Systèmes d'Exploitation
Ordonnancement des processus

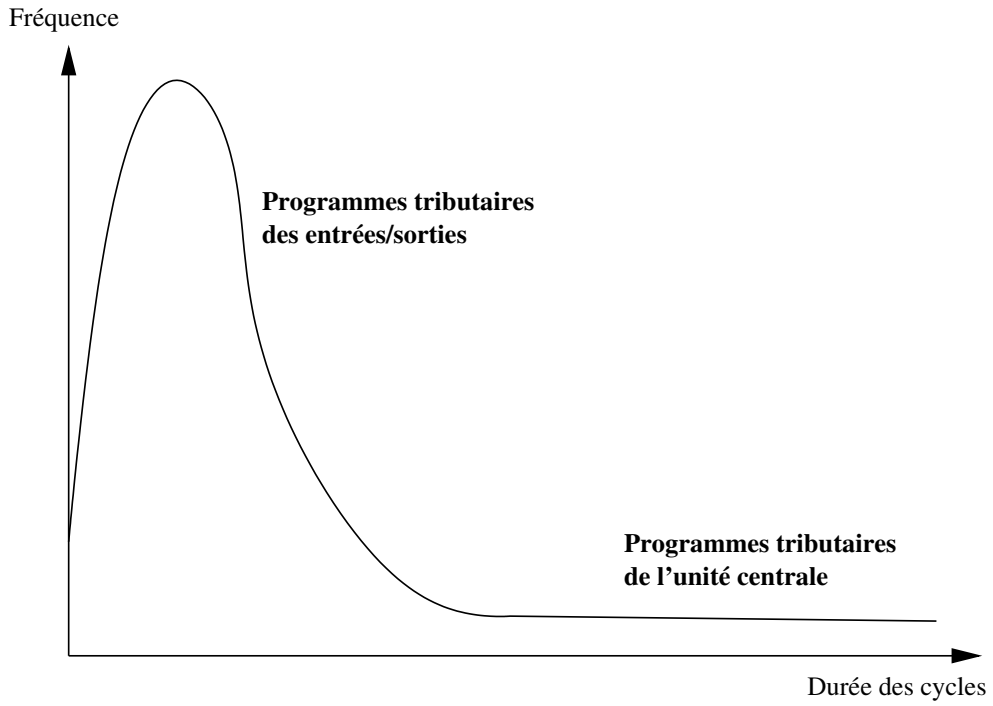
Didier Verna
didier@lrde.epita.fr
http://www.lrde.epita.fr/~didier

Version 2@1.5 – 7 décembre 2004

Table des matières

Généralités

Idée de base : effectuer une commutation dès que le processus en exécution doit attendre.



$\Longrightarrow$ La commutation a un coût : « latence de dispatching »

Ordonnancement et Réquisition

Motifs de commutation :

- **Aucun choix** : blocage ou terminaison d'un processus.
- **Choix** : arrivée d'un nouveau processus, passage des états actif ou bloqué à l'état prêt.

- **Ordonnancement sans réquisition** (système coopératif) : ne gère que le premier type de commutation. Windows 3.1.
- **Ordonnancement avec réquisition** (système préemptif) : gère tous les motifs de commutation. Nécessite des outils de synchronisation et du matériel spécifique (horloge). Windows 95, UNIX.

Critères d'ordonnancement

- **Tous systèmes :**
  - *Équité :* répartition du CPU).
  - *Respect de politique :* imposer les choix d'ordonnancement.
  - *Équilibre :* occupation de toutes les parties du système.
- **Batch :**
  - *Capacité de traitement / rendement :* nombre de processus exécutés par unité de temps.
  - *Temps de restitution / service :* délai entre la soumission d'un processus et sa terminaison (mise en mémoire, attente en état prêt, attente E/S, exécution).
  - *Utilisation du processeur*.

Critères d'ordonnancement (suite)

- **Interactifs :**
  - *Temps de réponse :* délai entre la soumission et le moment où l'on commence à répondre.
  - *Temps d'attente :* temps passé en état prêt.
  - *Proportionnalité :* aux attentes des utilisateurs. . .
- **Temps-réel :**
  - *Respect des dates limites :* éviter la perte de données.
  - *Prédictibilité :* stabilité des applications multimédia.

$\Longrightarrow$ Optimisation min, max, moyenne, variance. . .

Algorithmes d'ordonnancement

Premier arrivé, premier servi (FCFS)

- Algorithme sans réquisition.
- File d'attente FIFO pour les processus prêts.
- Facile à comprendre et à programmer.
- Intrinsèquement équitable pour des processus équivalents.

Mais :

- Grande variance des critères d'ordonnancement.
- Effet d'accumulation.

$\Longrightarrow$ Mauvais algorithme pour les systèmes en temps partagé.
OK pour les systèmes de batch.

Plus court d'abord / ensuite (SJF / N)

- Algorithme sans réquisition.
- Le prochain cycle le plus court est sélectionné.
- En cas d'égalité, on revient au FCFS.
- Version avec réquisition : « temps restant le plus court » (SRTF).
- Temps moyen d'attente minimal (quand tous les processus sont présents).

Mais :

- Difficulté de calculer la longueur des cycles.
- Approximation possible par moyenne exponentielle : $\tau_{n+1} = \alpha\tau_n + (1 - \alpha)\tau_{n-1}$

$\Longrightarrow$ Peu adapté pour l'ordonnancement à court terme.
OK pour les systèmes de batch.

Ordonnancement avec priorités

- Généralisation du SJF (priorité = inverse de la longueur du prochain cycle).
- Algorithme avec ou sans réquisition.
- Priorités internes (consommation de ressources. . .).
- Priorités externes (fixées par l'utilisateur).

Problème majeur de l'ordonnancement avec priorités : « famine » (blocage infini)

$\Longrightarrow$ technique du « vieillissement » (augmentation progressive de la priorité des processus en attente).

Tourniquet (RR)

Conçu spécialement pour le temps partagé

- FCFS avec réquisition sur une base de quantums (20 – 50ms).
- Nécessite une horloge.
- Le quantum doit être grand par rapport au temps de commutation.
- Le quantum ne doit pas être trop grand.
  (dégénérescence en FCFS)

$\Longrightarrow$ Réquisition pour les cycles plus longs que le quantum, commutation passive pour les cycles plus courts.

Files d'attente à plusieurs niveaux

- Découpage de la file d'attente des processus prêts en plusieurs files (processus système, interactifs, arrière-plan. . .).
- Ordonnancement spécifique au sein de chaque file (RR, FCFS).
- Ordonnancement des files entre elles (priorités fixes, allocation de tranches de temps. . .).

Ordonnancement avec feedback (recyclage) : possibilité de déplacer les processus d'une file d'attente à l'autre.

- implémentation du vieillissement.
- dégradation des priorités (ex. cycles longs. . .).

Ordonnancement par loterie

- Distribution de tickets (CPU, mémoire. . .).
- Tirage du gagnant à intervalle fixe.

- Implémentation légère d'un mécanisme de « promesse ».
- Efficace pour des processus coopératifs (transmission de tickets).
- Les processus importants peuvent obtenir plusieurs tickets.

Ordonnancement temps réel

Rappel : respecter les contraintes de temps

Types d'événements :

- **Périodiques :** distribution vidéo, chaîne industrielle...
- **Apériodiques :** monitoring hospitalier, contrôleur de bord...

Systèmes « ordonnançables » : Soient N événements périodiques de période P_i, nécessitant C_i temps CPU pour s'exécuter. Le système est dit ordonnançable si et seulement si :

$$\sum_{i=1}^{N} \frac{C_i}{P_i} \leq 1$$

Ordonnancement temps réel (suite)

- **Systèmes temps réel rigides** : faire de la « réservation de ressource ». L'ordonnanceur doit connaître exactement les échéances de chaque processus, et les ressources nécessaires.
- **Systèmes temps réel souples** : fournir des priorités hautes et non dégradables, minimiser la latence de dispatching.

Minimisation de la latence de dispatching : points de réquisition, réquisition des appels systèmes, ou plus généralement de tout le noyau (Solaris 2).
$\Longrightarrow$ Nécessité de synchroniser les accès au noyau (section critique).

- **Inversion des priorités** : un processus prioritaire attends des ressources noyau prises par un (des) processus non prioritaire(s).
- **Solution** : héritage des priorités (accès noyau en priorité haute).