NACHOS : Entrées/Sorties

DEVOIR 1

Guillaume CHARLET
Kenji FONTAINE

15 octobre 2017

Table des matières

1 Description du projet

Ce travail a été réalisé dans un cadre universitaire, par des étudiants en Master 1 informatique à l'université de Bordeaux.
Un système d'exploitation est un ensemble de programmes permettant de diriger l'utilisation des ressources d'un ordinateur. Il assure la liaison entre l'utilisateur, les applications et le matériel. Nachos est un processus permettant d'émuler un système d'exploitation et du matériel.
L'objectif de ce devoir est de mettre en place sous Nachos un système d'entrée-sortie minimale, permettant d'éxecuter de petits programmes. L'ensemble des modifications apportées sont entre balises #ifdef CHANGED et #endif.

2 Bilan

2.1 Partie 1

L'objectif de cette première partie est de créer un programme de test : putchar.c. Ce programme est placé dans le dossier /code/test.
La sortie attendue est "abcd", que nous arrivons à obtenir. La fonction nous paraissait relativement simple et ses appels par la suite ne nous ayant pas posé de problème particulier, nous n'avons pas pris le temps de tester certains cas plus particuliers.

2.2 Partie 2

Cette partie était relativement légère en code et était plutôt faite pour nous aider à mieux comprendre le mécanisme d'entrée-sortie de Nachos et aussi de nous faire comprendre en quoi lire un caractère avant d'être averti qu'un caractère soit disponible. En effet, la lecture se faisant au moment de l'appel, si nous n'avons pas l'assurance qu'un caractère soit disponible, nous allons lire ce qu'il y a dans le buffer, autrement dit nous allons obtenir un caractère inattendu.
Les tests ./nachos -c et ./nachos -sc ne peuvent plus être executés à cause de la partie 4, mais nous pensons avoir bien réussi cette partie.

2.3 Partie 3

Cette troisième partie a pour but de rendre synchrones les fonctions de la partie précédente. Pour se faire, nous avons utilisés des sémaphores. Il nous aura fallu un peu de temps et quelques explications, mais cette partie ne nous n'a pas posé de problème particulier. Cette partie nous a permis de comprendre le fonctionnement des sémaphores lorsqu'elles sont initialiées à 0, permettant de synchroniser des processus. Un premier processus va, après avoir effectuer son travail, réveiller un deuxième processus ; qui ne pourra pas commencer son travail tant qu'il n'aura pas reçu ce réveil.

2.4 Partie 4

L'objectif de cette partie est de mettre en place un appel système PutChar(c), qui prend en argument un caractère c en mode utilisateur, puis lève une interruption SyscallException. La SyscallException va provoquer un passage en mode noyau et l'éxecution du traitant standard, ExceptionHandler.
Nous avons suivi le sujet et cette partie ne nous a pas posé de problème particulier.

2.5 Partie 5

Cette partie a pour but de bufferisées la fonction putChar.

Pour l'implémentation de copyStringFromMachine, nous avons choisi de la mettre en tant que fonction dans le fichier /code/userprog/synchconsole.cc. Cette fonction copie une chaine de caractère du monde utilisateur vers le monde noyau. Pour cela, nous devons faire le lien entre ces deux mondes en utilisant les classes SynchConsole et Machine. De par ces critères, nous avons jugé pertinnent de mettre cette fonction dans synchconsole.cc où une instance de machine était facilement créable.

Nous avons testé plusieurs cas pour notre implémentation ; chaîne courte, trop longue , taille impaire/première. La fonction semblait marcher comme attendu et nous sommes donc passer à la partie suivante.

2.6 Partie 6

Lorsque nous avons enlevé l'appel à la fonction Halt(), nous avons obtenu un message d'erreur comme quoi l'interpution SC_Exit n'est pas gérée par l'ExceptionHandler. Pour éviter cela, nous avons ajouter le cas SC_Exit dans le switch. Nous pouvons lire la valeur de retour de main qui est sauvegardée dans le registre 2 lorsque celle-ci est déclarée à valeur entière.

2.7 Partie 7

Cette partie a pour but d'implémenter les appels systèmes GetChar et GetString.

Comme à la partie 5, où nous avons créé la fonction copyStringFromMachine, nous avons créé la fonction copyStringToMachine. Cet fonction copie une chaine de caractère du monde noyau vers le monde utilisateur, nous avons donc choisi de la mettre dans le même fichier que la fonction copyStringFromMachine.

Cette partie nous a posé beaucoup de problèmes et nous y avons passer beaucoup de temps. Expliquée plus en détails dans la partie suivante.

2.8 Partie 7 Bonus

Cette partie a pour but d'implementer les appels systèmes PutInt et GetInt, qui permettent de s'occuper des entier signé. Cette partie ne nous à pas posé de problème particulié, surtout grâce au fait de pouvoir utilisé les fonctions snprintf et sscanf.

2.9 Partie 8

Nous n'avons malheureusement pas eu le temps de nous pencher sur cette partie-là.

3 Points délicats

Nos principales difficultés viennent de notre négligence vis à vis des tests de nos fonctions. Nous n'avons pas pris la peine de tester suffisament de cas limites. Nous avons eu de la chance pour les premières fonctions du devoir, les appels de ces fonctions dans les parties suivantes ne nous ont pas posé de problème.

Lorsque nous avons implémenter la partie 5, nous avons testé quelques cas limites afin de vérifier son bon comportement. Ayant obtenu les résultats attendus, nous ne sommes pas attardés sur ses tests.

Puis nous nous sommes attaqués à la partie 7. Notre fichier de test pour la partie 7 faisait appel à PutChar, de la partie 5. Une fois arrivée à la phase de test de la partie 7, nous avons obtenu plusieurs erreurs d'origines inconnues. Malgrès plusieurs tentatives différentes, nous n'avons pas réussi à corriger ces erreurs. Nous avons donc décider d'essayer d'isoler le problème et avons rajouter des `\0` à la fin de chacune de nos sous-chaînes de caractères ainsi que des appels à printf. C'est à ce moment-là que nous nous sommes rendus compte que le problème ne venait pas de GetString, mais bien de PutString. Il s'agissait d'un cas particulier de PutString que nous n'avions pas pris le temps de tester ; la taille du buffer était un multiple de la taille de chaîne, ce qui avait pour effet de faire un tour de boucle de plus que prévu.

4 Tests

Les fichiers de tests se trouvent dans le dossier /code/test. Ils sont exécutables en utilisant la commande suivante :

```
$ ~/nachos/code/userprog ./nachos -x ../test/<fichier_test>
```

4.1 Partie 1 à 3

Nous n'avons pas pris le temps de tester entièrement ces fonctions. Leur implémentation était relativement simple et elles ne semblaient pas posé problème par la suite. Nous pensons que ces fonctions se comportent normalement. De plus, les tests des parties 2 et 3 ne sont plus exécutables à cause de l'implémentation de la partie 4.

4.2 Partie 4 : PutChar

Pour cette partie, nous avons essayer d'attendre longtemps avant de taper le caractère en entrée, le taper "vite". Nous avons également essayer des caractères spéciaux comme `\n` ou `\0`.

4.3 Partie 5 : PutString

Pour les tests de cette fonction nous devions jouer sur le buffer MAX_STRING_SIZE et la chaîne de caractères entrée par l'utilisateur. Nous avons essayé un buffer petit par rapport à la chaîne, afin d'avoir plusieurs tour de boucle. Nous avons également essayé des chaînes plus petites que le buffer. Des valeurs paires/impaires de taille. Un des cas que nous n'avions pas testé est un buffer petit dont la taille est un diviseur de la taille de la chaîne de l'utilisateur (Taille 4 pour le buffer et taille 16 pour la chaîne).

4.4 Partie 6 : Halt() et SC_Exit

Nous n'avons pas fait de test particulier pour cette partie. Nous avons simplement ajouté le cas SC_Exit au switch case de l'ExceptionHandler et avons relancé le fichier test PutChar, qui se comportait comme prévu.

4.5 Partie 7 : GetString

Les tests effectués pour cette partie sont similaires à ceux de la partie 5. Nous avons joué sur les tailles relatives du buffer et de la chaîne de caractères à récupérer. Le cas nous ayant posé problème à cause de PutString était avec un buffer petit et une chaîne longue (Buffer de taille 4 et chaîne de taille 16). Le fait d'appeler PutString sans être certain de son bon fonctionnement nous a fait perdre du temps.

4.6 Partie 7 : Bonus

Pour cette partie nous avons testé des entier positif et des entiers négatifs, afin de voir si le signe était bien respecté, nous avons pas traité les cas particulier où il y avaient d'autre caractères que le signe négatif et/ou des chiffres.