TD4 - InfoEmb

Jérôme Skoda, Joaquim Lefranc

Novembre 2017

1 Estimations

Initiale : 4h (Pas le choix il faut le rendre dans 4h)
Réel : 7h: (4h + 3h d'extra)

2 Etapes pour la création/destruction

2.1 Quel outil pour quelle mesure?

Et bien il faut mesurer le total, donc la ramette, puis le diviser par le nombre de feuilles dans la ramette. C'est le même principe avec le temps dans le tp. On peut mesurer un gros bloc d'opérations puis diviser le total par le nombre d'opérations dans le bloc.

2.2 (Optionnel) Fonctions de mesure de temps sous linux

gettimeofday(): C'est un appel system, il retourne le nombre de secondes écoulées depuis le 01/01/1970. Elle donne aussi les microsecondes. La précision est environ de 0,5ms sur Debian (Comme vu dans vos cours). (Temp mural)

time_t time : Elle vient de la librairie time.h, le point de départ est 01/01/1970. (Temp mural)

clock_gettime : Elle vient de la librairie time.h, elle peut mesurer le temps de différentes façons, elle retourne un résultat en secondes ou nanosecondes.

clock : Elle vient de la librairie time.h, temps de départ : lancement du processus (Mesure du temps CPU)
times : C'est une app system : Page (1) du man.

2.3 Mesure d'opérations en C : résultat des deux mesures

Processus
Source: tempsExecution/processus.c
Point initial de mesure du temps: Avant la boucle de fork
Point final de mesure du temps: Après la boucle de fork

Les mesures prennent le temps de création d'un fork ainsi que l'incrémentation de la variable n_processus. La mesure du temps d'incrementation parasite légèrement la mesure mais il s'agit surement de la solution la plus compréhensible et simple à mettre en oeuvre.

Thread
Source: tempsExecution/thread.c
Point initial de mesure du temps: Avant la boucle de pthread_create
Point final de mesure du temps: Après la boucle de pthread_create

Comme pour la messure des processus, l'incrémentation de la variable n_thread parasite la mesure.

Résultat obtenu

Chacun des résultat suivant sont produit avec "taskset -c 0" pour avoir une exécution sur un seul coeur.

	Desktop	Laptop
processus	53.939709 ms	86.157382 ms
thread	12.604273 ms	24.614902 ms

Desktop: CPU: i7 4790K @ 4.3GHz 4 cores 8 threads RAM: 16Go @ 1 600MHz Laptop: CPU: i5-2430M @ 2.40GHz 2 cores 4 threads RAM: 32Go @ 1 600MHz

2.4 Répétez votre mesure. Plusieurs fois. Le résultat obtenu est-il constant? Quelles méthodes statistiques devraient être utilisées pour « publier » des résultats ?

Le résultat n'est pas contant et il est fort probable que les résultat suivent une loi normale, utiliser une medianne, l'écart type ou la variance semble plus approprié.

2.5 Phénomènes et facteurs qui peuvent influencer la mesure

Les caractéritique physique d'une machine peuvent faire varier énormement les mesures (nombre de coeurs, fréquence etc) ainsi que l'état de la machine à l'instant de l'éxecution (nombre de processus actif). Il n'est pas possible de fournir un temps minimum ni même un temps maximum, cependant il est possible d'établir une ordre de grandeur. La création d'un thread est plus rapide de 3 à 5 fois que la création d'un processus.

Créer un thread prendre dans l'ordre de la dizaine de microseconde.
Créer un processus prend dans l'ordre de la cinquantaine de microseconde.

3 Changement de contexte

Source: tempsContext/processus.c et tempsContext/thread.c

	Desktop	Laptop
processus	1.187886 ms	2.895871 ms
thread	0.911997 ms	3.544407 ms

Desktop: CPU: i7 4790K @ 4.3GHz 4 cores 8 threads RAM: 16Go @ 1 600MHz Laptop: CPU: i5-2430M @ 2.40GHz 2 cores 4 threads RAM: 32Go @ 1 600MHz

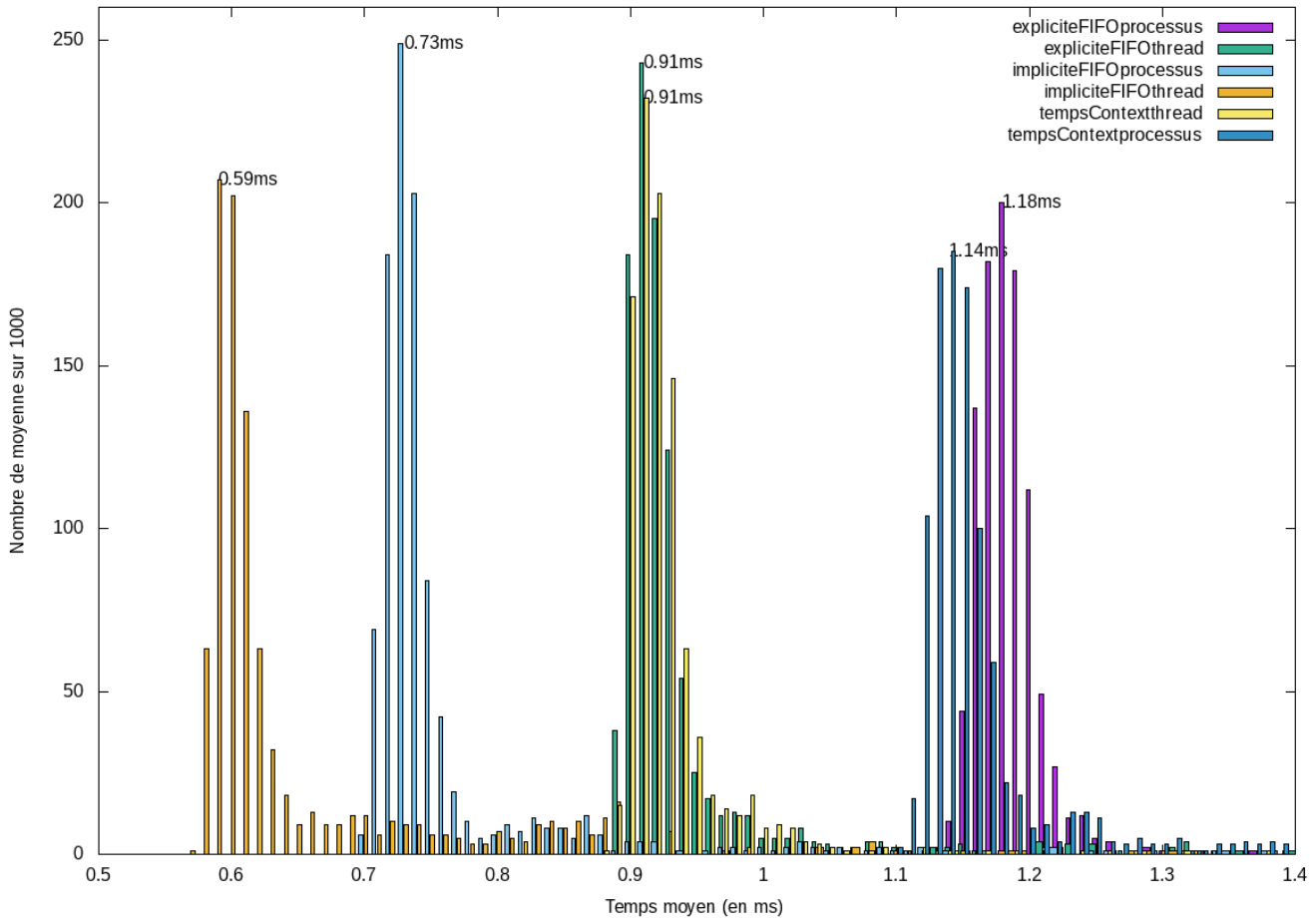
Pour faire l'ordonnancement, nous avons utilisé deux sémaphore: un est ouvert et l'autre bloqué à l'état initial. Chacun des processus ou thread attendent chacun l'ouverture d'un sémaphore et ouvre l'autre. De cette manière on obtient l'ordonnancement P1 P2 P1 P2 etc.

Les résultat peuvent varier en fonction de de sémaphore utilisé exemple dans thread.c avec des sémaphore partagé, le changement de contexte est plus long.

Source: expliciteFIFO/processus.c et expliciteFIFO/thread.c et impliciteFIFO/processus.c et impliciteFIFO/thread.c
Les résultats obtenus avec le script stat.py

Le changement de contexte implicite et explicite, on remarque que les thread sont plus rapide.

Voici un graph sur basé sur 1000 échantillon de moyenne. Le maximum de chaque courbe représente le résultat qui a statistiquement plus de chance de tomber.



On remarque que les temps avec le changement implificte FIFO sont plus bref que les temps avec un changement explicite.

4 Comparaison avec d'autres résultats de TP

Les différences peuvent venir du type de processeur, de sa charge actuelle (nombre d'applications éxécutées). De plus certain processeurs diminiuent leurs vitesse de fonctionnement ce qui peut influer sur le résultat. Cependant nos résultats sont cohérents avec ceux d'autres groupes, les ordres de grandeurs sont respectés.

5 Outils de « benchmarking »

La commande ./lat_proc -N 1000 fork nous donne une granularité de 62 microsecondes.

La personne chargé de cette partie est parti en voyage aux Bahamas.