Dias Alain

Rapport TP 3

1 – Réalisation

Pour la réalisation de ce TP, il fallait effectuer des mesures de temps afin de mesurer :

– La création de processus et de thread
– Le changement de contexte entre deux processus et deux threads.

Pour mener à bien ce travail il faut dans un premier temps identifier différentes fonctions de mesures de temps adaptées à notre besoin.

Lors de ma première lecture du sujet j'ai sous-estimé le temps que pourrait me prendre la réalisation de ce TP. En effet, je pensais parvenir à répondre aux questions posées ainsi que de produire le code demandé en 5 heures. Or, il m'a fallu 11h30 suite à différents essais de méthodes de synchronisation entre threads et processus lors de la partie des changements de contexte.

2 – Outils

Quels outils pour quelle mesures ?

Premier problème posé : *Mesurer l'épaisseur d'une feuille de papier issue de ramette à l'aide d'un mètre avec seulement des graduations en centimètres.*

Pour se faire, nous devons dans un premier temps mesurer l'épaisseur d'un minimum de feuilles pour obtenir au moins 1 centimètre, puis nous divisons cette épaisseur par le nombre de feuilles utilisées pour obtenir celle d'une feuille.

Ce problème est bien évidemment en rapport avec la question posée dans ce TP. En effet, nous devons mesurer des événements très courts avec des outils ne nous permettant pas forcément une précision telle qu'un événement sera mesuré. Il faut donc effectuer des mesures sur un paquet d'événements et non pas sur un seul (Exemple : Les changements de contextes)

3 – Identifications de fonctions de mesures de temps

Avant de commencer à chercher des fonctions, il fallait se poser la question « de quel temps parlons nous ? »

Les mesures étant orientées vers le « temps réel », nous parlons de temps « horloge murale ».

Une fois que nous avons déterminé le type de temps que nous voulons utiliser, nous pouvons maintenant commencer à regarder les différentes fonctions disponibles dans un evironnement Linux.

A) time

La fonction time (appel système) permet d'obtenir le nombres de secondes depuis le 1er janvier 1970. Le plus petit intervalle de temps observable par cette fonction est 1 seconde.

Puisque nous désirons faire des mesures d'événements beaucoup plus courts qu'une seconde, cette fonction ne nous intéresse pas.

B) gettimeofday

La fonction gettimeofday (appel système) permet d'obtenir le nombre de secondes et de micro-secondes depuis le 1er janvier 1970. Cette fonction a une résolution de l'ordre de la micro-seconde.

En exécutant mon programme à l'aide de l'option -d, j'obtiens pour ma machine :

Affichage Delta :
gettimeofday : 0 sec - 1 usec

C) clock_gettime

La fonction clock_gettime (librairie) nous permet d'obtenir des mesures de l'ordre de la nano-seconde.

En exécutant mon programme à l'aide de l'option -d, j'obtiens pour ma machine :

Affichage Delta :
clock_gettime : 0 sec - 200 nsec

J'ai choisi de travailler avec les fonctions clock_gettime ainsi que gettimeofday pour réaliser les mesures demandées dans ce TP

4 – Création et attente de terminaison

A) Processus

L'exécution de mon programme avec l'option -p ou --process nous permet de mesurer la création et l'attente de terminaison d'un processus. Le point de départ de ma mesure se situe juste avant le fork() et la fin de ma mesure se situe juste après le wait() dans le processus père.

Sur plusieurs exécutions j'obtiens sur ma machine :

Test de creation de processus :
Avec gettimeofday : 0 sec - 287 usec
Avec clock_gettime : 0 sec - 279952 nsec

Test de creation de processus :
Avec gettimeofday : 0 sec - 218 usec
Avec clock_gettime : 0 sec - 248493 nsec

Test de creation de processus :
Avec gettimeofday : 0 sec - 293 usec
Avec clock_gettime : 0 sec - 281889 nsec

Test de creation de processus :
Avec gettimeofday : 0 sec - 292 usec
Avec clock_gettime : 0 sec - 284620 nsec

B) Threads

L'exécution de mon programme avec l'option -t ou --thread nous permet de mesurer la création et l'attente de terminaison d'un thread. Le point de départ de ma mesure se situe juste avant l'appel de la fonction pthread_create() et la fin se situe juste après le pthread_join().

Sur plusieurs exécutions j'obtiens sur ma machine :

Test de creation de thread :
Avec gettimeofday : 0 sec - 180 usec
Avec clock_gettime : 0 sec - 86561 nsec

Test de creation de thread :
Avec gettimeofday : 0 sec - 181 usec
Avec clock_gettime : 0 sec - 94396 nsec

Test de creation de thread :
Avec gettimeofday : 0 sec - 181 usec
Avec clock_gettime : 0 sec - 61605 nsec

Test de creation de thread :
Avec gettimeofday : 0 sec - 152 usec
Avec clock_gettime : 0 sec - 35022 nsec

Test de creation de thread :
Avec gettimeofday : 0 sec - 181 usec
Avec clock_gettime : 0 sec - 45505 nsec

Lorsque l'on compare les mesures pour les processus et les threads, on constate que les threads sont beaucoup plus rapide à être créer que les processus. En effet, cela peut s'expliquer par le fait que lors de la création d'un processus, du temps est utilisé pour copier le contexte du père

pour le fils. La création d'un thread n'impose pas une telle copie puisque les deux threads evoluent dans le même espace d'adressage au sein du même processus.

De plus, les répétitions des mesures montrent qu'elles ne sont pas constantes.

Ceci peut être du :

– Aux effets qu'auront l'ensemble des processus en cours d'exécution sur la mémoire cache.
– L'ordonnancement opéré par le système entre les différents processus exécutes sur la machine

Je ne peux pas indiquer de temps minimum ou maximum pour la création d'un processus, c'est pourquoi je ne peux garantir aucune indication de temps pour une utilisation dans un système critique nécessitant du temps-réel.

5 – Changements de contextes

Afin de réaliser une comparaison entre un changement de contexte entre deux processus et deux threads, il est possible d'exécuter mon programme à l'aide de l'option « -s a » ou « --scheduling all »

Pour mesurer le temps d'un changement de contexte, plusieurs changements sont effectués, puis on divise par le nombre de changements total effectués (ici 1000 changements sont effectués par mesure).

On obtient donc sur plusieurs exécutions :

Test de scheduling
Entre processus
    Avec gettimeofday : 0 sec - 3 usec
    Avec clock_gettime : 0 sec - 3903 nsec
Entre threads
    Avec gettimeofday : 0 sec - 4 usec
    Avec clock_gettime : 0 sec - 3314 nsec

Test de scheduling
Entre processus
    Avec gettimeofday : 0 sec - 4 usec
    Avec clock_gettime : 0 sec - 3914 nsec
Entre threads
    Avec gettimeofday : 0 sec - 5 usec
    Avec clock_gettime : 0 sec - 3329 nsec

Test de scheduling
Entre processus
    Avec gettimeofday : 0 sec - 4 usec
    Avec clock_gettime : 0 sec - 4393 nsec
Entre threads
    Avec gettimeofday : 0 sec - 5 usec
    Avec clock_gettime : 0 sec - 3711 nsec

Lorsque l'on compare les mesures, on constate que les changements de contexte sont plus rapides entre threads.

Ceci peut s'expliquer par le fait que :

– Quand le système change de contexte pour un autre processus il doit sauvegarder l'état du premier et charger l'état du second.

– Contrairement aux processus, les threads d'un même processus partagent leur mémoire. Un changement de contexte sera donc plus rapide entre eux qu'entre deux processus

Afin d'obtenir l'ordonnancement désiré entre les processus ou threads, j'ai hésité entre l'utilisation de tubes et l'utilisation de sémaphores. J'ai finalement opté pour l'utilisation de sémaphores.

Puisque l'on ne contrôle pas l'ordonnancement de tous les processus sur la machine, nous ne pouvons que passer notre thread ou processus à l'état éligible pour que le système nous permette d'utiliser le CPU. L'ordonnancement du système peut donc perturber les mesures effectuées.