

Chapitre 5

Ordonnancement de processus

L'ordonnanceur (scheduler) définit l'ordre dans lequel les processus prêts utilisent l'UC (en acquièrent la ressource) et la durée d'utilisation, en utilisant un algorithme d'ordonnancement. Un bon algorithme d'ordonnancement doit posséder les qualités suivantes :

- équitabilité : chaque processus reçoit sa part du temps processeur
- efficacité : le processeur doit travailler à 100% du temps
- temps de réponse : à minimiser en mode interactif
- temps d'exécution : minimiser l'attente des travaux en traitement par lots (batch)
- rendement : maximiser le nombre de travaux effectués par unité de temps

L'ensemble de ces objectifs est contradictoire, par exemple le 3ème et le 4ème objectif.

On appelle temps de traitement moyen d'un système de tâches la moyenne des intervalles de temps séparant la soumission d'un tâches de sa fin d'exécution. On appelle assignation la description de l'exécution des tâches sur le ou les processeurs. A chaque tâche T_i du système de tâches, on associe deux réels :

- t_i : sa date d'arrivée
- τ_i : sa durée

Il existe deux familles d'algorithmes :

- sans réquisition (ASR) : le choix d'un nouveau processus ne se fait que sur blocage ou terminaison du processus courant (utilisé en batch par exemple)
- avec réquisition (AAR) : à intervalle régulier, l'ordonnanceur reprend la main et élit un nouveau processus actif.

5.1 ASR

5.1.1 Suivant l'ordre d'arrivée

On utilise l'ordonnanceur dans l'ordre d'arrivée en gérant une file de processus. Cet algorithme est facile à implanter, mais il est loin d'optimiser le temps de traitement moyen.

5.1.2 PCTE (Plus Court Temps d'Exécution)

On utilise l'ordonnanceur par ordre inverse du temps d'exécution (PCTE) : lorsque plusieurs travaux d'égale importance se trouvent dans une file, l'ordonnanceur élit le plus court d'abord. On démontre que le temps moyen d'attente est minimal par rapport à toute autre stratégie si toutes les tâches sont présentes dans la file d'attente au moment où débute l'assignation.

5.2 Ordonnancement circulaire ou tourniquet (round robin)

Il s'agit d'un algorithme ancien, simple et fiable. Le processeur gère une liste circulaire de processus. Chaque processus dispose d'un quantum de temps pendant lequel il est autorisé à s'exécuter. Si le processus actif se bloque ou s'achève avant la fin de son quantum, le processeur est immédiatement alloué à un autre processus. Si le quantum s'achève avant la fin du processus, le processeur est alloué au processus suivant dans la liste et le processus précédent se trouve ainsi en queue de liste.

La commutation de processus (overhead) dure un temps non nul pour la mise à jour des tables, la sauvegarde des registres. Un quantum trop petit provoque trop de commutations de processus et abaisse l'efficacité du processeur. Un quantum trop grand augmente le temps de réponse en mode interactif. On utilise souvent un quantum de l'ordre de 100 ms.

5.3 Ordonnancement PCTER

Il s'agit d'une généralisation avec réquisition de l'algorithme PCTE : à la fin de chaque quantum, on élit la tâche dont le temps d'exécution restant est minimal (PCTER = Plus Court Temps d'Exécution Restant). Cet algorithme fournit la valeur optimale du temps moyen de traitement pour les algorithmes avec réquisition.

5.4 Ordonnancement avec priorité

Le modèle du tourniquet suppose tous les processus d'égale importance. C'est irréaliste. D'où l'attribution de priorité à chaque processus. L'ordonnanceur lance le processus prêt de priorité la plus élevée. Pour empêcher le processus de priorité la plus élevée d'accaparer le processeur, l'ordonnanceur abaisse à chaque interruption d'horloge la priorité du processus actif. Si sa priorité devient inférieure à celle du deuxième processus de plus haute priorité, la commutation a lieu.

A chaque niveau de priorité, on associe une liste (classe) de processus exécutables, gérée par l'algorithme du tourniquet. Par exemple, supposons 4 niveaux de priorité. On applique à la liste de priorité 4 l'algorithme du tourniquet. Puis, quand cette liste est vide, on applique l'algorithme à la liste de priorité 3, etc... S'il n'y avait pas une évolution dynamique des priorités, les processus faiblement prioritaires ne seraient jamais élus : risque de famine.

5.5 Ordonnancement avec files multiples

On associe une file d'attente et un algorithme d'ordonnancement à chaque niveau de priorité. Si les priorités évoluent dynamiquement, le système d'exploitation doit organiser la remontée des tâches.

5.6 Ordonnancement par une politique

S'il y a n utilisateurs connectés, on peut garantir à chacun qu'il disposera de $\frac{1}{n}$ de la puissance du processeur. Le système d'exploitation mémorise combien chaque utilisateur a consommé de temps processeur et il calcule le rapport : *temps processeur consommé / temps processeur auquel on a droit*. On élit le processus qui offre le rapport le plus faible jusqu'à ce que son rapport cesse d'être le plus faible.

5.7 Ordonnancement à deux niveaux

La taille de la mémoire centrale de l'ordinateur peut être insuffisante pour contenir tous les processus prêts à être exécutés. Certains sont contraints de résider sur disque.

Un ordonnanceur de bas niveau (CPU scheduler) applique l'un des algorithmes précédents aux processus résidant en mémoire centrale. Un ordonnanceur de haut niveau (medium term scheduler) retire de la mémoire les processus qui y sont restés assez longtemps et transfère en mémoire des processus résidant sur le disque. Il peut exister un ordonnanceur à long terme (job scheduler) qui détermine si un processus utilisateur qui le demande peut effectivement entrer dans le système (si les temps de réponse se dégradent, on peut différer cette entrée).

L'ordonnanceur de haut niveau prend en compte les points suivants :

- depuis combien de temps le processus séjourne-t-il en mémoire ou sur le disque ?
- combien de temps processeur le processus a-t-il eu récemment ?
- quelle est la priorité du processus ?
- quelle est la taille du processus ? (s'il est petit, on le logera sans problème)

5.8 Ordonnanceur de chaînes de tâches

Supposons que l'on ait r chaînes de tâches à exécuter en parallèle : C_1, C_2, \dots, C_r avec $C_i = T_{i1}T_{i2}T_{i3} \dots T_{ik_i}$, c'est-à-dire que la chaîne C_1 comprend k_1 tâches, la chaîne C_2 comprend k_2 tâches, ... On note : $k_1 + k_2 + \dots + k_r = n$ (nombre total de tâches). On note τ_{ij} la durée de la tâche T_{ij} .

Il existe un algorithme produisant une assignation de temps moyen de traitement minimal :

pour chaque chaîne C_i , calculer la durée moyenne d'exécution (pas de traitement) ϵ_{ip} de chaque sous-chaîne C'_{ip} , pour $p = 1, \dots, k_i$

tant qu'il existe des tâches non assignées

déterminer j tel que : $\epsilon_{jp_j} = \min\{\epsilon_{mp_m} | m = 1, 2, \dots, n\}$

ajouter à l'assignation les tâches de la sous-chaîne C'_{jp_j} et les supprimer de la chaîne C_j

recalculer ϵ_{jp_j} pour la nouvelle chaîne C_j

fin tant que

Exemple :

$C_1 : \tau_{11} = 9, \tau_{12} = 6, \tau_{13} = 3, \tau_{14} = 5, \tau_{15} = 11$

$C_2 : \tau_{21} = 4, \tau_{22} = 1, \tau_{23} = 13$

$C_3 : \tau_{31} = 8, \tau_{32} = 4, \tau_{33} = 1, \tau_{34} = 11$

On détermine le minimum : $\tau_{22} = 2.5$, donc : l'assignation T_{21}, T_{22} puis $\tau_{33} = 4.33$; on complète l'assignation avec $T_{31}, T_{32}, T_{33}, \dots$

D'où l'assignation suivante :

$T_{21}, T_{22}, T_{31}, T_{32}, T_{33}, T_{11}, T_{12}, T_{13}, T_{14}, T_{15}, T_{34}, T_{23}$