Martial DUVERNEIX
Teiki PEPIN

Rapport
Systèmes d'exploitation
Projet 2 : Multithreading

I – Bilan :

Dans le cadre de la mise en place d'un système de multithreading, nous avons commencé par l'étude du fonctionnement des threads sous Nachos. Ces threads sont alloués et initialisés dans le fichier *threads/thread.h* via les fonctions *Thread(const char *debugName)* et *void Start(…)*. La pile d'un thread est de type *AddrSpace* dans *thread.h*.

La création d'un thread pourrait échouer dans les cas où la pile n'aurait pas suffisamment de place disponible ou si le constructeur *Thread(…)* ne fonctionne pas. Nous avons ensuite procédé à la création des fonctions nécessaires à la création et la destruction des threads dans le fichier *userthread.cc*.

Nous avons ensuite rajouté 2 sémaphores à notre classe *SynchConsole* afin d'empêcher l'accès simultané aux entrées-sorties. Un sémaphore est utilisé pour *SynchPutChar* et *SynchGetChar*, tandis que l'autre est utilisée dans les *SynchGetString* et *SynchPutString* afin d'empêcher que l'affichage ou l'entrée d'un string soit interrompue par un autre.

Dans le cas où le thread créé et le thread initial utilisent tous les deux *ThreadExit()*, *Nachos* ne fait jamais appel à *Halt()* et ne se termine pas. Ceci a été corrigé en ajoutant un compteur *int cptThreads* dans *addrspace.h* qui est incrémenté à la création d'un thread et décrémenté à sa destruction. Ces opérations sont protégées par un sémaphore *semCompteur*.

Lorsque l'on essaye de créer plusieurs threads, la pile du nouveau thread écrase la position en mémoire de thread créé précédemment. On corrige cela en affectant une position dans la pile à chaque thread créé en fonction du nombre actuel de threads existants.

Enfin, nous implémentons la classe *BitMap* pour gérer les positions dans la pile actuellement utilisées par les threads courants. Cela nous permet d'entièrement supprimer le risque de débordement de pile et d'allouer à nouveau un thread dans un emplacement libéré par un thread précédent.

II – Points délicats & limitations :

Initialement, la création des fonctions *do_ThreadCreate* et *StartUserThread* a posé de nombreuses difficultés. Plus particulièrement, un point critique est notre implémentation d'un *typedef struct {…} structSchmurtz* afin de transmettre la fonction et son argument à lancer via un seul *void**.

Pour protéger nos entrées-sorties, nous avons utilisé les sémaphores directement au lieu des verrous.

Après implémentation de *BitMap*, il est nécessaire d'utiliser l'ordonnancement préemptif en exécutant *Nachos* sur le nombre de threads que l'on souhaite créer. Ainsi, pour un programme qui prévoit de lancer un total de 9 threads, il faut le paramètre "*-rs 123456789*". Dans le cas où cela n'est pas fourni, certains threads risquent de se terminer au cours de leur exécution.

Les questions traitant de la terminaison automatique et de remonter l'accès aux sémaphores au niveau utilisateur n'ont pas été traitées.

III - Tests :

Au bout de la première partie, nous avons mis en place un programme de test dans *makethreads.c* qui nous a permis de vérifier le bon fonctionnement de la création d'un unique thread supplémentaire à partir du *main*. Le *main* fait alors utilisation d'une boucle infinie pour éviter que celui-ci se termine avant que le thread créé est fini son exécution.

En début de deuxième partie, le programme de test fut modifié afin de confirmer que les threads n'aient pas accès à la console de manière simultané et qu'un *PutString* ne puisse en interrompre un autre.

Avant d'utiliser la classe *BitMap* pour gérer les emplacements des threads dans la mémoire, notre programme de test nous permettait de lancer 3 threads en plus du thread initial et de faire des affichages en simultané.

Une fois le *BitMap* intégré à nos classes, nous avons modifié notre programme de test afin de lancer 9 threads à partir du *main*. Cela nous permet d'avoir 6 d'entre eux qui s'exécutent entièrement et les 3 autres qui renvoient un code d'erreur correspondant à une pile surchargée. Tous les threads y compris le thread initial se terminent correctement avant que Nachos se termine à son tour.