info524 : Systèmes d'exploitation
TD 2 : ordonnancement et processus, suite

Pierre Hyvernat et Rodolphe Lepigre
Pierre.Hyvernat@univ-smb.fr
Rodolphe.Lepigre@univ-smb.fr

Exercice 1 : Processus et threads

Question 1. On essaie d'ordonnancer les processus suivants sur un système temps réel :
- 2 communications sonores, qui utilisent chacune 1 ms de processeur toutes les 5 ms,
- un flux vidéo à 25 trames par seconde, chaque trame ayant besoin de 20ms de processeur.

Peut-on ordonnancer ce système ?

Question 2. On considère deux processus qui nécessitent chacun 10 minutes de temps processeur pour faire leurs calculs. Chacun des processus est bloqué sur des entrées sorties en moyenne pendant 50% du temps du temps total d'éxécution.
- combien de temps dure l'exécution des deux processus, s'ils sont exécutés séquentiellement ?
- combien de temps dure l'exécution si les processus sont entrelacés ?

Donnez dans chaque cas le taux d'utilisation du processeur, si on ne compte que ces deux processus.

Question 3. Pour des threads utilisateurs, est-ce que chaque thread possède sa propre pile d'exécution ? Qu'en est-il pour les threads noyaux ?

Question 4. À votre avis, quand un processus multi-thread fait un `fork`, est-ce que tous les threads sont dupliqués pour le fils ?

Question 5. L'interface POSIX pour les processus légers (threads) contient l'instruction `sched_yield` qui permet à un thread d'abandonner le processeur au profit des autres threads. Un processus normal n'a aucun intérêt à faire ceci. Pourquoi est-ce que cette instruction existe pour les processus légers ? Donnez des exemples d'utilisation possible.

Exercice 2 : Ordonnanceur Linux (noyau 2.6)

L'ordonnanceur de Linux utilise des files de priorités et des quantums de temps différents pour les processus. (Les processus prioritaires ont un quantum de temps plus long.)

À chaque fois qu'un nouveau processus est devient prêt, il peut prendre la place du processus en cours d'éxecution s'il est plus prioritaire. Ceci se passe même si le processus en cours d'éxecution n'a pas terminé son quantum.

Question 1. Que faut il faire avec le processus courant lorsqu'il est préempté de cette manière
- le remettre en fin de file en réinitialisant son quantum,
- le remettre en tête de file en réinitialisant son quantum,
- le remettre en fin de file en conservant son quantum actuel,
- le remettre en tête de file en conservant son quantum actuel ?

Question 2. L'ordonnanceur de Linux met la priorité des processus à jours suivant qu'il s'agit de processus "batch" (beaucoup de calculs) ou de processus interactif (beaucoup d'entrés/sorties).

Un processus "batch" doit-il être considéré comme plus prioritaire qu'un processus interactif ?

Question 3. Lorsque le processus en execution a terminé son quantum de temps, que faut il faire
- le remettre en fin de file en réinitialisant son quantum,
- le remettre en tête de file en réinitialisant son quantum.

Question 4. L'ordonnanceur choisit en fait toujours un processus qui n'a pas atteint la fin de son quantum : on parle de processus "actif". C'est seulement lorsque ces processus actifs ont tous terminé leur quantum que les autres processus (appelés expirés) peuvent s'éxécuter.

Pour que les processus intéractifs n'attendent pas trop, ces processus restent "actifs" lorsque leur quantum se termine, alors que les processus batch deviennent automatiquement "expirés".

Quel problème cela pourrait il poser ?

Question 5. En fait, les processus interactifs peuvent devenir inactifs dans deux cas :
- un processus expiré est plus prioritaire que le processus interactif,
- il y a un processus expiré "suffisamment vieux".

Question 6. En plus des 40 priorités habituelles (de -20 à +19), le noyau linux gère également des processus temps réels.

Donnez des exemples de processus temps réels et discutez de leur priorité (plus ou moins prioritaire que les processus normaux), et de la durée de leur quantum.