RAPPORT TP3

Abdoulaye DIALLO, Theophyl Hoguet

I . Bilan

Pour ce TP final, nous avons dû gérer la création de processus et la gestion de la pagination de la mémoire. En plus de cela nous avons implémenter la création de plusieurs processus concurrents(forkexec). Nous avons tout implémenté (sauf les bonus), tout fonctionne (normalement)

II . Points délicats

Nous avons commencé par modifier l'accès à la mémoire MIPS, car jusque là on pouvait directement écrire dans la mémoire physique de la machine. Ceci peut évidemment créer des problèmes plus tard. Nous avons donc modifié ces écritures en écritures dans la mémoire virtuelle. En plus de cette modification, nous avons dû ajouter une classe "PageProvider" qui va aider l'allocation des pages physiques en mémoire. Elle implémente "GetEmptyPage()" qui va chercher à utiliser "Find()" de "BitMap" afin de trouver un page libre en mémoire, et "ReleasePage()" qui va libérer une page précédemment occupée. A ces méthodes nous avons ajouté "bookPages()" qui nous permet de réserver des pages avant de l'allouer. La réservation de pages permet d'éviter que lorsque l'on essaie de réserver une plage mémoire pour x thread, on alloue un thread par un thread jusqu'à ce qu'il y ait une erreur (ou pas) et si c'est le cas alors désallouer tous les threads (qui viennent à peine d'être alloués). Ici on préfère vérifier s'il y assez de place avant d'allouer avec un "bookPages" qui sait combien de place il reste à l'avance. Pour que cette méthode fonctionne il faut que tous les threads utilisent les "bookPages()" et "ReleasePage()"

Avec "forkexec" nous créons de nouveaux processus, en plus du processus principal, qui sont exécutés puis font des appels syscall, puis terminent. Mais nous avons remarqué un problème, dès que l'un des processus termine, la machine s'arrête aussi avant la fin des autres processus. Par exemple, lorsque nous lançons "forkexectest", qui écrit 30 caractères deux fois(car nous lançons 2 processus). Ce test ne s'affiche jamais les 60 caractères car les 2 processus sont concurrents, dès que le premier fini la machine s'arrête. Pour corriger ce problème nous avons utilisé un compteur de processus, des méthodes pour incrémenter/décrémenter le nombres de processus, ainsi que des locks pour sécuriser le tout. Maintenant quand les 2 processus sont lancés, la machine ne s'arrête qu'une fois les 2 terminés. Dans notre test "forkexectest" nous avons maintenant les 60 caractères désirés.

Un autre point délicat que nous devons gérer était la terminaison des threads au sein d'un processus. Jusque là quand un thread se terminait la machine s'arrêtait, mais maintenant nous voulons que les threads se terminent sans arrêter la machine. Nous voulons qu'un processus ne s'arrête qu'une fois tous ses threads terminés. Pour réaliser cette partie nous utilisons notre compteur de thread pour savoir quand il n'y a plus de threads en cours. Dès

que le nombre de threads est nul nous pouvons tuer le processus. Ensuite la machine “Powerdown()” lorsque tous les processus sont terminés.

III . Limitations

Dans notre implémentation nous avons réussi à gérer la création d’un grand nombre de processus et de threads. Le problème que nous avons est quand on veut créer plusieurs processus il faut modifier “NumPhysPages” et “UserStackArea” à des tailles adéquates pour avoir assez de mémoire pour les processus ainsi que les threads. Nous avons réussi à créer jusqu’à 15 processus avec 15 threads par processus, en “NumPhysPages” à 2560 et “UserStackArea” à 10240. Ces valeurs pour “NumPhysPages” et “UserStackArea” sont peut-être trop élevées mais nous pensons que ces valeurs nous ont permis de maintenir notre implémentation stable. Nous admettons qu’il existe peut être des valeurs plus optimales pour “NumPhysPages” et “UserStackArea” pour utiliser les ressources de l’ordinateur de manière plus efficace.
Avec notre implémentation, tous les threads (même le thread principale) doivent utiliser “ThreadExit()” et non pas “Exit()”. De plus, pour qu’un processus se termine il faut que tous ses threads s'arrêtent d’eux même, ce qui n’est pas optimal.

IV . Tests

Pour nos tests nous avons utilisé la méthode dite du comptage, nous avons créé un certain nombre de processus et de thread par processus et nous vérifions que le nombre de caractères écrit par le programme correspond à ce que l’on attend. Par exemple si on a une fonction f qui écrit 31 caractères et que l’on fait deux processus qui font chacun un thread qui exécute f et qu’ils exécutent f eux même alors nous sommes censés avoir 2 * 2 * 31 = 124 caractères dans le terminal à la fin Avec ce test la condition de “stress” serait le nombre de processus et threads que nous créons. Comme dit dans la partie III, modifier “NumPhysPages” et “UserStackArea” est nécessaire pour le bon fonctionnement de notre implémentation et de ce test.
(les fichiers de test se trouvent dans test, nommés “limitproc.c” et “limitthread.c”)