Compte Rendu TP2 NACHOS
Multithreading
Système d'Exploitation

Ludovic DELAVOIS
Brian LEBRETON
Master 1 Informatique - Université de Bordeaux

20 novembre 2018

1 Bilan

Lors de ce deuxième TP de Système d'Exploitation, nous nous sommes intéressés au multithreading dans un système NACHOS.

1.1 Partie I: Multithreading dans les programmes utilisateurs

Dans Nachos, les threads sont alloués et initialisés à l'aide d'un pointeur de pile initialisée à NULL ainsi que des registres initialisés à 0. Au début du listing, un programme est installé dans la mémoire grâce à la classe AddrSpace dans userprog/addrspace.cc. Un thread ne peut pas être créé si la pile est pleine (par défaut : 1024 octets) et que la taille d'un thread est de 256 octets. Ainsi, si la pile est pleine, on retourne -1 (erreur).

Afin de de pouvoir lancer un thread depuis un thread utilisateur, nous avons mis en place deux appels systèmes. Nos appels système sont ThreadCreate et ThreadExit, le premier permet de créer un thread et le deuxième de quitter un thread.
Pour mettre en place ces appels systèmes, nous avons d'abord suivi la procédure que nous avons déjà faite avec l'appel Putchar dans le Devoir 1. Pour cela il nous a fallu créer do_ThreadCreate dans laquelle nous utilisons une autre fonction StartUserThread, dans laquelle nous initialisons tous les registres dont nous avons besoin pour le nouveau thread. Dans do_ThreadCreate nous utilisons la fonction Start.

Start n'a que deux paramètres: une fonction et un argument a donner à cette fonction. Or nous lui donnons StartUserThread qui a besoin de beaucoup d'informations (pour les registres à initialiser). Il a donc fallu donc créer une structure, que nous avons nommé schmurtz, pour que toutes ses informations soient passées à Start et donc à StartUserThread. De façon similaire, il a fallu créer do_ThreadExit qui détruit le thread à l'aide de l'appel Finish().
Enfin, nous avons ajoutés les appels systèmes de ThreadCreate et ThreadExit dans syscall.h .

1.2 Partie II: Plusieurs threads par processus

Maintenant que nous avons réussi à créer un thread, nosu devons maintenant faire en sorte que Nachos fonctionne avec plusieurs threads.

Nous avons donc défini une variable globale dans addrspace "nbThreads" initialisée à 0 et qui s'incrémente ou décrémente à chaque création ou destruction d'un thread. Mais cela veut dire que chaque thread à la main sur cette variable, elle est partagée. N'étant pas atomiques, nous avons dû penser à protéger les incrémentations et les décrémentations par des sémaphores. Ainsi, dans addrspace.cc, les fonctions IncNbThreads, DecNbThreads sont protégées par des sémaphores (lock->P() et lock->V()).
Pour gérer le cas où le nombre de thread est égalà 0 dans les exceptions SC_Exit et SC_ThreadExit, nous avons créé une barrière dans exception.cc. Nous nous sommes servi encore une fois de sémaphores

utilisés par le biais des fonctions synchroThreadsV() et synchroThreadsP().

Ensuite dans exception.cc, nous avons implémenté une boucle if qui, lorsque le nombre de thread est supérieur à 0, alors on décrémente nbThreads (avec synchronisation). Si le nombre de thread est égal à 0 (sans compter le currentThread), alors on lève la synchronisation et on autorise le thread principal à s'arrêter.

Enfin, nous avons vu que, avec un nombre de threads très grand (999) dans makethreads.c (voir partie Tests), il manque des incrémentations du Putchar. C'est dû au fait que la pile n'est pas correctement utilisée (on écrit sur des morceaux de piles déjà alloués). Ainsi, nous avons implémenté des bitmap pour mémoriser quels parties de la pile étaient déjà utilisés ou non, puis pour les libérer une fois le programme terminé.
Plus précisément, le bitmap va chercher un premier emplacement libre dans la pile de 256 octets afin d'y allouer le premier thread. Lors de l'arrêt du thread dans ThreadExit, l'espace dans la pile est libérée avec la fonction ClearBitMap.

2 Points délicats

La partie consistant à endormir un thread s'il n'est pas le dernier, ou réveiller les autres sinon, nous a demandé un temps de raisonnement conséquent avant de réussir à l'implémenter.

3 Limitations

Nous avons réussi à finir ce TP (sauf les parties Bonus) en vérifiant la bonne configuration du code par l'intermédiaire de tests.

4 Tests

Pour tester notre code, nous avons créé le fichier makethreads.c qui appelle la fonction PutChar pour afficher des caractères en console. Il nous a servi à démontrer les failles de notre code, dans le sens où pendant le TP, la pile était parfois mal utilisée. Ainsi, certains threads n'étaient pas actifs, ou encore, les incrémentations non sécurisées "sautaient" des affichages de caractère.

Finalement, nous avons une bonne implémentation du code puisque notre test fonctionne comme nous le souhaitions.