

Systèmes d'exploitation, 2ème année

Processus et ordonnancement

Yves STADLER

Université Paul Verlaine - Metz

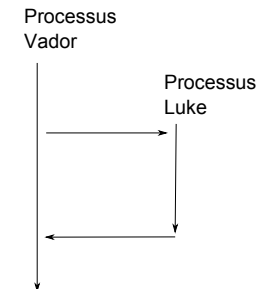
5 octobre 2011

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.



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`.

Modes et contexte

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, . . .

Modes et contexte

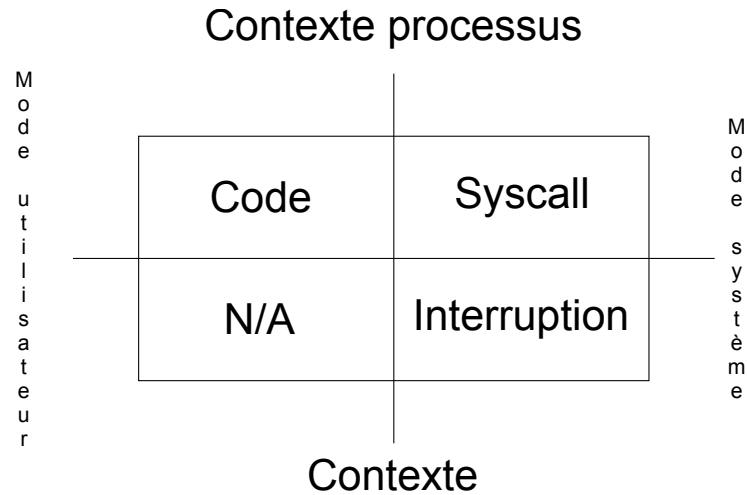


FIGURE: Modes et contextes

5/29

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.

Mécanismes de gestion

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

6/29

Vision des états de processus court terme

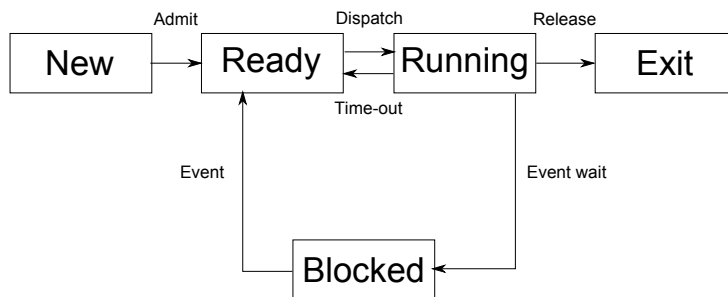
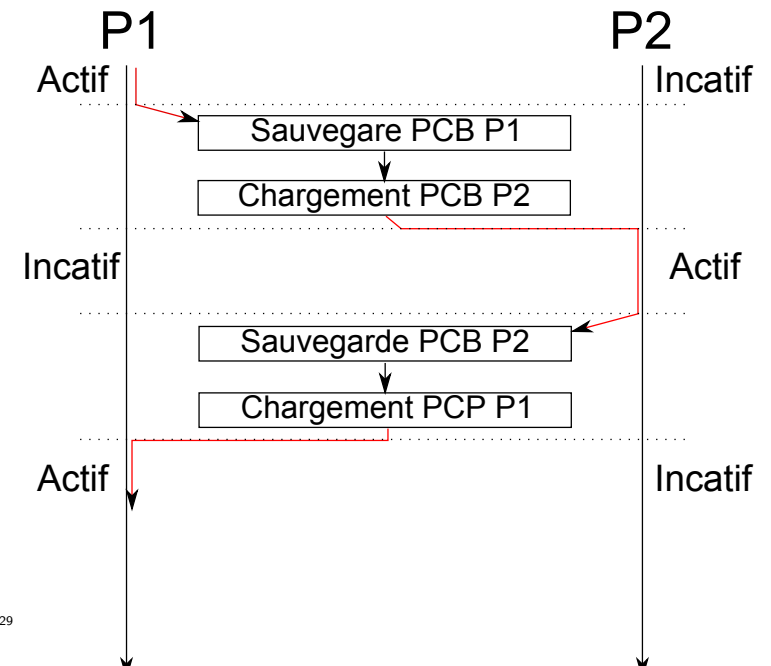


FIGURE: État simplifié des processus

7/29

Changement de contexte



8/29

Vision des états de processus long terme

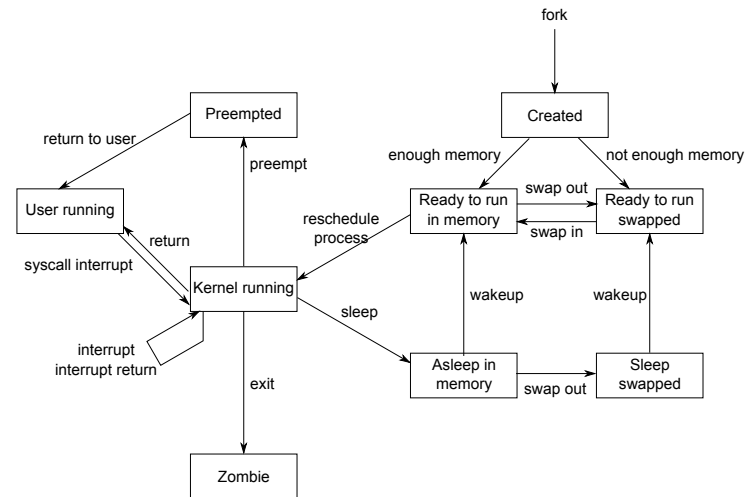


FIGURE: État d'un processus

9/29

Les appels systèmes

Processus

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

Répertoires

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

11/29

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.

10/29

Les appels systèmes

Fichier

- `creat()` créer un fichier ;
- `open()` ouvre un descripteur de fichier en lecture ;
- `close()` ferme le descripteur de fichier ;
- `read()` lit le descripteur de fichier ;
- `write()` écrit sur le descripteur de fichier ;

Mémoire

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

12/29

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 plusieurs niveaux
 - Long terme (Long-term/admission scheduler) : sélectionne les prochains processus à charger en mémoire;
 - Moyen terme (Medium-term scheduler) : sélectionne les processus qui vont être temporairement enlevés de la mémoire centrale (défaut de page ...);
 - Court terme (Short-term scheduler) : sélectionne le processus qui accède au CPU parmi ceux qui sont prêts (Le dispatcher les activera.).

13/29

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.

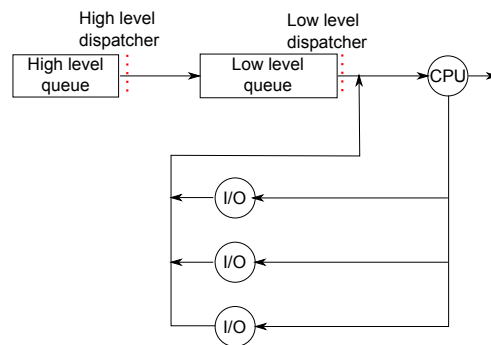


FIGURE: État d'un processus

15/29

Ordonnancement

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.

Les files d'attentes

- Traitement par flots;
- Multi-programmation;
- Temps partagé;
- Mémoire virtuelle.

14/29

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.

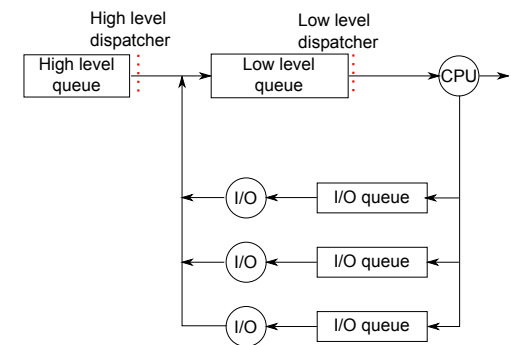


FIGURE: État d'un processus

16/29

Temps partagé

Time sharing

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

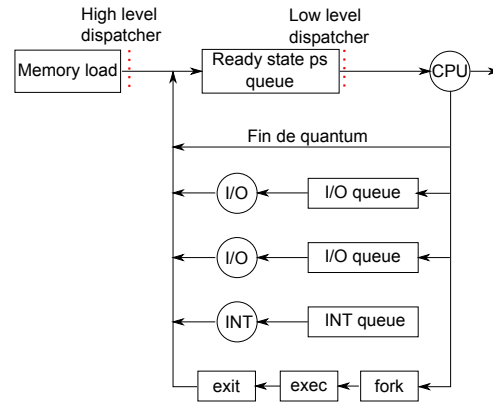


FIGURE: État d'un processus

17/29

Mémoire virtuelle

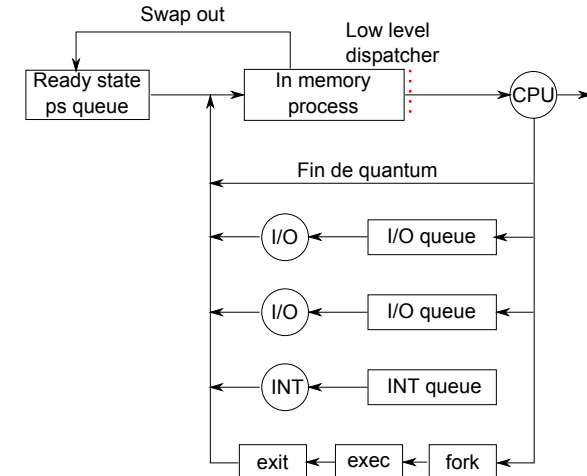


FIGURE: État d'un processus

18/29

Les algorithmes d'ordonnancement

Plan

- Critères d'ordonnancement ;
- First In First Out (FIFO) a.k.a. First Come First Served (FCFS) ;
- Shortest Job First (SJF) ;
- Round Robin ;
- Listes multiples ;
- Évaluations.

19/29

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.

20/29

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.

21/29

Shortest Job First

SJF

- La tâche la plus courte à la priorité (un autre processus *peut* être interrompu).
- 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 ?
- Famines possibles.

Estimation du temps CPU

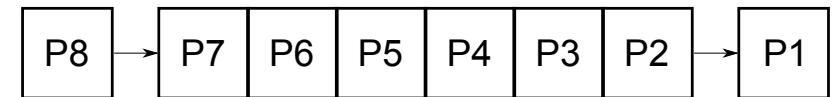
- Demander à l'utilisateur ;
- Estimer à partir de l'historique : $p_{n+1} = a * t_n + (1 - a) * p_n$
- SJF avec priorité

23/29

First In First Out

FIFO

- Principe d'une file standard, les éléments rentrent 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.



FIFO

- Changement de contexte à la fin du processus. Pas de réorganisation ;
- Les longs processus peuvent accaparer le processeur ;
- Problèmes lorsque certains processus ne se terminent pas.

22/29

Shortest Job First

Problème

- Comment éviter les famines ?

Problème

- Ajouter un vieillissement automatique des processus.

24/29

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.

Caractéristiques

- Meilleur que FIFO car les processus se terminent plus vite (pas de blocage pour un seul processus) ;
- Meilleur que SJF car les processus longs ne sont plus pénalisés ;
- Le temps d'attente dépend du nombre de processus ;
- Famine impossible ;
- Difficilement exploitable en temps-réel (deadlines).

25/29

Round Robin

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.

26/29

Files multiples

Principe

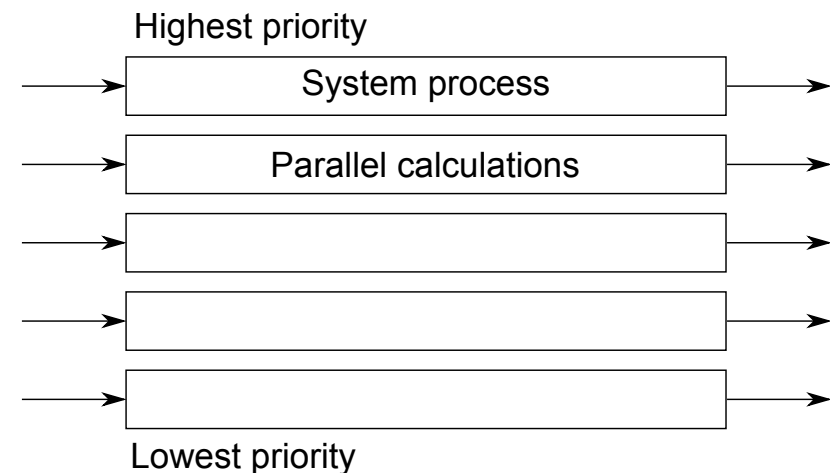
- Division en groupe de processus (système, professeurs, étudiant, avant plan, arrière-plan, ...) ;
- Différentes priorités par files (éventuellement quantum différents aussi).

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).

27/29

Files multiples



28/29

Files multiples

