Compte Rendu TD1 NACHOS
Entrées/Sorties
Système d'Exploitation

Ludovic DELAVOIS
Brian LEBRETON
Master 1 Informatique - Université de Bordeaux

16 Octobre 2018

1 Bilan

Lors de ce premier TP de Système d'Exploitation, nous nous sommes intéressés aux entrées/sorties dans un système NACHOS.

1.1 Partie I: Quel est le but?

Dans un premier temps, nous avons travaillé sur le fichier putchar.c pour exécuter des entrées/sorties synchrones. On utilise dans le programme l'appel Putchar qui affiche un caractère (à la fois) en console. Une fois avoir mis la condition "if" à 1, le résultat attendu ce ce programme est d'avoir affiché la suite de caractères "abcd" en console avec un retour à la ligne.C'est ce que nous avons réussi à obtenir facilement.

1.2 Partie II: Entrées-sorties asynchrones

Dans cette partie, nous avons appris à se servir d'une version d'entrées-sorties. Pour cela, nous avons utilisé les appels readAvail et writeDone. Ils servent à attendre que l'utilisateur tape un caractère sur son clavier et à finir de l'afficher en console.
C'est une erreur que de chercher à lire un caractère avant d'être averti qu'un caractère soit disponible car on ne sait pas si ce caractère est bien arrivé à destination et on devrait donc lire le dernier caractère sauvegardé. Chercher à écrire avant d'être averti que l'écriture précédente soit terminée est une erreur car on risque d'écraser le caractère tapé précédemment.
Le programme progtest nous demande de taper un ou plusieurs caractères puis de taper sur Entrée pour les afficher (et quitte le programme quand on tape sur 'q'). On nous a ensuite demandé d'afficher chaque caractère entre '<' et '>' ainsi qu'afficher 'Au revoir!' en console quand on appuie sur 'Ctrl+D'. Enfin il fallait également que cela fonctionne avec un fichier d'entrée qui retourne le contenu de ce fichier entre '<' et '>' dans un fichier en sortie.
Nous avons bien réussi cette partie et ne nous a pas posé de problème particulier.

1.3 Partie III: Entrées-sorties synchrones

La partie III consistait à implémenter le même code que la partie précédente mais de manière synchrone. Une exécution synchrone permet à plusieurs threads d'exécuter une tâche mais seulement lorsque la tâche précédente est terminée. Pour cela, nous avons utilisé des sémaphores (après les avoir initialisés) et ajouté l'option '-sc' en remplaçant grossièrement les appels de 'console' par 'synchconsole'. Nous avons pris plus de temps que prévu pour cette partie car nous avons été un peu vite pour recopier les fonctions et il y avait quelques coquilles.

1.4 Partie IV: Appel système Putchar

Nous avons ici mis en place l'appel système PutChar(char c) qui prend en argument un caractère c en mode utilisateur puis lève une interruption SyscallException. Pour cela, nous avons dû nous familiariser avec le langage en assembleur dans le fichier start.S puis ajouter l'appel système de Putchar dans le fichier syscall.h. Nous avons réussi cette partie, même si nous avons pris du temps à comprendre les notions de registres au niveau du compilateur.

1.5 Partie V: Des caractères aux chaînes

Maintenant que PutChar fonctionne pour un caractère, l'idée de cette partie est de créer PutString qui fonctionne pour une chaîne de caractères. Pour cela nous avons implémenté SynchGetString et SynchPutString qui, avec une boucle while, utilisent SynchGetChar et SynchPutChar pour chaque caractère. Étant donné qu'on ne peut pas directement copier une chaîne du monde utilisateur vers le monde noyau, nous avons écrit la procédure copyStringFromMachine. Cette partie nous a posé quelques soucis (voir points délicats) mais nous avons finalement réussi à tout faire fonctionner correctement.
Pour tester notre programme nous avons créé le fichier putstring.c en utilisant une chaîne de caractères petite, puis longue.

1.6 Partie VI: Mais comment s'arrêter?

Lorsqu'on a enlevé l'appel à Halt() dans la fonction main, nous avons obtenu un message d'erreur nous indiquant que l'interruption SC_Exit n'était pas gérée par l'ExceptionHandler. Pour palier à ce problème, nous avons rajouté le cas SC_Exit dans le switch du fichier exception.cc.
Nous pouvons lire la valeur de retour de main qui est sauvegardée dans le registre 2 lorsque celle-ci est déclarée à valeur entière.

1.7 Partie VII: Fonctions de lecture

Cette partie a pour but d'implémenter les appels systèmes GetChar et GetString.
Comme à la partie V, où nous avons créé la fonction copyStringFromMachine, nous avons créé la fonction copyStringToMachine. Cette fonction copie une chaîne 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 passé beaucoup de temps; expliquée plus en détails dans la partie suivante. Nous n'avons pas su résoudre le problème que nous avons eu sur la fonction GetString.

2 Points délicats

Concernant la partie II, nous n'avions juste pas compris au début pourquoi il y avait d'afficher un simple '<>' puis nous avons compris que cela correspondait au caractère " dont nous avons retiré l'encadrement avec une boucle if.

Au niveau de la partie IV, nous avons mis assez de temps à comprendre pourquoi cela ne compilait pas correctement. En fait, nous n'avions pas crée correctement le constructeur SynchConsole et nous nous étions trompé de registre (4 au lieu de 2) dans l'ExceptionHandler lors de l'implémentation de SC_PutChar.

En ce qui concerne la partie V, nous avons été confronté à plusieurs erreurs:

- nous n'avions pas le droit de copier un string directement depuis le mode utilisateur
- nous avions une référence infinie pour SynchConsole::CopyStringFromMachine avec une débordement mémoire.
- il n'y avait qu'un caractère d'affiché au début car nous avions laissé deux WriteDone()->P qui bloque ensuite toute la boucle

Ce qui a été le plus compliqué pour nous est la mise en place du GetString dans la partie VII. En effet au début, nous nous sommes rendus compte qu'on ne récupérait que (n-1) caractères à chaque fois. Autrement dit il nous manquait un caractère. Il se trouve que nous écrasions le dernier caractère de la chaîne pour y placer un \0.

Une fois ce problème résolu nous avons passé tout le reste de notre temps à essayer de résoudre un autre problème. En effet, il nous fallait récupérer grâce à SynchGetString, la chaine entrée que nous mettions dans un buffer de taille MAX_STRING_SIZE puis utiliser copyStringToMachine pour écrire, à l'adresse récupérée dans les registres, le contenu du buffer.

Malheureusement, nous n'avons pas su trouver l'origine du problème puisque notre buffer récupère bien morceau par morceau la totalité de la chaîne entrée au clavier mais ne s'affiche au terminal uniquement les "MAX_STRING_SIZE" premiers caractères.
Nous supposons donc que le problème vient de copyStringToMachine ou de son appel dans SC_GetString ou alors de PutString mais il semble bien fonctionner.

Exemple :
MAX_STRING_SIZE = 5
Chaine entrée au clavier (taille 15) : 123456789098765
Résultat : <1><2><3><4><5>
Pourtant à chaque tour de boucle, le buffer vaut 12345 puis 68790 et enfin 98765.

3 Limitations

Par conséquent GetString ne fonctionne qu'avec des chaînes de caractères inférieures ou égales à la taille du buffer MAX_STRING_SIZE. Si la chaîne est supérieure, on ne récupère que les "MAX_STRING_SIZE" premiers caractères. Cela n'est donc pas gênant pour une petite chaîne de caractère, mais l'est pour une plus grande qui dépasserait la taille du buffer.

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 4 : PutChar

PutChar fonctionne correctement. Aucun souci aperçu.

4.2 Partie 5 : PutString

PutString Fonctionne correctement tant que la taille est inférieure ou égale à MAX_STRING_SIZE.

4.3 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.

4.4 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. Malheureusement il ne fonctionne pas pour une chaîne longue supérieure à MAX_STRING_SIZE.