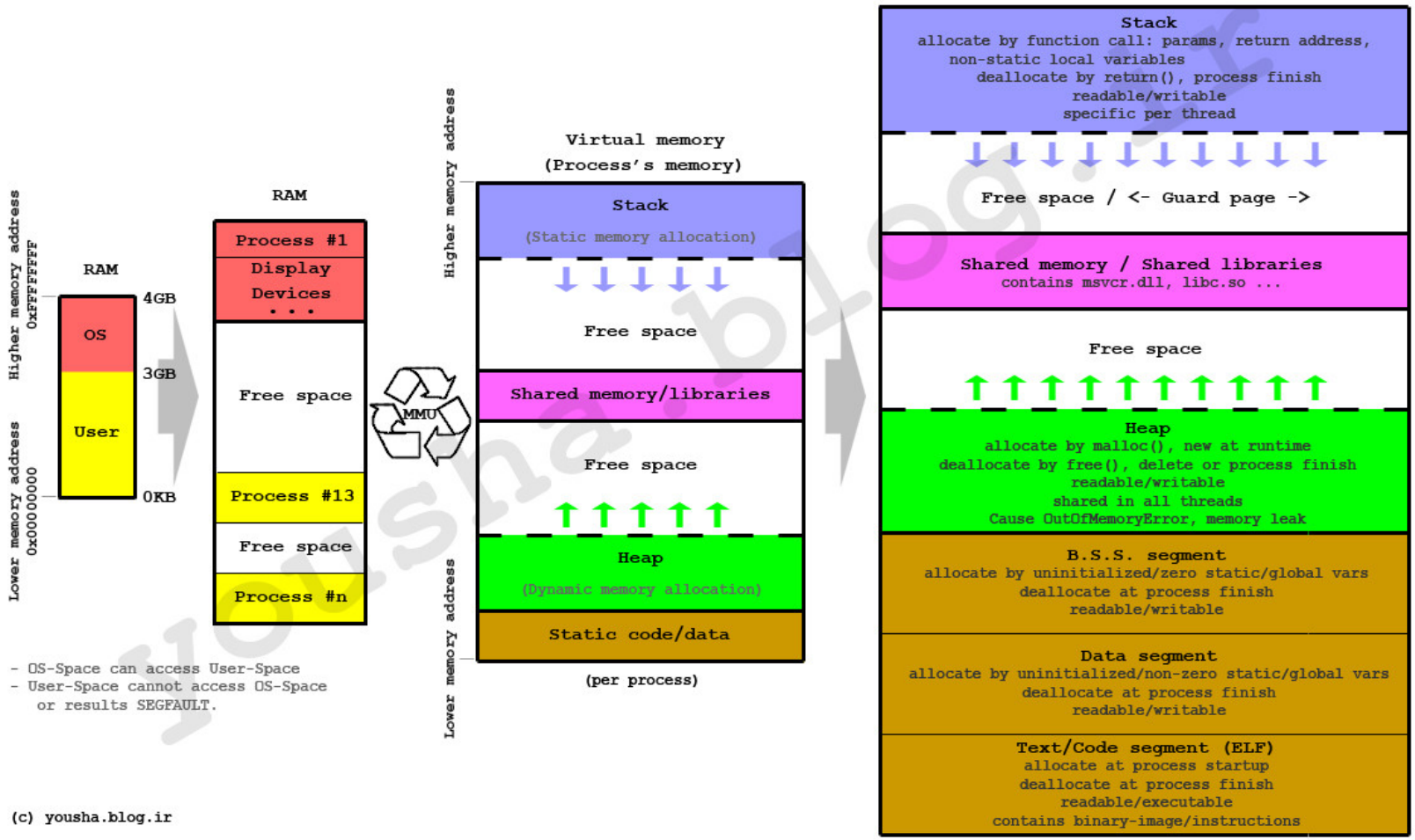
- En-tête pour décrire l'ensemble du fichier (attributs, etc.)
- La taille à allouer pour les variables non initialisées.
- Section TEXT qui contient le code à exécuter (en langage machine).
- Section DATA qui contient les données initialisées (en langage machine).
- D'autres sections.

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

Chargement en mémoire d'un exécutable (UNIX)



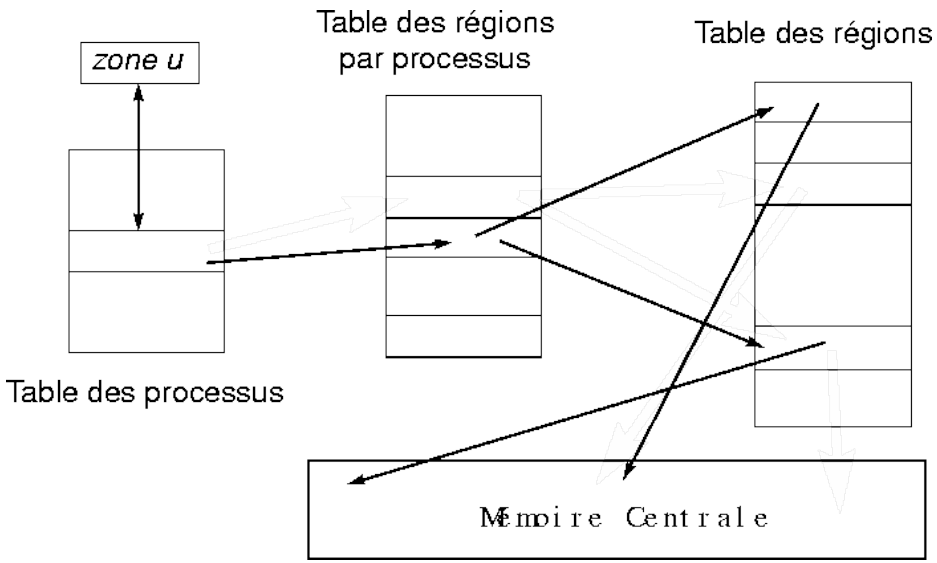
Source : stackoverflow.com/questions/79923/what-and-where-are-the-stack-and-heap

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 : interne au noyau.
  - Zone 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.

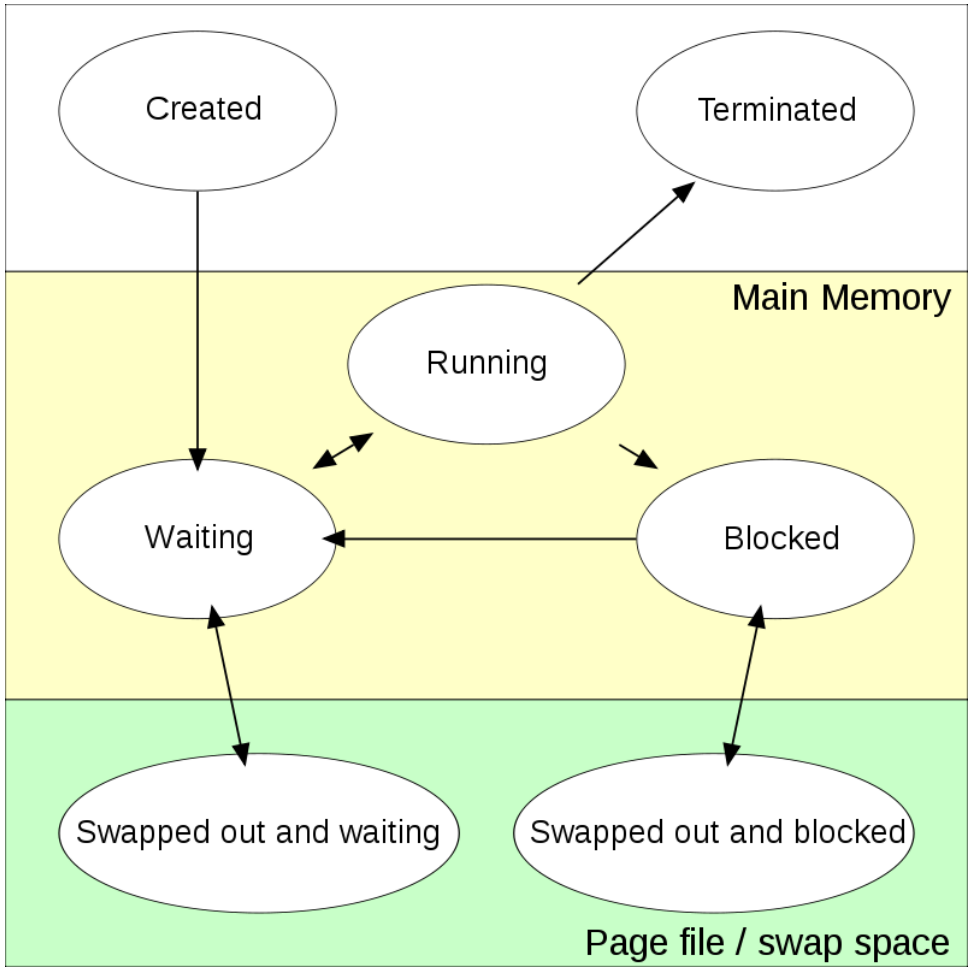


Source : http://www-igm.univ-mlv.fr/~dr/cs/node67.html

Rappel : Mode d'exécution dans un SE

- **Mode utilisateur :** Accès uniquement à son espace d'adressage
- **Mode noyau ou système** (*kernel mode*) **:** Exécution d'instructions propre au système, possibilité d'accéder à d'autres espaces d'adressage

Cycle de vie d'un processus



Source : Wikipedia

État d'un processus sous Unix

1. **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. **Exécution** (*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. **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.

Allocation du swap

- Gestion différente d'un disque
- Ensemble contigu de blocs de taille fixe
- Utilisation des fonctions
  - `swapalloc (int taille);` : demande d'unités d'échange
  - `swapfree (int adresse, int taille);` : demande de libération d'unités
- Possibilité d'avoir plusieurs unités d'echange
- Création et destruction dynamique

Gestion du swap

- Géré par le processus 0
- Allocation du swap
- Transfert vers la mémoire
- Transfert hors de la mémoire

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

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 (en général 2s)
- Si pas de possibilité : "re-scan" toutes les secondes

Problème : Étreinte fatale

- Tous les processus en mémoire sont “endormis”
- Tous les processus “prêts” sont transférés en swap
- Plus de place en mémoire
- Plus de place en échange sur le disque

Autres types de transfert

- Création d'un fils : possibilité de créer le fils en swap, lorsqu'il n'y a plus d'espace
- Accroissement de la taille d'un processus : Transfert du processus en swap avec l'allocation adéquate; au retour dans la mémoire le processus occupe plus de place

Explication de la commande `ps -l`

- 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
- S : 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)
- `UID` : User IDentifier
- `PID` : Process IDentifier
- `PPID` : Parent Process IDentifier

Explication de la commande `ps -l`

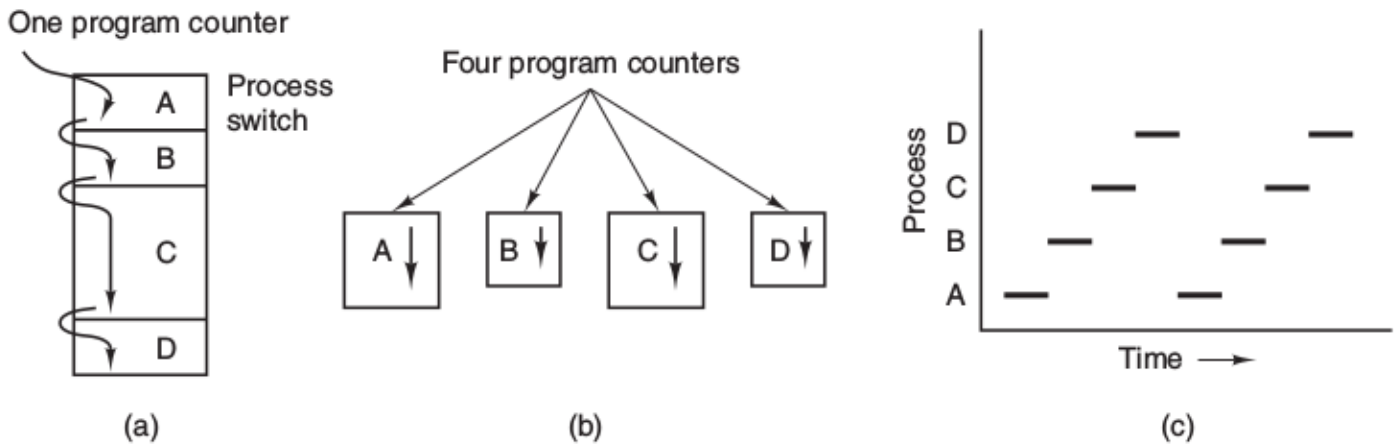
- C : Utilisation du processeur.
- PRI : Priorité du processus. Une valeur plus élevée veut dire une priorité plus basse.
- NI : Valeur de Nice
- ADDR : Adresse de la structure `proc`.
- SZ : Taille du processus
- WCHAN : Attente du processus
- TTY : Terminal associé
- TIME : Temps d'exécution cumulé
- CMD : Commande lancée

Plan

Ordonnanceur

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

- Interruption
- *Trap* ou déroutement sur erreur : extensions du mécanisme des interruptions (division par 0, débordement de pile, . . .); Contrôle passé au SE pour le traitement
- Appel au superviseur : interruptions destinées à un autre processus

Ordonnancement

- Système d'exploitation : plusieurs processus prêt simultanément
- Nécessité de faire un choix pour l'exécution

Choix d'un ordonnanceur

- Système de traitement par lots
  - Choix évident
  - Exécution du programme suivant dans la file
- Système multi-utilisateurs, multi-tâches et multi-processeurs
  - Choix complexe
  - Ordonnanceur peut évident à trouver
- Choix : dépend donc de l'utilisation de la machine

Critères

- *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

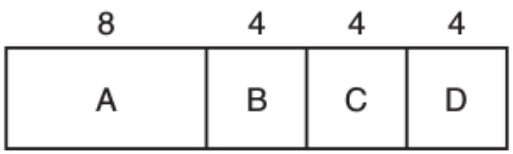
Techniques :

- **FCFS (*First Come First Served*) :** 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.
  - Algorithme facile à implanter.
  - Loin d'optimiser le temps de traitement moyen pour des processus avec temps d'exécution très différents.

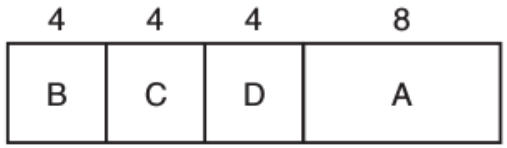
OSR : Ordonnanceur Sans Réquisition

Techniques :

- **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

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

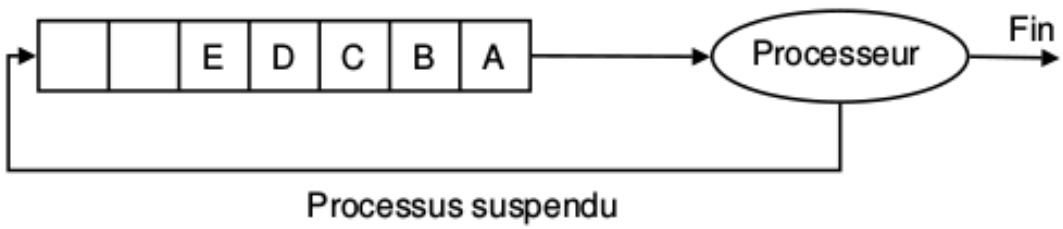
$t_{séjourB} = 4$
$t_{séjourC} = 4 + 4$
$t_{séjourD} = 4 + 4 + 4$
$t_{séjourA} = 4 + 4 + 4 + 8$
$\bar{t}_{séjour} = \frac{4*4+3*4+2*4+8}{4} = 11$

OAR : Ordonnanceur Avec Réquisition

Techniques :

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

Techniques :

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.

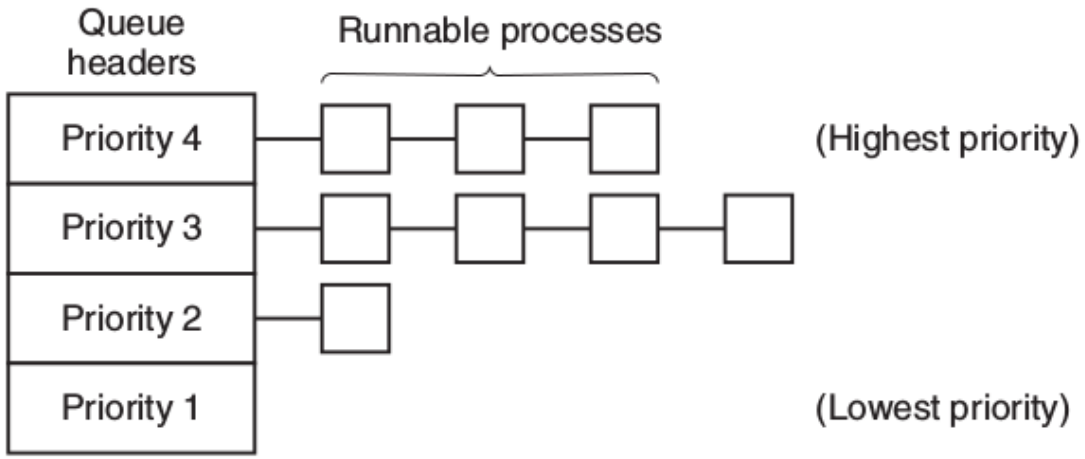
OAR : Ordonnanceur Avec Réquisition

Techniques :

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 avec priorité



- 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'ordonnanceurs sur :
http://lasdpc.icmc.usp.br/~ssc640/pos/i3s/

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