TP - ASR7 Programmation Concurrente

—

Synchronisation et Ordonnancement

Matthieu Moy, Grégoire Pichon, Sylvain Brandel, Guillaume Damiand,
Laurent Lefevre, Frédéric Suter

Printemps 2021

I Introduction

Édition spéciale coronavirus : pas de présentiel cette année. Vous avez reçu les consignes par email pour poser vos questions par messagerie instantanée. Connectez-vous à la messagerie avant de démarrer le TP.

Pour ce TP, nous allons jouer avec les politiques d'ordonnancement temps-réel du noyau Linux (`SCHED_FIFO`, `SCHED_RR`). Activer ces politiques est potentiellement dangereux pour le système (une boucle infinie en priorité temps-réel peut figer l'ensemble du système), donc interdit pour des simples utilisateurs : vous aurez besoin de passer root pour le faire. Vous ne pourrez donc pas faire ce TP sur les machines physiques de l'université.

Nous allons utiliser l'infrastructure de *cloud computing* du département financée par la région Rhône-Alpes. C'est un ensemble de machines pilotées par le logiciel `OpenStack` avec lequel vous allez avoir un premier contact. Vous allez l'utiliser afin de créer une machine virtuelle, que vous pourrez conserver, effacer (mais perdre ainsi toutes les configurations effectuées), re-générer... La machine virtuelle (VM) créée va ici nous servir de base d'expérimentation, et dans le TP suivant d'entraînement à des manipulations d'administration systèmes que vous pourrez ensuite effectuer sur vos machines personnelles.

II Modification de l'ordonanceur

Dans cet exercice, vous allez utiliser explicitement l'ordonnanceur du noyau Linux. C'est assez dangereux car un processus prioritaire qui ne s'arrête pas, par définition, bloque l'ordinateur. Vous devez donc utiliser les machines virtuelles.

Le code `sched.cpp` a été préparé, il lance un certains nombre de threads qui font des calculs en affichant de temps en temps des informations. Pour le moment, le code de la fonction `change_ordonnancement()` n'est pas fait et l'ordonnancement n'est pas modifié.

Q.II.1) - Complétez le code de cette fonction pour qu'elle change l'ordonnancement du thread qui l'appelle.

Il n'y a pas de fonctions C++ définies dans la bibliothèque standard pour manipuler l'ordonnanceur. Vous allez devoir utiliser les fonctions C suivantes pour faire ce travail :

— Retourne le numéro du thread qui l'appelle :

```
1    pthread_self();
```

— Récupérer les paramètres d'ordonnancement courant (elle vous permettra d'initialiser les variables) :

```
1    int pthread_getschedparam(pthread_t target_thread,
2                              int *politique,
3                              struct sched_param *param);
```

— modifier la politique d'ordonnancement et ses paramètres :

```
1    int pthread_setschedparam(pthread_t target_thread,
2                              int politique,
3                              const struct sched_param *param);
```

La valeur de la politique est définie dans des macros : `SCHED_FIFO`, `SCHED_RR` ou `SCHED_OTHER`.
Faites quelques expériences :

Q.II.2) - Lancez 3 threads RR, l'un de priorité 4 et deux autres de priorité 2.

Q.II.3) - Lancez 3 threads FIFO de priorité identique.

Q.II.4) - Lancez 1 threads FIFO et 2 RR de priorité identique.

Q.II.5) - Lancez 1 thread FIFO de priorité 99 et 2 autres FIFO de priorité plus faibles.

Q.II.6) - Pouvez-vous stopper le programme pendant que des threads très prioritaires tournent ? Pourquoi [1] ?

Q.II.7) - Si vous avez votre propre ordinateur, refaites les expériences sur ce dernier. Avez-vous les mêmes résultats ? Pourquoi ?

III Si le temps le permet, Intégration par la méthode des rectangles

III.1 Théorie

Les sommes de Riemann sont des sommes approximant des intégrales. Ainsi, si on considère $f : [a, b] \to \mathbb{R}$ une fonction partout définie sur le segment $[a, b]$; un entier $n > 0$ et une subdivision *régulière* définie pour $0 \leq k \leq n$ par :

$$x_k = a + k\frac{b-a}{n}$$

Alors la somme de Riemann est définie comme

$$S_n = \frac{b-a}{n} \sum_{k=1}^{n} f(a + k\frac{b-a}{n})$$

c-à-d

$$S_n = \sum_{k=1}^{n} (x_k - x_{k-1}) f(x_k)$$

Plus n est grand, plus [2] cette méthode des rectangles pour le calcul des intégrales donne une estimation proche de la valeur cherchée de l'intégrale

1. Pour vous inspirer, vous pouvez regarder les valeurs des 2 variables du noyau `/proc/sys/kernel/sched_rt_runtime_us` et `/proc/sys/kernel/sched_rt_period_us`
2. moyennant la prise en compte des erreurs d'arrondi latentes...

III.2 Pratique

Nous avons accès à des machines multi-cœur, et voulons en faire bon usage pour calculer des résultats d'intégration.

Q.III.1) - Avec ce qui a été vu en cours, vous devez proposer un programme qui sera capable de s'adapter au nombre de cœurs fourni par l'utilisateur en ligne de commande (de sorte que chacun soit utilisé pour calculer une partie d'intégrale). Le programme codé en C++ retournera la valeur de la somme totale calculée.
Vous pourrez utiliser des fontions à intégrer plus ou moins complexes pour vous assurer de la validité de votre code, et pour les questions suivantes.

Q.III.2) - Donnez 2 moyens de savoir combien de cœurs dispose la machine que vous utilisez actuellement.

Q.III.3) - À l'aide de la fonction `gettimeofday()`, vous mesurerez la durée du programme pour un nombre de cœurs valant 1, 2, 4, 8 et 64.
Commentez les résultats obtenus !

Q.III.4) - Vous pouvez trouver une solution en C utilisant OpenMP à l'URL `http://graal.ens-lyon.fr/~ycaniou/Teaching/1415/L3/integrale_OpenMP.c`. Quelles observations pouvez-vous faire ?