1 Introduction

Je suis nul en intro donc je propose de te laisser faire cette partie.

2 Ordonnancement

2.1 Traitement par lots

2.1.1 Premier arrivé, premier servi (FCFS)

Cet algorithme d'ordonancement est un des plus simple qui existe autant pour ce qui est de le comprendre que de le programmer. Essentiellement, cet algorithme garde une file dans laquelle les processus sont rangés en attendant d'être exécutés dans leur ordre d'arrivé. Il n'y a aucun autre facteur qui est pris en compte pour donner la priorité aux processus. Le plus grand avantage du FCFS est le fait que l'algorithme garanti qu'il n'y aura jamais de famine. En contrepartie, le temps d'attente des processus est loin d'être optimal, car des processus qui prennent beaucoup de temps à exécuter vont être exécuter en entier même s'il y a plusieurs processus très cours qui se trouvent derière dans la file.

Par exemple, si on a 3 processus qui entrent dans la file un après l'autre : $F_1 = 18$ sec, $F_2 = 6$ sec et $F_3 = 3$ sec. Le temps d'attente moyen sera de (0 + 18 + 24) / 3 = 14 sec. Ce temps d'attente est loin d'être le meilleur et exécuter les processus dans un ordre différent le réduirait drastiquement. Dans ce cas-ci, l'ordre optimal serait F_3, F_2, F_1 ce qui ferait un temps d'attente moyen de (0 + 3 + 9) / 3 = 4 sec.

2.1.2 Plus court en premier

Cet algorithme va toujours exécuter la tâche la plus courte en premier comme son nom l'indique. Cela permet de règler le problème de temps d'attente du FCFS, mais vient avec le désavantage majeur qu'il peut causer de la famine s'il y a un processus qui prend beaucoup de temps à exécuter et un flot de processus plus courts qui vont toujours avoir la priorité sur ce processus dont le temps d'attente va continuer à croître.

Il existe aussi une autre variante de cet algorithme qui applique le même principe, mais avec le temps restant le plus court. La différence majeur est qu'avec le plus court en premier un processus sera exécuté en entier même s'il est plus long qu'un nouveau processus qui arrive. Cet variante de l'algoritme fait en sorte que si le temps restant du processus en cours est plus long que la nouvelle job qui arrive il sera interrompu et la priorité sera donnée au nouveau processus.

2.2 Interactifs

2.2.1 tourniquet (round robin)

L'ordonnancement de type tourniquet est un type qui est en général plus efficace et donc plus utilisé que les algorithmes décrits précédemment. Le principe de cet algorithme est que chaque processus se voit attribuer une période de temps qu'on peut appeler quantum pendant lequel il peut s'exécuter. Donc, chaque processus s'exécute pour le temps d'un quantum à tour de rôle. Évidemment, si la job se termine avant la fin du quantum on passe directement au prochain processus. L'avantage de ce type d'ordonnancement est le fait que nous pouvons être certain qu'il n'y aura pas de famine et, en plus, cela ne vient pas avec le désavantage du FCFS où un processus long s'exécutera en entier faisant attendre les autres processus pendant longtemps. Par contre, l'ordonnancement de type tourniquet n'est pas parfait et pose le problème d'un choix approprié pour la longueur d'un quantum. Idéalement, on voudrait pouvoir choisir un quantum très cours pour pouvoir éviter de faire attendre les processus trop longtemps s'ils sont nombreux à arriver en même temps. Par contre, un choix de quantum trop cours vient avec le problème majeur que le temps de changement de processus n'est pas nul et doit être pris en compte pour ne utiliser un pourcentage trop élevé des ressources sur cette tâche qui n'accompli rien.

En réalité, la longueur choisi pour un quantum varie entre 20-50 ms. Par exemple, si 40 requêtes arrivent dans un interval de temps court et que la durée du quantum est de 30ms nous pouvons être certain que la dernière requête se fera attribuer du temps d'exécution au maximum 1.2 secondes après son arrivée ce qui est très raisonnable. Par ailleurs, si le changement de processus prend environ 1ms, il y a une perte de performance de seulement 3.33% (1ms/30ms) ce qui, sans être négligable, n'est pas trop élevé.

2.2.2 ordonnancement par priorités

Cet algorithme s'utilise en donnant un niveau de priorité à chaque processus et en exécutant le processus qui a le plus haut niveau de priorité en premier. Sans aucun ajustement cet méthode causerait un problème de famine et seul les processus à priorité élevé se verraient attribuer du temps. Pour cette raison, le niveau de priorité d'un processus diminue avec le temps permetttant de laisser la place aux processus qui sont moins prioritaires.

2.3 Temps réel

3 Conclusion

Références

[1] Tanenbaum, Andrew. 2008. Systèmes d'exploitation. 3ème édition. France : Pearson Education France, 1052p.