

---

# Systemes d'exploitation Processus

L3 Info JFC ISM Dakar

Nicolas GARRIC

# Besoin de parallélisme

- Les ordinateurs modernes ont besoin et sont capables de faire plusieurs choses en même temps
- Exemples :
  - Un serveur web doit :
    - répondre aux navigateurs qui lui demandent une page depuis le réseau
    - Lancer des requêtes disques si une page web n'est pas disponible en mémoire
  - Un SE doit :
    - Gérer la souris, le clavier, les différentes applications, ...



# Notion de processus

- Un processus est une abstraction d'un **programme en cours d'exécution**
- Il permet de transformer une unique UC en un ensemble d'UC virtuelles
- Cette transformation se fait grâce au **pseudo-parallélisme** : exécuter les processus les uns après les autres sur des périodes très courtes





# Métaphore culinaire

- Le programme est la recette
- Le processus est l'exécution de la recette
- Le processeur est le cuisinier
- ...Lorsque le cuisinier se fait piquer par une guêpe, il bascule vers un autre processus « trouver l'armoire à pharmacie et l'aspi venin »



# Indépendance des processus

- Pour que le pseudo parallélisme puisse fonctionner, chaque processus doit pouvoir s'exécuter indépendamment des autres
- Il doit avoir :
  - Son propre espace mémoire (code et données)
  - Ses propres registres et son propre compteur ordinal
  - Sa propre pile et son registre de pointeur de pile
  - Ses propres pointeurs sur fichiers ouverts





# Contexte d'un processus

- Mais un ordinateur contient 1 UC, 1 mémoire, 1CO, ...
- On définit alors le contexte d'un processus
  - La valeur de tous les registres (y compris CO et RE)
  - Son espace d'adressage : chaque processus possède une zone mémoire qu'il est (a priori) seul à pouvoir accéder. Cet espace contient données, code du programme et pile d'exécution
  - La liste des descripteurs de fichiers ouverts





# Changement de contexte

- Pour réaliser le pseudo parallélisme, on change de contexte dès qu'un processus a épuisé le temps qui lui était imparti
- Après la piquêre de la guêpe, le cuisinier doit noter sur un bout de papier la où il s'est arrêté
- Le changement de contexte nécessite donc que les contextes des processus soient stockés quelque part
- Cet endroit s'appelle : la table des processus



# Table des processus

- Le SE maintient un tableau de structures appelé **table des processus**
- Chaque structure contient :
  - La valeur des registres dont CO , RE et pointeur de pile
  - L'état du processus (actif, suspendu, ...), sa priorité, son temps d'exécution
  - L'ID du processus, du parent, du groupe
  - Une description de l'espace d'adressage du processus
  - Une description des fichiers ouverts







# Les interruptions

- Comment un ordinateur gère-t-il l'interactivité ?
- Comment tenir compte d'un clic souris ? De l'appui sur une touche ?
- Tous les périphériques d'E/S possèdent un mécanisme d'interruption.
- Lorsqu'un périphérique génère une interruption, le processeur suspend le processus actif et lance un processus de traitement d'interruption



# Vecteur d' interruption

- Un emplacement mémoire (**vecteur d'interruption**) est associé à chaque périphérique
- Ce vecteur d'interruption contient l'adresse du programme de traitement d'interruption
- Au moment d'une interruption, un processus de traitement de l'interruption est alors lancé
  - Celui-ci sauvegarde le processus en cours dans la table des processus
  - Gère l'interruption
  - Relance le processus précédemment interrompu





# Gestionnaires d'interruption

- Le mécanisme d'interruption est visible au niveau le plus bas de la machine : en assembleur
- Il n'est pas visible pour un programmeur C, JAVA, ...
- Les procédures de traitement d'interruptions sont donc écrites en Assembleur



# Création de processus

- Pour fonctionner un SE a besoin que des processus soient lancés
- Il y a trois circonstances permettant de lancer un processus :
  - Initialisation du système
  - Requête utilisateur (shell ou gestionnaire de fenêtres)
  - Appels systèmes



# Initialisation du système

- Lors de l'amorçage du système le bios passe la main au SE : c'est à dire le processus d'initialisation du SE démarre.
- Celui-ci lance de nombreux processus aux fonctions spécialisées qui tournent en parallèle :
  - Les processus de premier plan : ceux qui interagissent avec l'utilisateur (en particulier le gestionnaire de fenêtres)
  - Les processus d'arrière plan (les **démons**): serveur de mail, serveur web, serveur d'impression, .... :





# Lister les processus

Pour accéder aux processus en cours d'exécution:

- Sous UNIX :
  - La commande shell **ps**
  - Le **Moniteur système**
- Sous Windows:
  - Le **Gestionnaire de tâches**

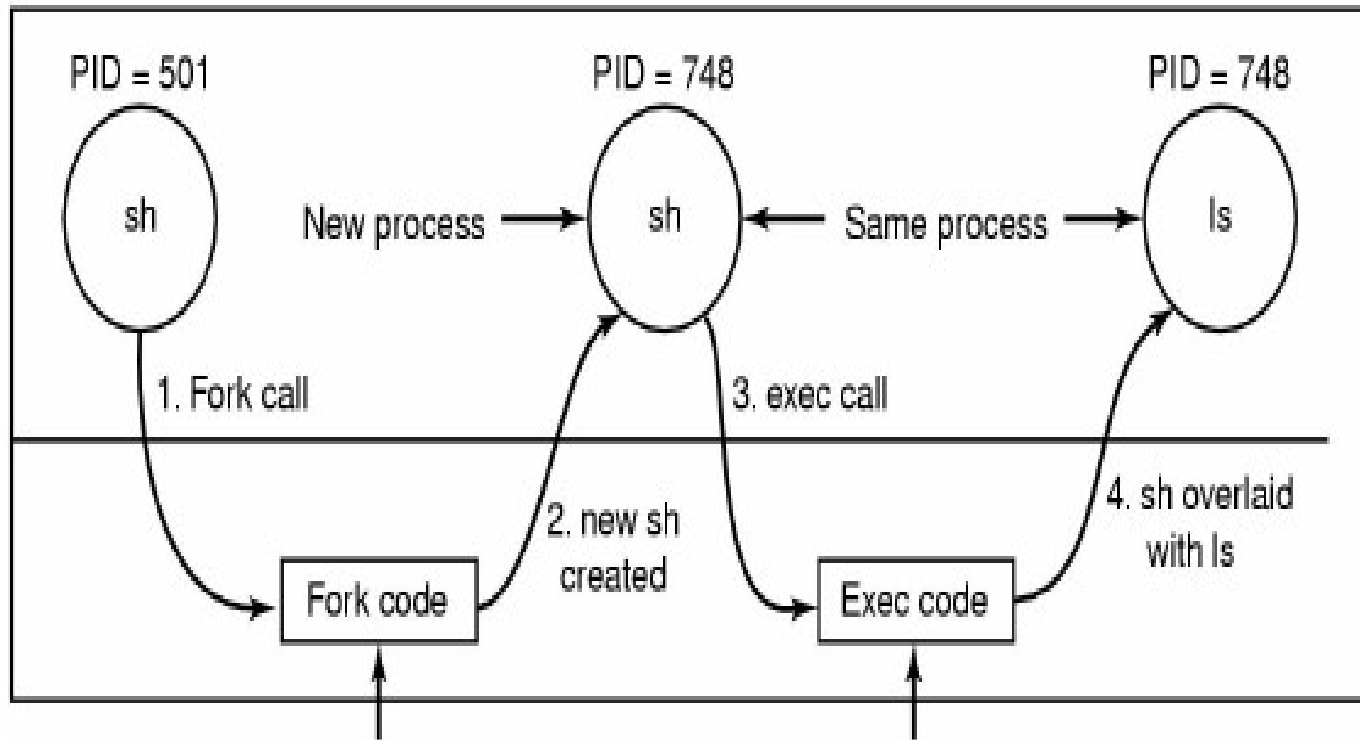


# Appels systèmes

- Un processus peut créer d'autres processus en effectuant des appels systèmes
- Sous UNIX :
  - **fork()** est le seul appel système qui permet de créer un processus
  - Il crée un clone du processus appelant
  - Après l'appel les 2 clones ont la même image mémoire, les mêmes fichiers ouverts, ...
  - Exemple : une commande est tapée dans un shell : **fork()** est appelé et le processus enfant exécute le code de la commande tapée grâce à l'appel **execve()**



# Exemple d'exécution de ls dans sh



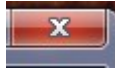


# Appels systèmes

- Sous Windows:
  - L'appel système **CreateProcess()** permet de créer un nouveau processus ...
  - ....et le chargement du nouveau programme dans le nouveau processus
- Dans tous les cas :
  - Au moment de la naissance , il y a copie de l'espace d'adressage
  - Le processus enfant possède son **propre** espace d'adressage



# Fin normale d'un processus

- Normalement un processus se termine par un appel système de terminaison :
  - Sous UNIX : **exit()**
  - Sous Windows : **ExitProcess()**
- Cet appel système est appelé par les applications graphiques quand on clique sur 



# Fin anormale d'un processus

- Si le processus provoque une erreur :
  - Division par 0
  - Accès mémoire non autorisé
  - ...
- Une interruption logicielle est effectuée :
  - Un processus de gestion de l'interruption est lancé...
  - ... qui arrête le processus incriminé
  - ... et l'enlève de la table des processus



# Fin d'un processus par signal

- Un processus peut aussi s'arrêter si un autre processus fait un appel système qui demande l'arrêt du processus :
  - Sous UNIX : **kill()**
  - Sous Windows **Terminate-Process()**
- Exemples :
  - La commande **kill** du shell UNIX
  - Le bouton « Arrêter le processus » du gestionnaire de tâches Windows





# Hiérarchie de processus sous UNIX

- Sous UNIX, tout processus créé possède le nom de son parent (sauf init)
- On obtient ainsi une arborescence de processus dont init est la racine
- Chaque processus qui naît hérite des propriétés de son processus père (puisque'il est créé par copie)
- Tous les processus lancés au démarrage sont les fils de init





# Processus sous Windows

- Sous Windows , il n'y a pas de notion de parent
- Tous les processus sont au même niveau
- CreateProcess renvoie un descripteur (handle) du processus fils, mais ce descripteur peut être utilisé par n'importe quel autre processus

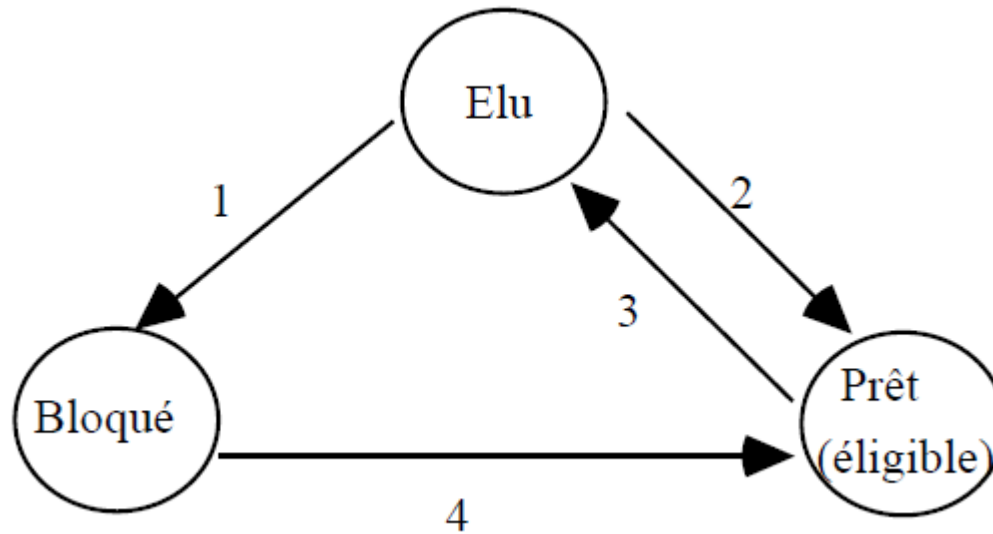


# État des processus

- un processus peut se bloquer car il attend une ressource :
  - que l'imprimante soit prête
  - que l'utilisateur tape quelque chose au clavier ....
  - Il se retrouve alors dans l'état **Bloqué** : ne pouvant s'exécuter tant qu'un événement extérieur ne se produit pas
- Un processus peut être interrompu par le système :
  - car il a utilisé tout son quantum d'UC
  - Il se retrouve alors dans l'état **Prêt** : exécutable mais arrêté pour laisser la place aux autres



# Transitions entre les états



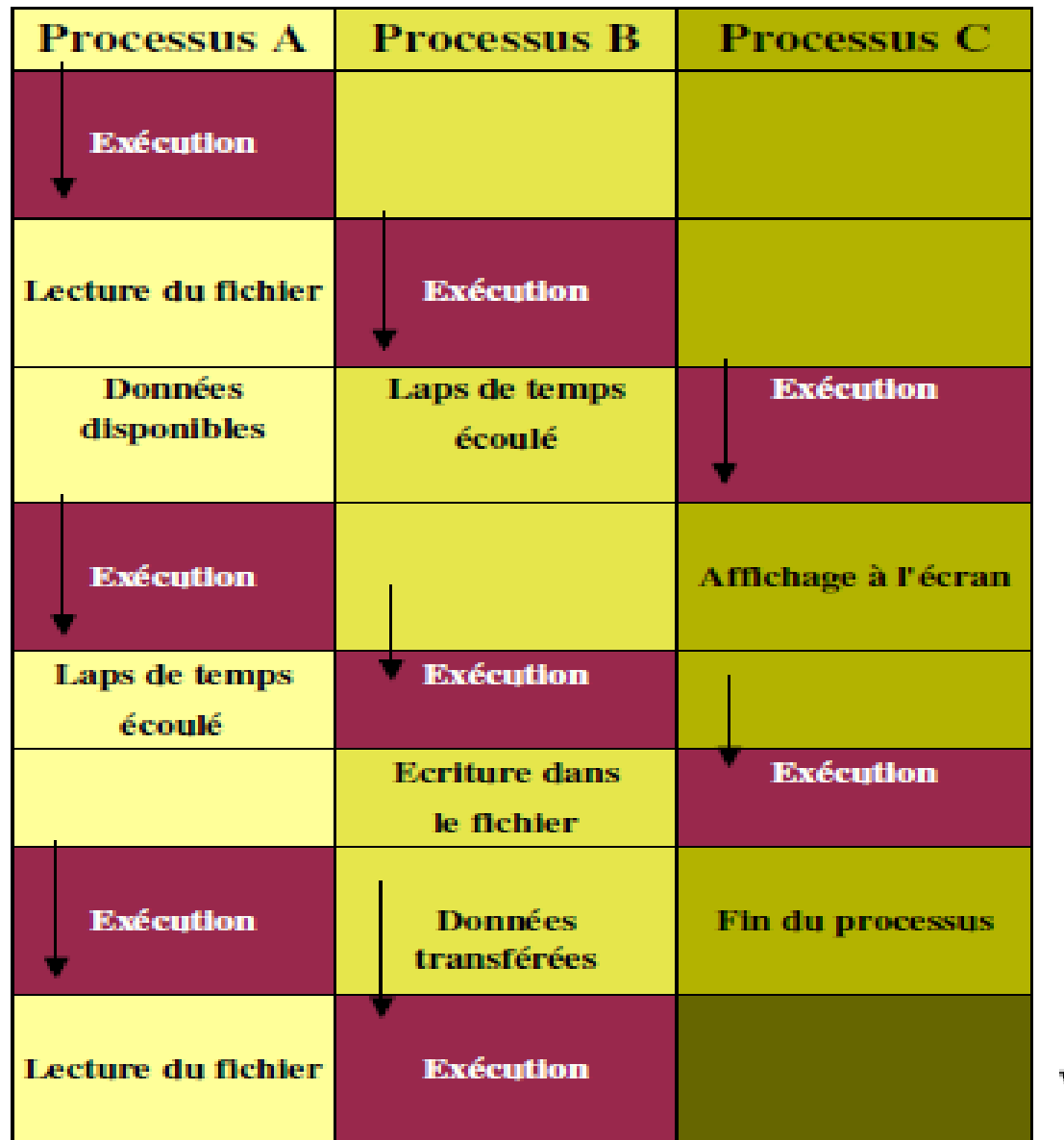
1 : blocage en attente d'une donnée

2,3 : effectué par l'ordonnanceur

4 : la donnée devient disponible







temps



# Autres états intermédiaires

- processus s'exécutant en mode noyau,
- processus s'exécutant en mode utilisateur,
- processus résidant en mémoire principale,
- processus en cours de création
- processus en cours de terminaison (Zombie).



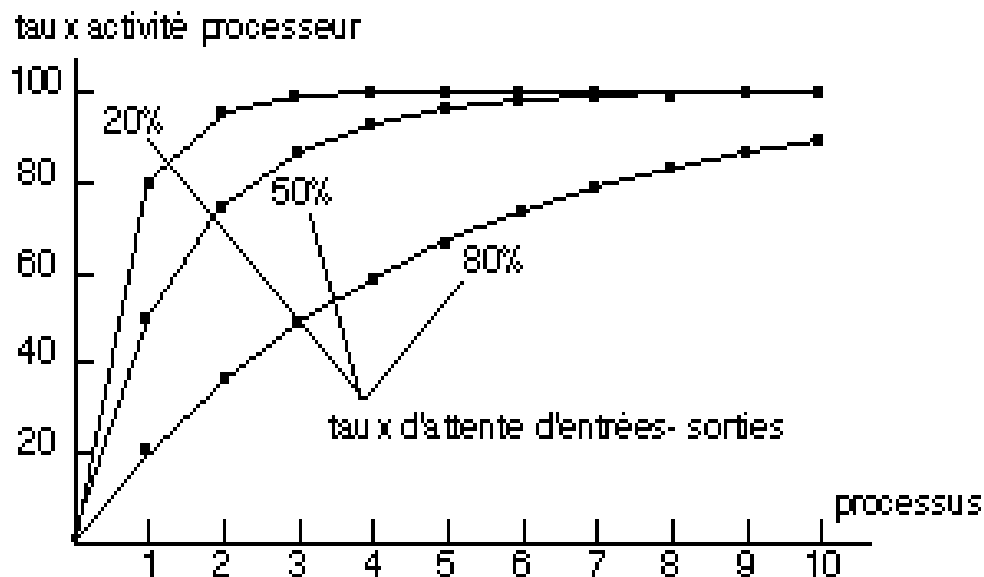
# Multiprogrammation

- Chaque processus travaille en moyenne 20% et attend des ressources (mémoire, disque, imprimante, ...) pendant 80%
- Si trop peu de processus sont exécutés :
  - l'UC passe son temps à ne rien faire (en attendant que les ressources soient disponibles)
- Si trop de processus sont exécutés :
  - Chaque processus n'a pas assez de temps CPU pour avancer
- Bien régler la multiprogrammation permet d'améliorer l'utilisation de l'UC et les exigences des processus



# Modéliser la multiprogrammation

- Supposons qu'un processus attende  $p$  % du temps
- Le taux d'utilisation de l'UC est de  $1-p^n$



# Conséquence

- Si on a un ordinateur de 512 Mo de RAM et si chaque processus utilise 128 Mo et si  $p=0,8$  :
  - 3 programmes + le SE peuvent être en mémoire ensemble
  - Le taux d'utilisation de l'UC =  $1-0.8^3 = 49\%$
- Si on ajoute 512 Mo de RAM :
  - 7 programmes + le SE peuvent être en mémoire ensemble
  - Le taux d'utilisation de l'UC =  $1-0.8^7 = 79\%$
- Morale : la RAM doit être bien dimensionnée pour un processeur donné



# Droits et processus

- Un processus est lancé avec les droits (UID et GID) de l'utilisateur qui lance le processus.
- Le GID utilisé correspond au groupe **principal** de l'utilisateur (celui qui est mentionné dans /etc/passwd)
- Il s'agit de l'UID et du GID **réels** du processus : ils identifient l'utilisateur qui a lancé le processus





# Identificateurs effectifs

- l'UID et le GID **effectifs** (EUID et EGID), identifient les droits d'un processus.
- En général les identifiants réels et effectifs sont les mêmes....
- ... sauf dans certains cas.



# Retour sur /etc/passwd

- Le fichier **/etc/passwd** est protégé en écriture pour un utilisateur normal
- Or il peut changer son shell (**chsh**).
- Ceci est possible grâce au **SetUser ID** (SUID)
- C'est lorsque sur le groupe de permissions « propriétaire » le x se change en s :

```
-rwsr-xr-x root root ... /usr/bin/chsh  
-rwsr-xr-x root root ... /usr/bin/passwd
```





# Fichiers SetUID et SetGID

- Lorsque le x est remplacé par un s : **l'EUID** (effectif) du processus est celui du propriétaire du fichier exécuté (et non pas celui de l'utilisateur qui l'a lancé) :
  - on parle alors de fichier **SetUID**
- De même lorsque le x est remplacé par un s dans le groupe « groupe propriétaire » : **l'EGID** (effectif) du processus est celui du groupe propriétaire du fichier exécuté (et non pas celui de l'utilisateur qui l'a lancé) :
  - on parle alors de fichier **SetGID**



# chmod pour SetUID et SetGID

- On utilise **chmod** pour mettre un fichier en SetUID ou SetGID
  - `chmod u+s nom_du_fichier` pour activer le Setuid
  - `chmod g+s nom_du_fichier` pour activer le Setgid
- On peut utiliser la notation octale
  - `chmod 4755 nom_du_fichier` positionne le setuid
  - `chmod 2755 nom_du_fichier` positionne le setgid



# Sticky bit

- Quand on met à s le droit d'exécution du groupe de permissions « Autres utilisateurs ».
- Pour un fichier :
  - `-rwxr-xr-s root root ... fichier`
  - cela oblige le code du fichier à rester en mémoire vive
- Pour un répertoire :
  - `drwsxr-sr-x root root ... repertoire`
  - Modification plus fine des droits d'accès

