Rapport: Pagination dans NACHOS

MAHDI Yanis, LUBET Damien

December 8, 2024

Contents

1 Bilan

Dans le cadre ce ce projet, nous avons implémenté un mécanisme de gestion mémoire paginée dans Nachos, en nous appuyant sur les concepts vus en cours. Voici les principales étapes de notre travail :

- Modification de la table des pages pour introduire un décalage entre les pages physiques et virtuelles, en effet nous avons gérer la traduction d'adresses virtuelles en adresses physiques en introduisant un décalage.
- Création d'une classe `PageProvider` pour gérer l'allocation des pages physiques.
- Ajout de l'appel système `ForkExec` pour permettre l'exécution de plusieurs processus en parallèle.
- Gestion de la terminaison propre des processus, incluant la libération des ressources.

Les fonctionnalités de base fonctionnent correctement, nous avons aussi essayer de faire le bonus mélangeant Threads et Exit mais nous n'avons pas réussi à le faire fonctionner correctement. Nous n'arrivons pas à faire en sorte que les threads se terminent correctement avant de terminer le processus.

2 Points délicats

Parmi les aspects les plus complexes, la mise en place du lancement de plusieurs processus en parallèle avec chacun un grand nombre de threads a été particulièrement difficile, car déjà il fallait gérer le nombre de pages disponibles pour chaque processus, et ensuite il fallait gérer la terminaison de ces processus, tout en vérifiant que si deux processus arrivent en même temps, on devait les traiter en parallèle. Pour cela, nous avons implémenter un système de réservation de pages pour chaque processus qui permet de savoir si il y a assez de pages disponibles et de les réserver pour le processus en question, le cas échéant on refuse la création du processus.

De plus, le bonus avec les threads et l'appel système `Exit` a été difficile à mettre en place, car il fallait gérer la terminaison des threads avant de terminer le processus, et cela n'a pas été possible à mettre en place correctement. On a mit en place une liste de Threads liés à l'addrspace du processus, pour y ajouter/retirer les threads lors de leur création/terminaison, ainsi qu'une variable booléenne `stopThread` attribué directement à l'objet `Thread` pour savoir quand celui-ci doit se terminer quand il reprend la main via le `scheduler` dans la fonction Run. Malgré tout ça, nous n'avons pas réussi à faire en sorte que les threads se terminent correctement avant de terminer le processus. Il semblerait que la variable `stopThread` ne soit pas mise à jour correctement mais on ne comprend pas pourquoi. On a décidé d'inclure le code du bonus dans le rendu, même s'il ne fonctionne pas correctement, pour montrer que nous avons essayé de le faire et aussi parce qu'il ne casse pas le code de base (quand on ne fait pas appel à l'appel système `Exit`).

3 Limitations

Notre implémentation présente les avantages suivants :

- Une gestion mémoire paginée fonctionnelle.
- La possibilité d'exécuter plusieurs processus en parallèle.
- Plus besoin de spécifier ThreadExit pour terminer le main d'un programme de test en tant que Thread.

Cependant, notre implémentation présente également des limitations :

- Avec notre implémentation des pages, il est pas possible de créer un processus si il n'y a pas assez de pages disponibles. Par conséquent, pour l'Action II.4, disant que l'on peux lancer une douzaine de processus en parallèle, chez nous cela n'est pas possible sans augmenter la valeur de `NumPhysPages` mais nous créer le maximum de processus en simultané sans soucis.
- L'appel système `Exit` ne fonctionne pas correctement avec les threads.
- La tentative d'implémentation du bonus n'étant pas aboutie, il y a des risques de problèmes de mémoire surtout en utilisant l'appel système `Exit` avec des threads.

4 Tests

Nous avons conçu plusieurs tests pour vérifier le bon fonctionnement de notre implémentation de la pagination, Le code source des tests contient tous les commentaires nécessaires pour les exécuter avec les bons paramètres et pour connaître la sortie normale attendue à l'écran.

4.1 Stratégie de test pour ReadAtVirtual

Pour valider le bon fonctionnement de l'appel système `ReadAtVirtual`, nous avons conçu un test qui lit un fichier texte et affiche son contenu à l'écran. L'objectif est de vérifier que la lecture des pages virtuelles se fait correctement, et que le contenu du fichier est bien affiché à l'écran.

4.2 Stratégie de test pour ForkExec

Pour tester l'appel système `ForkExec`, nous avons écrit plusieurs programmes `simpleFork, twoProcessTwoThread, forkAndThreads` qui lancent plusieurs processus en parallèle. Chaque processus exécute un test simple, comme afficher un caractère spécifique plusieurs fois à l'écran. L'objectif était de vérifier que :

- Les processus sont bien créés et exécutés en parallèle.
- Les processus fonctionnent correctement, même avec plusieurs threads.
- Les ressources sont correctement libérées à la fin de chaque processus.
- L'utilisation d'un `Lock` sur les appels système permet d'éviter les conflits entre les processus concurrents.

Les résultats ont montré que tous les processus s'exécutaient correctement, même avec des scénarios de stress (par exemple, lancement simultané de 12 processus avec plusieurs threads chacun).

4.3 Tests de terminaison des processus

Pour tester la terminaison des processus, nous avons conçu un scénario où plusieurs processus sont lancés et se terminent. L'objectif étant de vérifier que :

- Chaque processus libère ses ressources correctement avant de se terminer.
- La machine s'éteint automatiquement lorsque le dernier processus se termine.
- Aucun conflit ou comportement indéterminé n'apparaît lors de la libération des pages.

Les tests ont montré que notre implémentation gérait correctement ces scénarios.

4.4 Test du bonus Threads et Exit

Pour tester la terminaison d'un processus proprement avec des threads via un exit, nous avons conçu un scénario où un thread appelle Exit pour terminer l'entièreté du programme. L'objectif étant de vérifier que :

- Tous les threads du processus se terminent correctement avant que le processus se termine.
- Que toutes les ressources sont bien libérées.

Cependant, cela ne fonctionne pas correctement, les ressources ne sont pas libérées correctement et donc le programme se termine avant que les threads aient le temps de se terminer si la boucle contenant Yield dans `do ForkExit` est commentée. Et si elle est décommentée, le programme ne se termine pas de lui-même.

5 Conclusion

Ce projet nous a permis de mieux comprendre les mécanismes de pagination et de gestion des processus dans un système d'exploitation. Bien que nous ayons réussi à implémenter les fonctionnalités de base, certaines améliorations pourraient encore être apportées :

- Ajout de tests supplémentaires pour valider les cas extrêmes (par exemple, surcharge mémoire).
- Implémentation des bonus pour une gestion mixte des threads et des appels système `Exit`.

6 Conclusion

Ce projet nous a permis de mieux comprendre les mécanismes de pagination et de gestion des processus dans un système d'exploitation. Bien que nous ayons réussi à implémenter les fonctionnalités de base, certaines améliorations pourraient encore être apportées :

- Ajout de tests supplémentaires pour valider les cas extrêmes (par exemple, surcharge mémoire).
- Implémentation des bonus pour une gestion mixte des threads et des appels système `Exit`.

En conclusion, notre implémentation est fonctionnelle et répond aux objectifs principaux du projet, tout en fournissant une base solide pour des améliorations futures.