Hoël Jalmin, Mathieu Dumax-Vorzet, Iheb Mastoura

Compte rendu
TP Scheduler Nanvix

Les objectifs :

Pour ce TP, nous avons choisi d'implémenter les systèmes de scheduling suivant :

- Priority Scheduling (statique et dynamique)
- Lottery Scheduling

1) Priority Scheduling

a) Static Priority

Dans cette version statique, l'ordonnancement se fait par comparaison des priority, des nice puis des compteurs des processus prêts à s'exécuter. L'implémentation naïve de cette idée soulève 2 problèmes majeurs :

- un risque de famine, un processus avec des priorités très faible peut ne jamais avoir la main. Ce problème est inhérent à l'idée d'origine et sera résolu par la version dynamique de cet ordonnancement.
- *IDLE* a les priorités par défaut d'un processus (*USER* et *NZERO*) et peut donc passer devant des processus avec un nice inférieur. Pour remédier à cela, la valeur la plus faible possible de nice est réservée à *IDLE*. Cela engendre des modifications sur les fonctions fork, nice, au lancement de *IDLE* et enlève une valeur possible à nice.

b) Dynamic Priority

Nous avons par la suite tenté d'améliorer la gestion des priorités en introduisant un système de vieillissement des processus, afin que les processus à priorité basse mais ayant attendu très longtemps puissent s'exécuter avant des processus à priorité plus élevée, évitant ainsi les famines

Pour ceci, nous calculons pour chaque processus une priorité personnalisée, basée sur la formule suivante:
3*priority + 2*nice - counter

L'idée est de garder la priorité comme champ plus important, mais également de prendre en compte le champ nice et et le vieillissement du processus.

Les changements apportés au code sont assez minimes, il suffit de modifier légèrement **sched.c**.

2) Lottery Scheduling

L'ordonnancement par loterie est un ordonnancement sans famine (car chaque processus a une probabilité non nulle d'être sélectionné) où un certain nombre de tickets de loterie est attribué aux processus. L'ordonnanceur tire un ticket de façon aléatoire pour sélectionner le processus à exécuter parmie les processus prêts. La distribution des tickets dépend de la priorité des processus, on accorde plus de tickets à un processus plus prioritaire pour lui conférer une chance plus élevée de sélection.

On crée un tableau de pid global afin de retrouver le processus associé au numéro de ticket tiré. On pourrait également gérer cela avec un tableau propre à chaque processus mais la synchronisation des tables est complexe à maintenir.
À sa création, un processus se voit attribuer un nombre de tickets et les gardera jusqu'à sa mort, même s'il est endormi ou non-prêt. Si on tire le numéro d'un processus non-prêt, on laisse *IDLE* être sélectionné, ce qui revient à relancer le tirage.

On a créé les fonctions suivantes, placées dans **sched.c** et déclarées dans **pm.h** :

- **num tickets**(int priority) renvoie le nombre de tickets à allouer à un processus selon ses champs priority et nice
- **add_tickets**(struct process* proc) : Alloue un nombre déterministe de tickets à un processus. On alloue aucun ticket au processus *IDLE*.
- **rm_tickets**(struct process* proc) : Retire les tickets d'un processus, et regrouper le reste des tickets au début du tableau pour pouvoir facilement faire un tirage aléatoire dessus.

On a également modifié les 2 fichiers **fork.c** et **die.c.** Nous utilisons nos fonctions afin d'assigner des tickets aux processus à leur création (**add_tickets**) et les retirer à la destruction des processus (**rm_tickets**).

Remarque :

Le tableau contenant les tickets a une taille très importante, ce qui pourrait poser problème si un processus utilise beaucoup de ressources. Pour limiter cela, on pourrait améliorer la structure de stockage des tickets en utilisant un tableau d'occurrence. Cette amélioration n'a pas été implémentée par manque de temps mais pourra l'être par la suite.