Systèmes d'Exploitation

Devoir 3 : Pagination

Noms:	***Numéros étudiants:***
Kerim Canakci	220 189 32
Corentin Drezen	223 094 35

I . Bilan

On a mis en place la pagination de la mémoire et l'appel système qui permet à un processus de se cloner et de lancer un nouveau processus (forkexec.cc).

Nous avons refait le bonus du TP précédent différemment pour appeler ThreadExit automatiquement à la fin d'un thread.

On a également implémenté la gestion de multiples processus et ajusté la taille des pages et des piles utilisateur.

Nous avons fait un shell grâce à nos nouvelles fonctionnalités.

D'après nos programmes utilisateurs de tests tout semble fonctionner normalement hormis les bonus de la partie II, il y a juste un problème de synchronisation et un problème de mémoire (pertes?) d'après valgrind.

II . Points délicats

Comme on avait implémenté le ThreadExit automatique à l'aide de macros au lieu de modifier le code assembleur au TP précédent on a d'abord pensé qu'il serait nécessaire de le refaire en assembleur pour ce TP. Donc on a rajouté une ligne au Start.S pour que lors de l'appel à ThreadCreate l'adresse de ThreadExit soit dans un registre et donc que l'on puisse ajouter une dernière instruction ThreadExit à la fin chaque nouveau thread.

On a pensé à modifier notre gestion de la mémoire pour qu'elle libère le stack à la destruction d'un thread et qu'elle libère les pages à la destruction de processus.

On a aussi remplacé l'appel automatique à Exit par l'appel automatique à ThreadExit (dans le Start.S) et modifié l'appel (dans userthread.cc) afin de ne pas avoir à les appeler

implicitement dans les programmes test et que le thread parent ne coupe pas l'herbe sous le pied de ses enfants lorsqu'il n'a plus rien à faire. Pour ce dernier point on a dû rajouter un compteur de threads actifs dans l'objet qui représente l'espace d'adressage (donc le processus). Pour gérer l'arrêt de la machine lorsqu'il y a plusieurs processus on a simplement rajouté un compteur de processus dans "forkexec.cc".

Cependant, comme on nous a demandé de permettre l'arrêt avec à la fois ThreadExit et avec Exit dans un bonus, on suppose que Exit appelé explicitement doit permettre l'arrêt brutal de tous les threads d'un processus.

On a alors essayé de stopper les processus de notre List avec Thread::Finish() de la même façon que l'on stoppait le thread courant avec ThreadExit mais nachos s'est plaint "Cannot finish another thread". On a donc essayé de changer de thread courant de plusieurs façons (sans vraiment réussir à priori) et nachos s'est plaint "Cannot destroy ourself".

```
abcdgfijekqsunzAssertion this != currentThread
Cannot destroy ourself!
```

```
abcdgfijekqsunzAssertion this == currentThread
Cannot finish another thread!
```

On s'est alors dit qu'il faudrait simplement remplacer l'instruction suivante NextPCReg de tous les threads d'un processus vers ThreadExit. On a essayé (dans UserThread::do_Exit) mais on n'a pas réussi à trouver comment changer de contexte, nos threads continuent alors à s'exécuter malgré notre prise en compte de leur obsolescence.

On a aussi essayé de simplement libérer l'espace d'adressage de l'appelant mais on a eu une autre erreur nachos.

On a pas testé si les threads à terminer n'étaient tout simplement pas encore lancés.

Nous avons aussi tenté de régler le problème de verrous jamais libéré du bonus II.6 en ajoutant une méthode releaseLocks à ConsoleDriver qui seraient utilisée avant la destruction prématurée des threads gérés dans do_Exit.

III . Limitations

Notre pagination semble bien fonctionner et on a pensé à mettre des verrous.

Cependant l'appel à Exit pour arrêter un processus ne fonctionne pas.

Il manque aussi certainement des sections critiques et des verrous notamment pour "NbProc–" et "nbUserThreads". Un de nos programme test fonctionne mal avec -rs 1234 d'ailleurs et c'est sans doute dû à un problème de synchronisation, on a essayé d'ajouter des verrous sur des accès à ces variables mais le programme échouait encore plus souvent.

Valgrind nous dit qu'il y a des problèmes de mémoire mais on ne sait pas si ils viennent uniquement du programme test avec lequel on l'a lancé.

IV . Tests

- Pour tester la virtualisation de la mémoire on avait fait un programme test simple “userpage0”.

- Pour tester ForkExec on a d’abbord fait “testforkexec” qui lance deux processus de notre ancien programme test “putchar”.

- On a fait “testforkexec1” pour tester le fait que le programme courant et le programme lancé peuvent eux-même contenir des thread et aussi pour tester d'éventuel problème de synchronisation de la console sous plusieurs processus (en utilisant aussi un programme du précédant).

- Il y a le programme “testfockexecbcp” qui sert à vérifier que l’on peut lancer de nombreux processus contenant de nombreux threads, en ré-utilisant aussi un programme du précédent que l’on a modifié pour plus de clarté. Ce programme est censé faire des opérations un peu lente pour déceler plus facilement des problèmes de mémoire.

- Pour tester le bonus II.5 et II.6 on a fait le programme “testforkexecexit” qui lance un processus qui lance 27 threads affichant chacun une lettre de l’alphabet excepté un thread qui doit arrêter tous les autres encore en cours prématurément.

- Enfin on a fait un programme “testforkexecshell” qui implémente le petit shell de la partie III en remplaçant dans le code fourni les appels non implémentés par nos appels.