Lionel Beltrando TP3

Compte-rendu du TP 3

Caractéristiques de la machine et système d'exploitation :

Tout d'abord je fournis les caractéristiques de ma machine sur laquelle j'ai fais tourné les programmes. (Informations obtenus avec **cat /proc/cpuinfo**) :
Machine multiprocesseurs : 4 processeurs de même caractéristiques :

-Modèle : Intel Core i5-4200H
-Fréquence 2.80 Ghz
- Cpu MHz 1100 MHz
-Taille du cache : 3072 KB
- Nombre de cœurs par processeur : 2
- Nombre de threads par coeur : 2 (donc 4 thread par processeur)
- Adresses physiques sur 39 bits
- Adresses logiques sur 48 bits

J'utilise comme système **Ubuntu 14.04 LTS**.

1. Étapes pour la création/destruction

1.1 Quel outil pour quelle mesure ?

Si l'on dispose d'un mètre avec seulement des graduations en centimètres et que l'on souhaite mesurer l'épaisseur d'une feuille issue d'une ramette, on va mesurer l'épaisseur d'un paquet de feuille de tel sorte que ce paquet soit mesurable et on divise par le nombre de feuille dans le paquet pour avoir l'épaisseur d'une seule feuille. Par exemple on pourrait prendre un paquet d'un centimètre. Cependant pour un paquet d'un centimètre on a une grande incertitude sur la mesure (mesure t-il 1cm, 1,1 cm, 0,9 cm ?). Cette incertitude est liée en partie à la performance de l'œil humain, il est donc mieux de prendre un paquet de plus grande taille encore (1 mètre ou au moins 10 cm), ainsi l'incertitude sur la mesure du paquet reste la même mais comme on mesure l'épaisseur de plus de feuilles, l'incertitude sur l'épaisseur d'une feuille est plus petite.

Dans le TP, on mesure des temps pour effectuer une opération (par exemple la création et attente de la terminaison d'une thread). J'ai donc choisi d'effectuer à chaque fois les opérations un grand nombre de fois (en pratique je les réalise 1 000 000 de fois), je mesure le temps juste avant d'effectuer les opérations et juste après ce qui me donne le temps total, que je divise par le nombre de fois que j'ai réalisé cette opération.

Cependant contrairement à l'exemple de la feuille de papier il y a d'autres facteurs qui entrent en compte ici. Le problème (qui n'en est pas forcément un à mon avis) est que dans tous les cas je n'effectue pas que les opérations que je veux (typiquement je place mes opérations à effectuer dans une boucle for, donc en plus des opérations à effectuer il y a les opérations de la boucle for). Pour remédier à ce problème on peut enlever au temps final, le temps que met à s'exécuter une boucle for qui ne fait rien.

Une autre solution (que je n'ai pas implémenté) que j'ai vu chez quelques camarades est de mesurer uniquement l'opération à exécuter, ainsi on ne mesure que l'action à effectuer mais on augmente l'incertitude sur la mesure (problème de la feuille de papier). Je pense donc que cette solution n'est pas adaptée.

1.2 Question optionnelle :

- **clock()** : C'est une fonction de la **bibliothèque <time.h>**
- **clock_gettime()** : C'est une fonction de la **bibliothèque <time.h>**, la fonction donne accès à plusieurs horloges et donne un résultat en secondes et nanosecondes.
La fonction peut donne des résultats différents selon l'horloge qu'on lui fournit en argument.
J'ai trouvé quatre horloges :
    - CLOCK_REALTIME : Fournit un **temps réel**. On peut initialiser cette horloge si on possède les privilèges requis. (On ne peut donc pas utiliser cette horloge si ce que l'on mesure implique un redémarrage du système qui peut modifier cette horloge).
    - CLOCK_MONOTONIC : Fournit un **temps réel**. Cette horloge ne peut pas être initialisée et elle compte le temps depuis un point de départ non spécifié.
    - CLOCK_PROCESS_CPUTIME_ID : Fournit un t**emps cpu** adapté à une mesure de temps concernant des processus.
    - CLOCK_THREAD_CPUTIME_ID : Fournit un **temps cpu** adapté à une mesure de temps concernant des threads.

- **gettimeofday()** : C'est un **appel système** qui renvoie en secondes et en microsecondes (en plus des secondes) le temps écoulé depuis le 01/01/1970 à minuit. Pour l'utiliser on récupère le temps avant d'effectuer ce que l'on désire mesurer, puis on récupère le temps après et on fait la différence entre les deux. Il s'agit de **temps réel**.

1.3 Programmation (création et attente de la terminaison d'un processus/thread)

Pour cette partie (et dans tout les autres programmes d'ailleurs j'ai utilisé la fonction **clock_gettime()** avec l'option **CLOCK_MONOTONIC** qui me semblait la plus appropriée. Au début j'utilisais **gettimeofday()** et pour cette partie je pense que cette fonction est appropriée, mais pour les changements de contexte, la précision de **gettimeofday()** n'était pas assez bonne.

Pour les processus (respectivement threads) :
    - Point **initial** de mesure du temps : **Avant l'entrée dans la boucle for** qui va successivement créer puis détruire les processus (respectivement threads).
    - Point **final** de mesure du temps : Juste **après la même boucle for**.
    - La **séquence d'opérations** ainsi mesurée est la **création et attente de la terminaison d'un processus (respectivement d'une thread)** ainsi que les opérations de la boucle for : **initialisation du compteur** et, à chaque étape, une **comparaison** entre la valeur du compteur et la valeur de la borne et une l'**incrémentation** du compteur. Il y a aussi l'opération qui **appelle la fonction clock_gettime()** à la fin et éventuellement les opérations effectuée par la fonction clock_gettime() entre la mesure effective et la fin de la fonction. On peut négliger les opérations qui n'ont lieu qu'une seule fois comme on effectue beaucoup de mesures. Donc en **mesures non négligeables supplémentaires** il y a la **comparaison** et l'**incrémentation** dans la boucle for. Dans le sujet on voulait juste mesurer la création et attente de la terminaison d'un processus (respectivement d'une thread). Pour enlever ce temps nécessaire aux opération de la boucle for, on peut mesurer le temps d'exécution d'une boucle for qui ne fait rien et la soustraire à la mesure avant de diviser par le nombre de processus (respectivement threads) crées et détruits.

Commentaires sur les résultats obtenus :
Pour la création et attente de la terminaison d'un **processus** on obtient un temps de 70 microsecondes en moyenne en lançant le programme sans spécifier de processeur. J'obtiens un temps moyen de 48 microsecondes lorsque j'exécute le programme sur un seul processeur (avec

taskset -c).
Pour la création et attente de la terminaison d'une thread on obtient un temps de 9 microsecondes en moyenne. Le résultat est le même si l'on exécute le programme sans spécifier de processeur ou si l'on exécute explicitement sur un processeur.
Ces résultats me paraissent cohérents, je m'attendais à trouver que les thread soient crées et détruits plus rapidement que les processus (puisqu'ils n'ont pas à copier l'espace d'adressage).

Le résultat obtenu est constant, les mesures ne varient presque pas. Si on voulait publier les résultats, il faudrait effectuer plusieurs mesures à l'aide de nos programmes. Puis on calcule la moyenne, l'écart-type, la variance et éventuellement la médiane, le minimum, le maximum. On pourrait éventuellement aussi présenter les résultats sous forme d'histogramme.

Facteurs pouvant influencer la mesure effectuée :
L'utilisation d'un ou plusieurs processeur influence la mesure (je le constate pour les processus), pour observer l'impact, il suffit d'utiliser un processeur (ou un nombre arbitraire possible) explicitement grâce la commande taskset -c.
Un autre facteur qui intervient dans la mesure est le niveau d'utilisation des processeurs. En effet si le processeur a beaucoup d'autres tâches à effectuer en même temps, il aura moins de temps à consacrer à ce programme et donc le temps sera plus important. On peut vérifier cet impact en modulant la charge de travail des processeurs.
La priorité respective des applications peut aussi modifier la mesure. On peut vérifier cet impact en attribuant une priorité faible ou élevée à ce programme et aux autres applications.

Les résultats que j'obtiens fournissent des temps en moyenne, je ne peux donc pas fournir des temps minimum et maximum. Je peux juste fournir sur un grand nombre de mesures le minimum et le temps maximum des temps moyens.

4. Changement de contexte

Résultats des mesures :
- Changement de contexte entre deux processus :
- Sans spécifier de processeur : 2100 nanosecondes en moyenne avec des valeurs entre 1800 et 2250 nanosecondes.
- En exécutant explicitement le programme sur un processeur : 1000 nanosecondes en moyenne avec des valeurs entre 950 et 1050 nanosecondes.

- Changement de contexte entre deux threads :
- Sans spécifier de processeur : 2100 nanosecondes en moyenne avec des valeurs entre 1800 et 2250 nanosecondes (je trouve pareil que pour les processus).
- En exécutant explicitement le programme sur un processeur : 720 nanosecondes en moyenne avec des valeurs entre 700 et 780 nanosecondes.

Je remarque que si je ne spécifie pas de processeur les temps de changements de contexte sont les même pour les processus et pour les threads.
Si mon programme s'exécute sur un unique processeur, le temps de changement de contexte est plus faible pour les processus et les threads d'un facteur 2 pour les processus et 3 pour les threads). Ce temps de changement de contexte plus faible s'explique en partie je pense par le fait que les accès mémoire sont plus rapide si on effectue l'ensemble dans le même processeur. (On accède plus

rapidement aux registres).
La différence de temps des changements de contexte entre deux processus et entre deux threads en utilisant un unique processeur s'explique je pense par l'hyperthreading. Le gain de performance sur les changements de contexte est de 1,4 ce qui correspond à ce que l'on a vu en cours sur l'hyperthreading. Pour vérifier cela je devrais faire des tests en désactivant l'hyperthreading mais je n'ai pas trouver comment faire.

Pour obtenir l'ordonnancement désiré, j'utilise deux sémaphores sem1 et sem2, je procède de façon analogue pour les processus et les threads. Je fais deux boucles for (pour les threads : une dans chaque thread, pour les processus : une dans le processus père et une dans le processus fils), dans une des boucles j'attends sem1 puis j'active sem2, dans l'autre boucle j'attends sem2 puis j'active sem1. Ainsi le processeur va alternativement effectuer les opérations dans une boucle puis dans l'autre. Ensuite je divise le temps obtenu par deux fois le nombre d'itérations dans une boucle for.

5. Comparaison avec d'autres résultats de TP

J'ai échangé mon programme de benchmark avec des camarades, nous avons obtenu des résultats cohérents. On retrouvait des différence entre les processus et les threads de même ordre. Concernant l'implémentation nous avons notamment eu une discussion sur le fait de savoir si l'on doit faire une grande mesure ou plusieurs petites mesures (mesurer un paquet de feuille, ou une feuille). Je trouve ça d'ailleurs surprenant que l'on trouve des résultats cohérents alors que j'effectue une mesure sur plusieurs événements quand d'autres effectuent plusieurs mesure pour un événement (puis calculent la moyenne). Je pense que cela s'explique parce que statistiquement on retrouve un bon résultat même si individuellement les mesures sont mauvaises.

6. Outils de benchmarking

J'ai utilisé **lmbench** avec les commandes **lat_proc fork** pour mesurer le temps de la création et sortie d'un processus et **lat_ctx 2** qui calcule le temps de changement de contexte entre deux processus. Voici les résultats que j'obtiens :

- Création et attente de la terminaison d'un processus :
- Sans spécifier de processeur : 75 microsecondes en moyenne avec des valeurs entre 74 et 76 microsecondes.
- En exécutant explicitement le programme sur un processeur : 56 microsecondes en moyenne avec des valeurs entre 53 et 59 microsecondes.

- Changement de contexte entre deux processus :
- Sans spécifier de processeur : 1,70 microsecondes en moyenne avec des valeurs entre 1,68 et 1,74 microsecondes.
- En exécutant explicitement le programme sur un processeur : 0,63 microsecondes en moyenne avec des valeurs entre 0,62 et 0,65 microsecondes.

Je trouve donc des valeurs cohérentes avec lmbench. Les valeurs plus faible pour les changements de contexte me font penser que j'ai du écrire des instructions qui ne sont pas forcément nécéssaire et rajoutent un peu de temps de calcul.