

# Systèmes d'exploitation, 2ème année

Fichiers et processus

Yves STADLER

Université Paul Verlaine - Metz

18 octobre 2011

# Les processus

## Définition

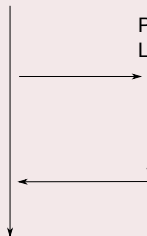
- Entité associée par le noyau à l'exécution d'un programme. Un programme peut être exécuté par un utilisateur et peut avoir plusieurs processus associés.

## Hiérarchie des processus

- Hiérarchie arborescente ;
- Relation père-fils ;
- Processus *init* (pid : 1), racine ;
- *daemon* (cron, mémoire, messagerie, ... ) ;
- indépendants.

Processus  
Vador

Processus  
Luke



# Les processus

## Initialisation

- Le noyau est chargé par le bootstrapping;
- Le noyau va résider en mémoire tant que la machine est allumée.
- Au démarrage le noyau exécute le processus `init` pid 1;
- `init` lit `/dev/tty` qui décrit les terminaux;
- `init` créer un fils pour chaque terminal et se met en veille;
- chaque fils exécute le programme `login`.

## Deux espaces différents

- Espace usager : information du processus, fichiers ouverts, registre
- Espace kernel : fonction de tous les processus en appels système.

## Deux contextes différents

- Processus : le kernel opère pour le compte du processus (appels système) ;
- Système : le kernel gère les problèmes du système : interruptions périphérique, ...

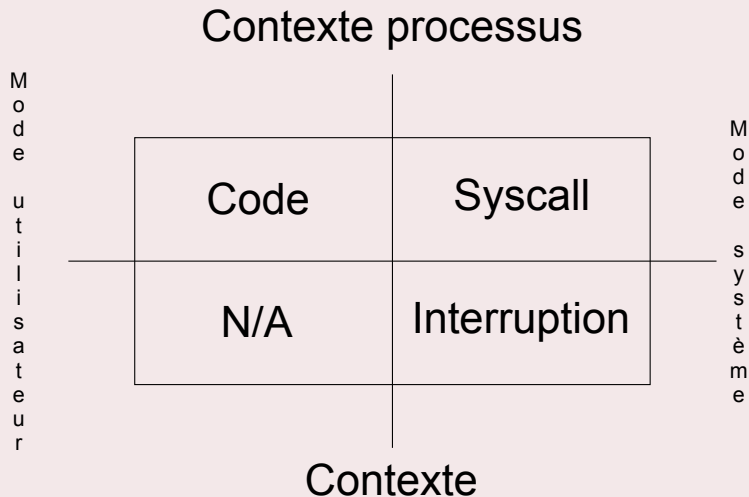


FIGURE: Modes et contextes

# États d'un processus

## Définition

- Un processus peut se décrire comme une alternance de section actives, durant lesquelles des unités de temps CPU sont consommées, avec des attentes d'entrées sorties.

## Deux mécanismes de gestion

- Ordonnancement (*scheduling*) : choix des processus à activer ;
- Synchronisation : gestion des ressources partagées ;
- Le *Process Control Block* : structure qui représente l'état d'un processus (lorsqu'il n'est pas en mémoire).

# États d'un processus

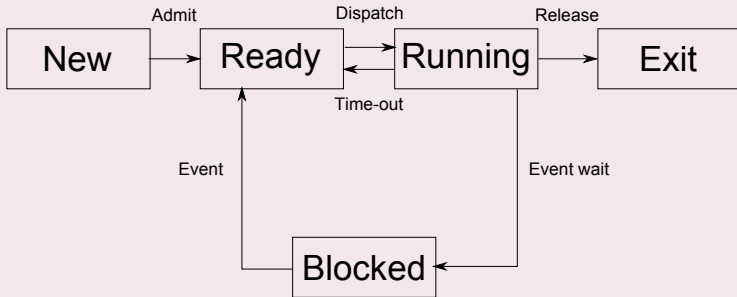
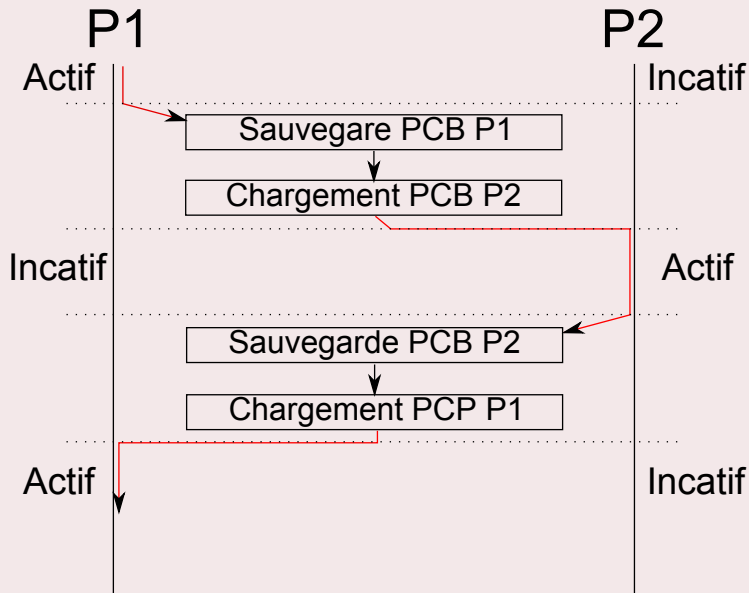


FIGURE: État simplifié des processus



# Changement de contexte



# États d'un processus

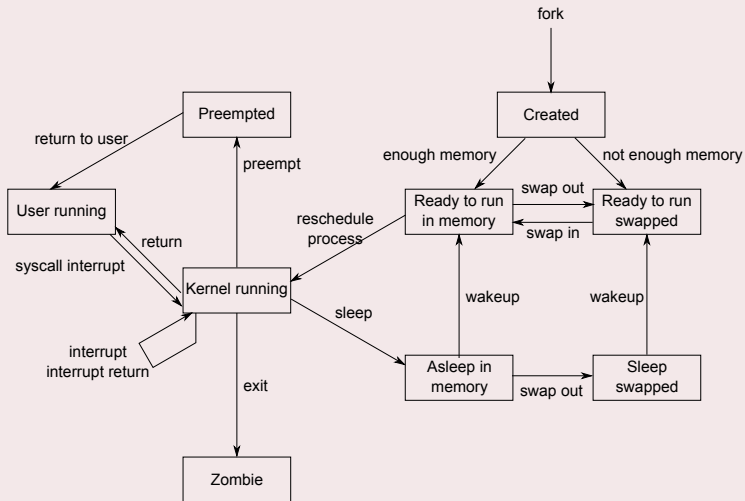


FIGURE: État d'un processus

# Primitives

## Création

- `fork()` (Retourne 0 dans le processus fils, pid de l'enfant dans le processus père) ;
- héritage de l'image mémoire ;
- héritage de la table des fichiers ouverts ;
- copie du PCB du père à l'emplacement du PCB du fils (seul différence les pid).

## Identité

- `getpid()` donne le numéro du processus en cours d'exécution ;
- `getppid()` donne le numéro du père du processus en cours d'exécution.

# Les appels systèmes

## Processus

- `fork()` ;
- `kill()` ;
- `waitpid()` ;
- `alarm()` ;
- `pause()` ;

## Répertoires

- `mkdir()` créer un répertoire ;
- `chdir()` change de répertoire ;

# Les appels systèmes

## Fichier

- `creat()` créer un fichier ;
- `open()` ouvre un descripteur de fichier en lecture ;
- `close()` ferme le descripteur de fichier ;
- `read()` read the file descriptor ;
- `write()` write in the file descriptor ;

## Mémoire

- `malloc()` allocation de mémoire ;
- `free()` libération de la mémoire ;

# Ordonnancement

## Rôle de l'ordonnanceur

- Choisir les processus qui vont accéder au CPU
- Lorsqu'il faut gérer de la mémoire virtuelle, l'ordonnancement se fait à deux niveaux
  - Haut niveau : sélection les prochain processus à charger en mémoire
  - Bas niveau : sélectionne le processus qui accède au CPU parmi ceux qui sont prêts.

## Les files d'attentes

- Un ordonnanceur gère plusieurs files d'attentes :
  - Processus prêts ;
  - Processus en attente d'entrée / sortie.
- La question est : comment insère-t-on un processus dans une file et
- comment un processus passe d'une file au CPU.

# Traitement par flots

## Batch processing

- Pas de répartiteur de bas niveau (ou répartiteur trivial)
- Lorsqu'une opération d'entrée sortie est déclenchées, le CPU est inactif.

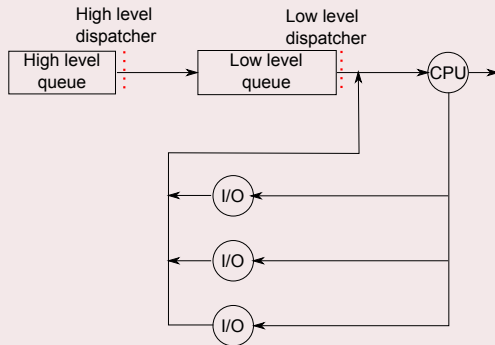


FIGURE: État d'un processus

# Multi-programmation

## Multi-programmation

- Lorsqu'une opération d'entrée sortie est déclenchées, le CPU est libéré ;
- les processus qui termine une entrée sortie sont remis dans la file.

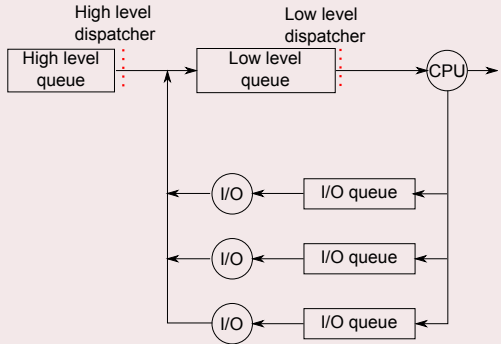


FIGURE: État d'un processus



# Temps partagé

## Time sharing

- Chaque processus dispose d'un certain temps de CPU ;
- les processus qui dépasse ce délai sont remis dans la file ;
- On appelle quantum de temps la durée maximale allouée à chaque processus.

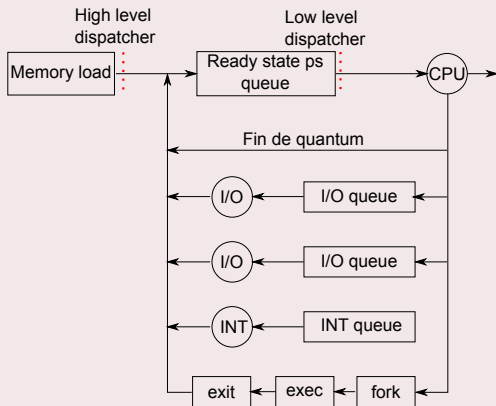


FIGURE: État d'un processus

# Mémoire virtuelle

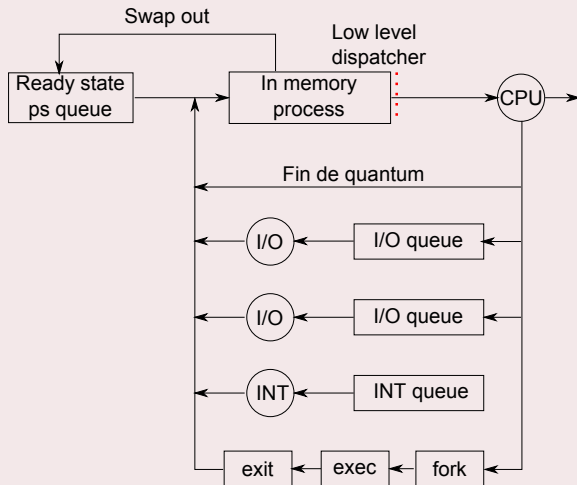


FIGURE: État d'un processus

# Les algorithmes d'ordonnancement

## Plan

- Critères d'ordonnancement ;
- First In First Out ;
- Shortest Job First ;
- Round Robin ;
- Listes multiples ;
- Évaluations.

# Les critères

## Minimiser, maximiser ?

- On peut minimiser
  - le temps entre soumission et fin d'une tâche
  - le temps de passage en file bas niveau ;
  - le temps d'exécution des entrées sorties.
- maximiser :
  - Le taux d'activité du CPU ;
  - Le nombre de processus traités par unités de temps.

# Les critères

## Classes d'ordonnanceurs

- Non-préemptifs : un processus libère le CPU quand il n'en a plus besoin ;
- Préemptifs : l'utilisation du CPU est limitée dans le temps.

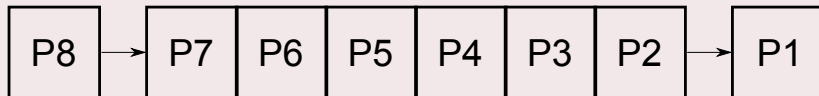
## Classes d'ordonnanceurs

- Préemptif : Unix, WinNT, BeOS, Win2000, MacOS X (et suivants) ;
- Non préemptif : MacOS, Win9X, Millenium.

# First In First Out

## FIFO

- Principe d'une file standard, les éléments rentre à la suite de tous les processus déjà présents ;
- Le processus qui n'a plus de prédécesseur est la tête de la file et est élu à la prochaine sélection.



# Shortest Job First

## SJF

- La tâche la plus courte à la priorité
- Les processus sont triés en fonction du nombre d'unités CPU qu'il requiert.
- En cas d'égalité : FIFO.
- Comment connaît-on le temps CPU des processus ?

## Estimation du temps CPU

- Demander à l'utilisateur ;
- Estimer à partir de l'historique :  $p_{n+1} = a * t_n + (1 - a) * p_n$
- SJF avec priorité

# Shortest Job First

## Problème

- Comment éviter les famines ?

## Problème

- Ajouter un vieillissement automatique des processus.



# Round Robin

## Principe

- FIFO préemptif
- Utilisation d'un quantum de temps
- Libération du CPU lors des entrées sorties, de la fin du processus ou lorsque le quantum est épuisé.

## Quel quantum choisir

- Trop court : important changement de contexte qui pénalise l'efficacité du système ;
- Trop long : les processus court sont pénalisés ;
- Infini : on a une FIFO ;
- Unix utilise 100ms.

# Files multiples

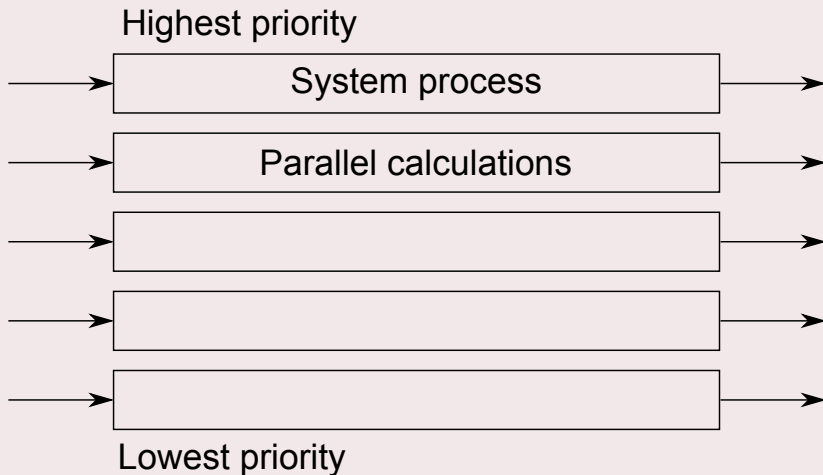
## Principe

- Division en groupe de processus (système, professeurs, étudiant, avant plan, arrière-plan, ...);
- Différentes priorités par files (éventuellement quantum différents aussi).

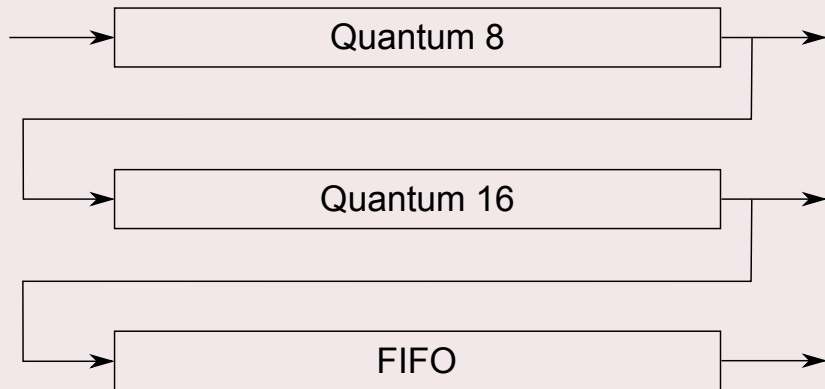
## Caractéristiques

- Adapté à des contraintes particulières ;
- Priorité en fonction de l'importance des travaux (de manière *meta*)
- Généralement utile pour privilégier les processus dépendant d'entrées sorties ;
- Un processus qui quitte volontairement l'UC garde sa file ;
- Lors des dépassements de quantum, on change le processus de file (perte de priorité au profit d'un quantum plus long).

# Files multiples



# Files multiples



## Suite

- TD : Processus et ordonnancement.