RAPPORT TP2

Abdoulaye DIALLO, Theophyl Hoguet

I . Bilan

Pour ce TP nous avons implémenté toutes les questions sauf la partie III et la question II.4, par manque de temps. Tout fonctionne correctement jusqu'à la fin de la partie II. Nous avons géré la création et la gestion de plusieurs threads dans le noyau, sans qu'il y ait de problèmes. Cependant comme expliqué plus précisément dans la partie limitation: nous n'arrivons pas à créer plus de 4 threads.

II . Points délicats

Pour gérer la création d'un nouveau thread avec la fonction `Start`, nous avons décidé d'utiliser une structure qui contiendra la fonction, son argument, ainsi que son espace mémoire. Nous passons ensuite cette structure `Start` pour son paramètre `schmurtz`, car il est de type `void *`, donc on peut lui donner tout type d'entrée de paramètre. Nous avons passé beaucoup de temps à savoir où "allouer" la place pour le thread dans la pile et comment communiquer cette information lors de la création du thread. Un autre point délicat qu'on a dû gérer était comment gérer l'allocation en mémoire des différents threads créés. Pour cela nous avons dû créer un compteur de threads qui était incrémenté à chaque nouvelle création de threads et qui était décrémenté à chaque destruction de thread. De plus , nous avons décidé de mettre en place des locks, plutôt que des sémaphores pour protéger le compteur de thread.

III . Limitations

Nous avons un problème avec les threads : il ne peut y avoir plus de 4 threads, même si on supprime la condition : `threadCounter > (UserStacksAreaSize / 256)` dans addrspace, il sera créé mais jamais exécuté. Il en découle que nous n'avons pas réussi à faire le II.4. nous ne comprenons pas comment nous pouvons créer plus de 4 threads si notre pile ne fait que 1024 "octets" même si nous mémorisons les slots de piles qui sont déja utilisé nous ne pouvons pas en créer plus, il faudrait déplacer la mémoire des threads en attente à un autre endroit ? Nous avons essayé de mettre en place le bitmap qui aurait sauvegardé les slots utilisés, mais nous n'avons pas bien pu implementer les fonctions. On a d'abord créé une fonction `IdFreeBitMap`

qui retourne le premier slot libre, on donne cette valeur à notre deuxième fonction `AllocateUserStack` qui alloue le thread en attente au slot donné. Ceci aurait permi un meilleur ordonnacement des threads en attente. Malheureusement, nous n'avons pas su comment modifié `userthread.cc` afin de prendre ces modifications en compte.

IV . Tests

A travers l'évolution de notre gestion des threads nous avons dû modifier notre test sur nos threads. Initialement `makethreads.c` ne faisait qu'une simple création d'un thread puis lançait `Putchar` dans une fonction `f`, pour vérifier que le thread ait bien été créé.

Lorsque nous avons du créer plus de threads nous avons modifié notre main, nous avons ajouté plusieurs `ThreadCreate` qui lancent la même fonction `f` avec différents paramètres pour que nous puissions voir que les threads sont bien créés.