Rapport Système d'Exploitation

Pagination

14 décembre 2015

TISNÉ Romain - VIGUÉ Thomas

1 Bilan

1.1 Partie I

Dans la première partie du TD nous avons vu à l'instruction suivant :

```
machine->mainMemory[adresseVirtuel]
```

que la méthode ReadAt écrivait directement dans le mémoire physique de la machine. On prend l'adresse virtuel du code ou des datas et on le recherche dans le tableau de la mémoire physique ce qui n'est pas bon.

On a donc du créer une fonction ReadAtVirtual qui permet de faire la même chose que ReadAt mais en écrivant dans la mémoire virtuel correspondante cette fois ci.
Pour obtenir la bonne mémoire virtuel on a du faire changer la machine de processus. Pour cela on a du sauvegarder l'état actuel de la machine, se placer dans la mémoire virtuel correspondante puis restaurer l'état de la machine une fois les opérations finis.

Comme demandé, on a modifié la création de la table des pages pour que la page virtuelle i soit une projection de la page physique i + 1, on a remarqué dans la trace que au niveau du translate, toutes les pages étaient bien décalées d'une page. En effet tous les octets étaient décalés de 128, taille d'une page.

On a créer une classe **PageProvider**, contenant une instance de la classe **BitMap**, et permettant de gérer la mémoire physique de la machine. Nous l'avons instancié qu'une seule fois dans la méthode **Initialize** car accèder a la mémoire physique par plusieurs instances pourrait créer des conflits si c'est instance venait à y accéder en même temps.
Nous avons ensuite remplacé l'allocation d'une page virtuelle :
pageTable[i].physicalPage = i + 1 ;
par une allocation de page physique :
int emptypage = pageprovider->GetEmptyPage() ;
dans le constructeur d'**AddrSpace**.
Nous avons également libéré toutes les pages utilisées dans le destructeur avec la méthode **ReleasePage**.
On a testé avec un ASSERT si on peut toujours allouer une page. Si ce n'est pas le cas alors le programme s'arrête pour ne pas avoir de problème avec la mémoire.

1.2 Partie II

On essaye de créer le plus de choses possible dans le processus courrent avant de passer au nouveau afin de pouvoir mieux réagir en cas d'erreurs plutôt que d'être obligé de se "killer" si jamais on l'avait fait dans le nouveau thread, par exemple la création d'un nouvel espace d'adressage.

2 Points délicats

On a remarqué que pour obtenir une page vide il fallait récuperer le numéro de la première page vide disponible et récuperer cette page dans la **mainMemory** de la machine au bit numéro : numeroDeLaPageVide * TailleDeLaPage

3 Limitations

Une limitation que nous avons et que nous "killons" l'application dans le cas où nous avons plus de place lors de la création d'un nouveau processus.

4 Tests

Pour les tests nous avons créer les tests basiques demandés dans le sujet :

— **test/userpages0.c** qui sert de programme test afin d'être executé en temps que nouveau prossesus.

— **test/makeprocess.c** qui permet de lancer l'appel système **ForkExec** qui va créer un nouveau thread.