Informatique embarquée : TP n°4

Temps estimé : ~ 4 heures.

Temps passé : ~ 12 heures.

Mesures de performances : création et terminaison de processus/thread

Avant-propos

Mes calculs de performance concernant la création et la terminaison d'un processus et d'un thread ont été effectués en utilisant les fichiers *bench_proc.c* et *bench_thread.c* compilés avec **GCC 6.3.0** (via un *makefile*) avec les options suivantes :

```
-Wall -Wextra -O0 -lrt -lm -pthread
```

De plus, les tests ont été effectués sur ma machine personnelle équipé d'un processeur Intel(R) *Celeron(R) CPU N3350 double cœur @ 1.10GHz* et 4 Gio de mémoire vive.

J'ai, en outre, utilisé le méthodes des feuilles de papier pour mesurer le temps de création et terminaison d'un processus/thread. C'est-à-dire que j'ai effectué *N* créations/terminaisons de processus/thread, j'ai mesuré le temps total et je l'ai divisé par *N* (içi, N = 100000).

Le temps a été mesuré avec la fonction *clock_gettime()*.

La partie création et terminaison d'un processus/thread se fait dans une fonction dédiée (*process_stat()* pour les processus, *thread_stat()* pour les threads). Le point initial de mesure tu temps se fait juste avant l'appel de la fonction, tandis que le point final de mesure du temps se fait juste après que la fonction est terminé son exécution.

Résultats

	Processus	Thread
Temps total en moyenne (en ns)	12836608637	4011689339
Temps pour 1 création/terminaison en moyenne (en ns)	128366	40116
Temps total en moyenne (en µs)	12836608	4011689
Temps pour 1 création/terminaison en moyenne (en µs)	128	40

Ratio : x3.2

On constate qu'en moyenne la création et la terminaison d'un thread est environ 3 fois plus rapide que la création et la terminaison d'un processus. Cela peut s'expliquer par le fait que le processus, plus gros que le thread en mémoire, prend plus de temps pour être crée.

Mesures de performances : changement de contexte explicite entre processus/thread

Les conditions de test sont identiques, à le différence que j'utilise les fichiers *bench_context_proc.c* et *bench_context_thread.c*.

La méthode de mesure du temps est la même que pour avant (N = 1000000), à la différence que la mesure se fait dans le processus/thread père. Le point de début de mesure se situe avant la boucle qui fera *sem_wait()* N fois, et le point de fin de mesure se situe juste après la boucle.

Dans le cadre de ce benchmark, j'utilise des sémaphores binaires de manière à ce que le processus/thread père fasse l'opération *sem_wait()* pour qu'il soit mis en attente, et que le processus/thread fils fasse *sem_post()* pour rendre à nouveau le père actif.

Dans le cadre de ce test, la classe d'ordonnancement appliquée est la classe temps partagé.

Résultats

	Changement de contexte entre processus	Changement de contexte entre threads
Temps total en moyenne (en ns)	847024541	818100688
Temps pour ~1 changement de contexte en moyenne (en ns)	847	818
Temps total en moyenne (en µs)	847024	818100
Temps pour ~1 changement de contexte en moyenne (en µs)	0,847	0,818

D'après ces mesures, on constate qu'il y très peu de différence dans le temps mis entre chaque changement de contexte. Il est possible que cela soit dû à la classe d'ordonnancement temps partagée. En effet, les deux programme tels quels, ne permettent pas de s'assure que le processus/thread père se lance avant le processus/thread fils. De ce fait, il est possible qu'à certaines itérations, il y ait deux changements de contexte, tandis que dans d'autres, il n'y en ait qu'une. De plus, il est possible que les résultats soient complètement différents, dans la mesure où ils peuvent tout aussi bien indiquer que le changement de contexte est plus rapide entre 2 threads qu'entre 2 processus et vice-versa.