Algorithme d'ordonnancement de tâches et de gestion de ressources

niveau : MPI

Motivation Les applications les plus répandues de l'informatique mettant toutes en œuvre des solutions pour permettre l'exécution de tâches concurrentes. Les approches sont variées, et il y a rarement de solution sans compromis.

I. Objectifs

déf 1. Une ressource est un composant matériel utilisé par un agent (logiciel) pour réaliser une tâche. ex: mémoire, processeur, carte réseaux, périphérique.
Une ressource peut être exclusive ou partagée.

déf 2. On parle d'exécution concurrente lorsque plusieurs agents utilisent simultanément une même ressource.

Plusieurs propriétés sont désirables :
- équité : les tâches ne sont pas involontairement désavantagées les unes par rapport aux autres ;

- vivacité : les ressources sont pleinement utilisées ;
- progrès : aucune tâche n'est bloquée indéfiniment.

Il s'agit donc de mettre en place des algorithmes d'ordonnancement mettant en place une politique d'accès aux ressources vérifiant les propriétés énoncées.

II. Ordonnancement (de tâches)

Dans le cadre d'un système, le processeur est une ressource allouée aux processus par le noyau. Cette allocation est appelée ordonnancement.
Les approches se distinguant selon les cas d'usage ; d'une part, la politique d'ordonnancement peut être :
- coopérative : les tâches travaillent à souhait puis laisse leur place (yield), ou bloque ; ou
- préemptive : le système reprend la main de lui-même.

D'autre part, il est possible de les classifier comme suit.

1. Ordonnancement par lot (batch scheduling)

déf 3 Une politique d'ordonnancement est dite

par lot lorsqu'elle est conçue pour être appliquée à un ensemble de tâches prédéterminé.

prop 4 (Plus courte tâche d'abord (SJF)) :
Pour un ensemble $\{J_0, \dots, J_m\}$ de tâches, de durées $(d_i)_1^n$, l'ordonnancement $[J_{i_0}, \dots, J_{i_m}]$ qui trie par durée de tâche croissante est optimal au sens où il minimise la somme des attentes de chaque tâche.

prop 5 (Extension au cas à plusieurs coeurs)
Une approche similaire reste optimale dans le cas multicoeur. DEV. 1

prop 6. (Ordonnancement avec pénalité)
Soit I un ensemble de tâches de même durées, avec $d : I \to \mathbb{N}$ leurs dates d'échéance et $r : I \to \mathbb{N} \cup \{+\infty\}$ leurs pénalités. Alors, l'algorithme glouton qui minimise à chaque étape la pénalité est optimal.

preuve : On peut associer à cette construction un matroïde. [3]

!⚙ Les algorithmes d'ordonnancement peuvent utiliser des mesures de coût/optimalité incomparables (sommes des attentes, nombre de tâches en retard, ...) qui dépendent des informations connues.

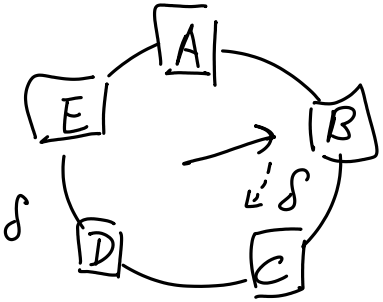
2. Systèmes interactifs

def. 7 Une politique d'ordonnancement sur un système interactif s'occupe d'un ensemble variable de tâches indépendantes.

Cette configuration se rencontre naturellement sur des serveurs ou des ordinateurs personnels.

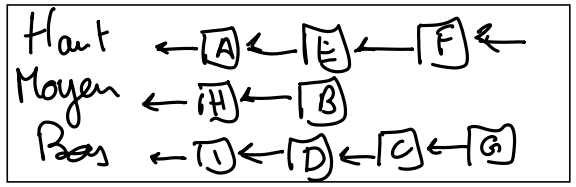
!⚙ Les informations dont on dispose sur les tâches diffèrent en fonction du cas d'usage : certains tâches n'ont pas de fin ; les pénalités/priorités sont tacites.
→ on ne traite pas formellement de l'efficacité des algorithmes suivants

a. Par quanta
Les tâches sont ordonnées et ont droit chacun leur tour à (au plus) un temps δ



[4]

b. Par priorité



Le prochain processus est choisi dans la file non-vide la plus haute. L'approche peut se cumuler avec un quanta, ou être coopérative, etc.

c. Heuristique SJF

On estime et actualise le temps pris par un processus, pour appliquer SJF ensuite: s'il prend un temps $T_0, T_1, \ldots$ à chaque exécution, une estimation peut être T_0, puis $\frac{1}{2}T_0 + T_1$, puis $\frac{1}{4}T_0 + \frac{1}{2}T_1 + T_2, \ldots$.

III Mémoire : Concurrence & gestion

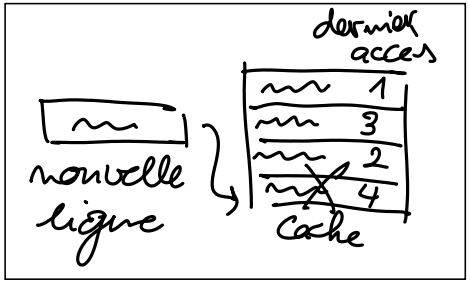
De nombreuses autres ressources s'exploitent avec des idées similaires : l'accès à une carte réseau, à une base de données,... mobilisent souvent le concept de quanta, de files (à priorité ou non). La mémoire, elle, n'est pas uniquement soumise à cette problématique.

1. Aspects concurrents

En l'absence de place libre dans l'ensemble des

pages (pour le système) ou des lignes (pour un cache), il faut pouvoir prendre la décision de la portion à rejeter (éventuellement sur le disque) pour mettre en place la nouvelle donnée.

La stratégie LRU ("utilisée en dernier") consiste à évincer la donnée qui a été lue il y a le plus longtemps.



2. Nettoyer et partager

Le principe d'équité se traduit également au niveau utilisateur. Gérer sa mémoire de manière économe permet une utilisation plus efficace de la mémoire entre fils d'exécution par exemple.

Cet aspect se rencontre :

- dans les langages à allocation explicit (ex. C avec malloc), qui nécessitent une structure de données adéquate.
- dans les langages avec garbage collector, comme OCaml DEV. 2, qui fournissent un compromis intéressant.