info507 : Systèmes d'exploitation
TD 2 : ordonnancement

Pierre Hyvernat, François Boussion, Gérald Cavallini
Pierre.Hyvernat@univ-smb.fr
Francois.Boussion@univ-smb.fr
Gerald.Cavallini@univ-smb.fr

Exercice 1 : Ordonnancement

On considère les processus suivant :

processus	temps d'arrivée	durée
P_1	0	2
P_2	1	4
P_3	3	2
P_4	4	1

(Les processus arrivent "juste avant" leur temps d'arrivée.)

Question 1. On considère l'ordonnancement FIFO, sans préemption.
- Donnez l'ordre d'exécution des processus.
- Calculez le *temps processeur* moyen, le *temps de réponse* moyen et le *temps d'exécution* moyen.

Est-ce que cette politique d'ordonnancement est *juste* ? (càd, peut elle provoquer des famines ?)

Question 2. On considère maintenant l'ordonnancement "plus court temps d'exécution", avec préemption possible lors de l'arrivée de processus.
- Donnez l'ordre d'exécution des processus.
- Calculez le *temps de réponse* moyen et le *temps d'exécution* moyen.

Est-ce que cette politique d'ordonnancement est *juste* ?

Question 3. On considère maintenant l'ordonnancement "tourniquet" ("round-robin" en anglais) avec un quantum de 1 unité de temps.
- Donnez l'ordre d'exécution des processus.
- Calculez le *temps de réponse* moyen et le *temps d'exécution* moyen.

Est-ce que cette politique d'ordonnancement est *juste* ?

Exercice 2 : Multiprogrammation

Question 1. On considère deux processus nécessitant chacun 10 minutes de *temps processeur*. Sur une exécution complète (*temps utilisateur*), ces processus sont bloqué (entrées/sorties) pendant 50% du temps.
- combien de temps (utilisateur) dure l'exécution des deux processus, s'ils sont exécutés séquentiellement ?
- combien de temps (utilisateur) dure l'exécution si les processus sont entrelacés ?

Donnez dans chaque cas le taux d'utilisation du processeur (si on suppose que seuls ces deux processus sont présents).

Généralisez en considérant n processus.

Exercice 3 : processus légers ("threads")

Question 1. On essaie d'ordonnancer les tâches suivantes sur un système temps réel :
- 2 canaux sonores, qui utilisent chacun 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. 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.

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 ?

Exercice 4 : 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 typiquement un quantum de temps plus long.)

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

Question 1. Lorsque le processus en exécution 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 2. Que faut il faire avec le processus courant lorsqu'un processus plus prioritaire prend sa place avant la fin du quantum :
- 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 3. L'ordonnanceur de Linux met la priorité "interne" 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 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 (dit processus *expirés*) peuvent s'exécuter.

Pour que les processus interactifs 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 ?

Proposez une solution.