Correction serie threads SE M1 SII

Reponses aux questions exercice 1:

1- Pourquoi ne vois t'ont pas s'afficher le message Au revoir du thread cre e ?
parceque toute l'application s'est terminee a la terminaison du processus principal (le main) qui s'est termine avant le thread. → fin du main signifie fin de tout les threads crees.

2- De-commenter les 2 dernie res instructions de la fonction main et recompiler. Ve rifier les affichages. Quel est le ro le de la fonction thread_join ?

Maintenant tout s'affiche correctement car on a oblige le processus principal a attendre le thread avant de se terminer, cela laisse le temps au thread de se terminer correctement.

3- Que ce passera t il si on remplace l'appel a la fonction pthread_exit par un appel exit au niveau de la fonction thread_function ?

Un exit dans un thread va terminer toute l'application y compris le processus main.

4-Calculer le temps de calcul avec les n threads et comparer avec le temps se quentiel si le processus main avait exe cute la fonction thread_function dans une boucle au lieu de cre er des threads. Vous pouvez utiliser la commande time ./prog.

On remarque que le temps d'execution en sequentiel (un prog qui calcule le carre des elements d'un petit tableau) est plus petit que le temps obtenue avec l'utilisation des threads. Le prix (temps) de creation des n threads n'est pas couvert par le temps de calcul des carres.

5-Question sur l'ordonnancement des threads d'une application sur un CPU multi-coeurs : est ce qu'on gagne toujours du temps de calcul en cre ant un thread pour chaque tache d'une application ? Interet de l'utilisation de plusieurs threads dans une application ? Est ce que les threads d'une application sont ordonnances directement par le systeme ou alors ils s'executent dans le quantum de temps du processus principal ?

Non, on ne gagne pas toujours du temps en utilisant des threads paralleles, il faut que le traitement a effectuer par chaque thread soit assez complexe (prend bcp de temps) pour que son temps d'execution couvre le temps de creation du thread lui meme, sinon on perd du temps.

Les threads sont utilises pour divers raisons: parallelisation de programmes complexes sur plusieurs processeurs, implementation d'applications client-serveur sur systemes a memoire partagee, implementation de serveurs avec plusieurs threads pour traiter les requetes des clients en parallele, implementation d'applications avec interface utilisateur et plusieurs fonctionalites (utiliser un thread par fonctionalite pour meilleure fluidite) etc. → C'est pas toujours dans le but d'accelerer les calcul.

La politique d'ordonnacement depend du systeme d'exploitation, mais on peut voir et modifier cette propriete a travers les attributs des threads, l'attribut scope permet justement de voir le scope par defaut: quand scope=system chaque thread a son quantum de temps (parmis tout les threads du systeme) pour acceder aux CPUs de la machine, si scope=PTHREAD_SCOPE_PROCESS le thread sera execute dans le quantum de temps de l'application et donc on obtient du psoeudo parallelisme grace a l'utilisation du temps partage.