Système d'exploitation

Processus

Florent Devin

Ecole Internationale des Sciences du Traitement de l'Information

Processus de démarrage

Démarrage

BIOS

- Contenu dans le ROM
- Lit le premier secteur de la partition activée
- Disque : MBR
- Unix : charge un *loader*
  - LILO : LInux LOader
  - GRUB : GRand Unifier Bootlaoder
- Doivent charger le système

Plan

Processus de démarrage

Initialisation

- Décompression du noyau linux
- Initialisation des périphériques (disque, réseau, son, ...)
- Configuration “automatique” des pilotes
- Montage du système de fichier racine
  - Lecture seule : permet de vérifier s’il y a des erreurs
- Lacement de `init`

Processus de démarrage

Initialisation

- Lecture du fichier `/etc/inittab`
- Lancement des processus contenu dans `/etc/rc<n>.d/`
- Commutation vers le mode multi utilisateurs
- Exécution de `getty` pour chaque console
- Remontage du système de fichier en lecture-écriture
- Exécution des processus de services (`daemon`)
- Construction d'une arborescence de processus

Arrêt du système

Procédé d'arrêt

- Nécessité d'arrêter le système "proprement"
- Risque de corruption du système de fichiers (utilisation de cache en écriture)
- Commande : `shutdown` provoque l'arrêt
- Commande : `shutdown -r` ou `reboot` provoque le redémarrage
- Démontage des systèmes de fichiers, arrêt des processus utilisateurs et service, enfin le système est stoppé

Plan

Processus `init`

`init`

- Lancement du noyau : démarrage du processus `init`
- Père de tous les processus : PID 1
- Lecture du fichier `/etc/inittab`
  - Informations de la forme : `id:runlevel:action:processus`
  - `id` : identifieur
  - `runlevel` : niveau de démarrage
  - `action` : manière de lancer le processus
  - `processus` : ce qui est lancé

Processus `init`

`action`

- Manière de lance le processus :
  - `respawn` : relance le processus une fois celui-ci terminé
  - `wait` : attendre le fin du processus avant de continuer
  - `boot` : doit être lancer au démarrage uniquement
  - `ctrlaltdel` : processus lance quand séquence tapée

Niveau de démarrage

runlevel

- État dans lequel se trouve le système
- Pas de standard
  - 0 : Arrêt de la machine
  - 1 : Single user
  - 2-6 : laisser à la disposition de l'utilisateur
- `init` recherche la ligne qui contient `initdefault` et lance le runlevel associé.
- Possibilité de définir un nouvel état par `telinit` ou `init`

Processus

Plan

Simultanéités

Simultanéité

- Activation de plusieurs processus en même temps
- *Simultanéité totale ou vraie* : nombre de processus $\leq$ nombre de processeur
- *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.

Mécanismes de commutation

Mécanismes

- 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

Plan

Introduction

Plusieurs processus

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

Choix d'un ordonnanceur

Comment choisir

- 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

Différents 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
- Problème : plusieurs critères mutuellement contradictoires (KLEINOCK - 1975)

Problèmes

Problèmes

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

Famille d'ordonnanceur

Deux familles

- sans réquisition (OSR) : nouveau processus sur blocage ou terminaison du processus courant
- avec réquisition (OAR) : à intervalle régulier, l'ordonnanceur reprend la main et élit un nouveau processus actif

Plan

Sans réquisition

- FCFS : First Come First Served. Ordre d'arrivée en gérant une file des processus. Cet algorithme est facile à implanter, mais il est loin d'optimiser le temps de traitement moyen
- 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.
  - Temps d'attente moyen minimal SI toutes les tâches sont présentes dans la file d'attente au moment où débute l'assignation

Sans réquisition

- Modification de priorité : Utilisateur fixe des priorités
  - Problème : Risque de famine pour les priorités faibles
  - $\Rightarrow$ Augmentation de la priorité avec le temps d'attente
  - Problème : Possibilité de dégradation importante du système
  - $\Rightarrow$: Préemption

Plan

Ordonnancement circulaire

Round robin

- Un des plus simples et des plus robustes
- Attribution d'un quantum de temps
- Temps d'exécution maximal pour un 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

Ordonnancement circulaire

Round robin

- Quantum de temps trop petit : gaspillage des ressources processeurs
- Quantum de temps trop grand : peu d'interactivité
- Quantum acceptable : un centaine de ms

Ordonnancement avec réquisition

PCTE R

- Extension de l'algorithme PCTE
- Création d'un quantum de temps
- Réquisition à la fin du quantum

Algorithme à queues multiples

Plusieurs queues

- Besoin de différencier facilement les processus 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

Ordonnancement à deux niveaux

Ordonnancement à deux niveaux

- Taille de la mémoire centrale : peut être insuffisante pour contenir tous les processus prêts à être exécutés
- Certains sont contraints de 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

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

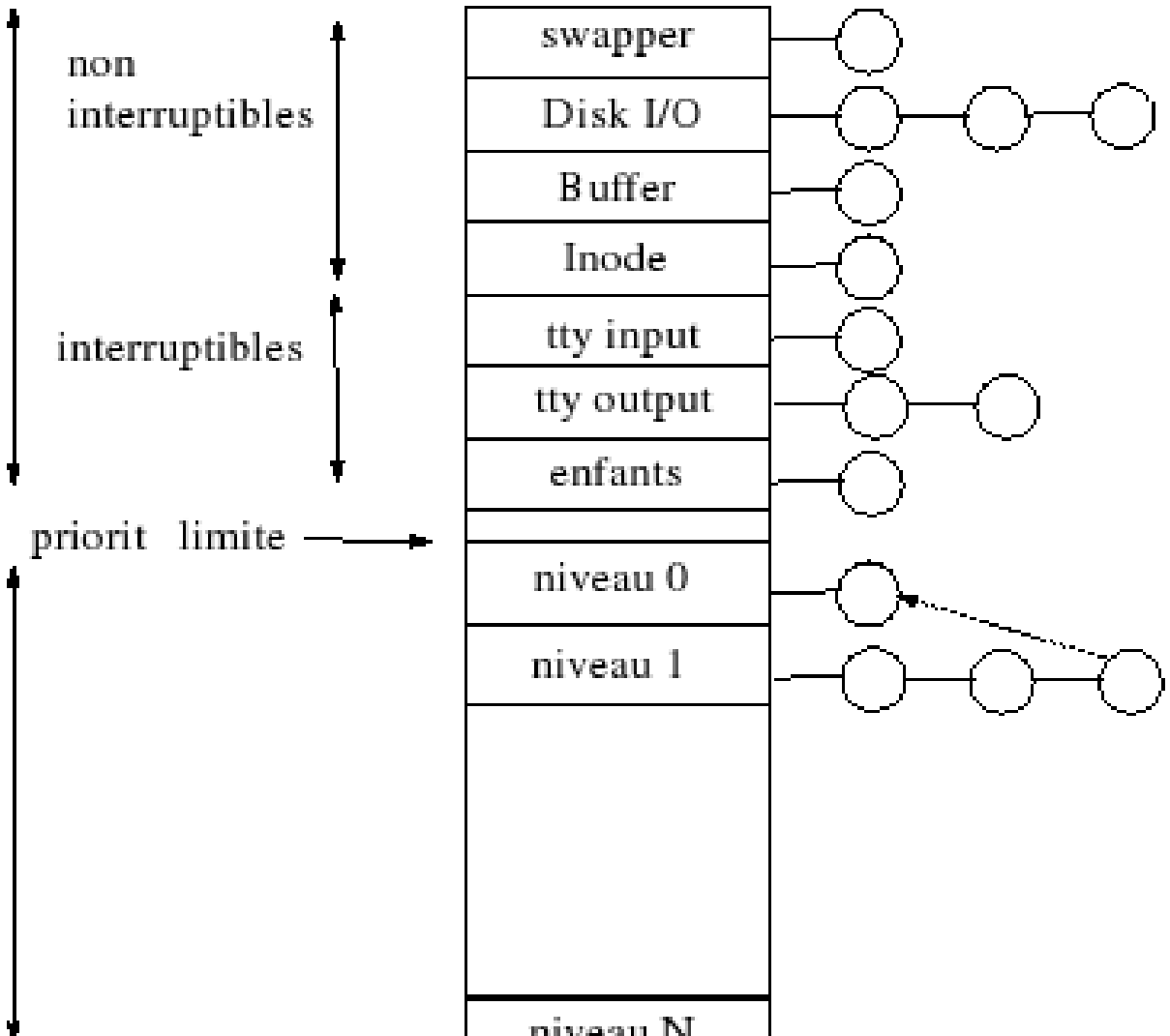
Multi-level-feedback round robin Queues

Unix

- Plusieurs file d'attente
- Matérialisation des niveaux de priorité
- Pour chaque niveau de priorité : Système de tourniquet

Niveaux de priorité

Niveaux de priorité



Niveaux de priorité

Niveaux de priorité

- Parcourt des listes une par une du haut vers le bas
- Stop dès que processus éligible trouvé
- Tant qu'il y a des processus "supérieur", attentes des autres
- Listes internes du noyau : Files d'attente utilisées
- Processus utilisateurs : même règle, mais Round-Robin utilisé
- Plus la priorité est haute, plus elle est faible

Évolution de la priorité

BSD

- Nécessité de faire évoluer les priorités
- Évolution réglée par trois équations
- P_{CPU} : temps passé sur le CPU
- P_{nice} : priorité utilisateur (entre -20 et 20)
- *load* : Charge moyenne du processus

Évolution de la priorité

BSD

- $$P_{pri} = PUSER + \frac{P_{CPU}}{4} + 2 \times P_{nice}$$
- Permet de diminuer linéairement la priorité en fonction de l'utilisation du processeur

Évolution de la priorité

BSD

- $$P_{CPU} = \frac{2 \times load}{2 \times load + 1} \times P_{CPU} + P_{nice}$$
- Pour éviter que la consommation de CPU soit trop pénalisante

Évolution de la priorité

BSD

- $$P_{CPU} = (\frac{2 \times load}{2 \times load + 1})^{sleep\_time} \times P_{CPU}$$
- Pour permettre au processus qui sommeille de pouvoir retrouver de la ressource