Programmation des threads POSIX : Exercice n°2

Points de matière concernés :

- Armement et masquage des signaux dans les threads (sigaction et sigprocmask)
- Annulation d'un thread (pthread_cancel)
- Fonction de terminaison (pthread_cleanup_push et pthread_cleanup_pop)

Etape 1 (armement d'un signal SIGINT)

A partir du thread principal, lancer 4 threads secondaires (on va les appeler ici **threads Slave**) sur la même fonction dans laquelle ils

- affichent leur identité (pthread_t, utilisation de **pthread_self()**) et un petit message disant qu'il vont attendre un signal
- attendent la réception d'un signal (utilisation de **pause()**)

Après avoir lancé ces 4 threads, le thread principal attend sur un **pause()** également, avant de se terminer par (**exit()** ou **pthread_exit()** → tester et comparer).

Dans le thread principal, armer (utilisation de **sigaction**) le signal SIGINT sur un handler dans lequel le thread récepteur du signal affichera

- son identité (pthread_t) et
- un petit message disant qu'il a reçu le signal.

Après gestion du signal le thread en question se termine par un pthread_exit().

Une fois le processus démarré, lancez-lui le signal SIGINT (commande kill ou <CTRL-C>) afin d'observer qui reçoit le signal.

Etape 2 (masquage du signal SIGINT)

A partir du thread principal, créer le **thread Master**. Ce thread attendra (dans une boucle infinie) la réception d'un signal (utilisation de pause()). Seul ce thread pourra recevoir le signal SIGINT. Vous devez donc masquer (utilisation de **sigprocmask**) correctement le signal SIGINT dans tous les autres threads. Vérifiez que c'est bien le cas en envoyant le signal SIGINT au processus.

Etape 3 (Armement et masquage du signal SIGUSR1)

Lorsque le **thread Master** recevra le signal SIGINT, il devra, dans le handler de SIGINT, envoyer le signal SIGUSR1 au processus en utilisant

kill(getpid(),SIGUSR1) ;

Seuls les **threads Slave** pourront recevoir le SIGUSR1 → vous devez donc masquer correctement ce signal dans les autres threads.

A la réception de SIGUSR1, les **threads Slaves** exécuteront un handler dans lequel ils afficheront

- leur identité (pthread_t) et
- un petit message disant qu'ils ont reçu le signal.
- Après gestion du signal les threads slaves se terminent par un pthread_exit().

Après avoir lancé tous les threads (le thread Master et les 4 threads Slave), le thread principal devra attendre la fin des 4 threads Slave (utilisation de **pthread_join()**) avant de se terminer lui-même.

Etape 4 (Annulation du thread Master) : BONUS

Il est impossible d'attendre la fin du thread Master qui ne se finira jamais vu qu'il se trouve dans une boucle infinie. Dès lors, après avoir attendu la fin des 4 threads Slave, le thread principal annulera le **thread Master** (utilisation de **pthread_cancel**) qui devra accepter immédiatement de disparaître.

Avant de terminer proprement le processus, le thread principal attendra la fin du thread Master.

Avant de disparaître complètement, le thread Master devra passer par une fonction de terminaison (utilisation de **pthread_cleanup_push()** et **pthread_cleanup_pop()**) dans laquelle il affichera son identité et un petit message disant qu'il se termine.