Rapport Système d'Exploitation

Entrées / Sorties

14 octobre 2015

TISNÉ Romain - VIGUÉ Thomas

1 Bilan

1.1 Partie II

Lorsque l'on cherche à écrire un caractère sans asvoir si l'écriture précedente est terminée peut amener plusieurs processus à écrire dans la même zone et donc erroner les données. De même lorsqu'on tente de lire un caractère, on attend d'être prévenu qu'il y a quelque chose à lire pour éviter de lire des données qui seraient en cours d'écriture par exemple.

Nous avons éffectué les instructions demandées sans problèmes et si les arguments in et out sont à **null** la console s'exéxute est attend des caractères à recopier sur la sortie standard.

Remarque :. Les **retour-chariots** sont pris en compte et recopiés. Dans le cas d'encadrer les caractéres par des **< et >**, le retour-chariot est donc considéré comme un caractère et donc est entouré par les crochets.

1.2 Partie III

La console synchrone a bien été implémentée et est également fonctionnelle dans le cadre d'une copie de fichier. Afin de vérifier graphiquement que tout fonctionnait correctement nous affichons le caractère saisi entre **< et >** comme demandé.

1.3 Partie IV

Les appels systèmes suivent tous le même schéma de création. On s'est donc inspiré du code existant pour creer l'appel systéme PutChar. Pour l'assembleur on a modifié le code de Halt pour l'adapter a **PutChar** afin d'appeler l'instruction système correspondante. Aprè l'implémentation de tout le reste des instructions putchar fonctionne bien comme il faut.

1.4 Partie V

Pour la fonciton **copystringfrommachine** nous n'avions, dans un premier temps, pas pris en compte le fait que pour la dernière partie d'une longue chaine de caractère, la chaine de caractère était moins grande que le paramètre **size**. On a donc ajouté une condition pour s'arrêter au caractère **\0**. On a placé cette fonction dans le fichier **userprog/exception.cc** car c'est dans celui-ci qu'elle est appelée.
Pour l'appel système **PutString**, on a découpé la chaine MIPS en plusieurs bout de taille max **MAX_STRING_SIZE** afin de récuperé les chaines de caractère pouvant depasser cette limite jusqu'à rencontrer le caractère de fin de chaine **'\0'**.

1.5 Partie VI

En enlevant halt on obtient cette erreur :

```
Unimplemented system call 1
```

En effet, on a observé que lors d'un retour de fonction, l'exception **SC_Exit** était levée, celle ci n'étant pas implémentée, on a donc dut le faire durant cette partie.
Pour pouvoir se passer de l'appel a **Halt**, on récupère la valeur de retour dans le registre 4, on a du modifier le fichier **start.S** car de base le retour de fonction est dans le registre 2 qui est écraser par le numéro d ?appel système, et on appelle la fonction **EXIT** de nachos avec la valeur de retour.

1.6 Partie VII

Dans cette partie, l'implémentation de **GetChar()** s'est faite sans grande difficulteé, on récupère le caractère grâce à la console synchrone et on le stock dans le registre 2.
Pour GetString on a complèté la méthode SynchGetString, on a ajouté la fonction utilisateur nachos dans syscall.h. Cela permet de remplir un tableau de char s de taille n avec la chaine de caractères saisi par l'utilisateur dans le terminal. Comme pour la fonction **PutString** le buffer coupe la chaine de caractère de l'utilisateur pour éviter de devoir stocker l'intégralité de la chaine de caractères (parce que dans le noyau l'espace est limité).
Pour **PutInt()** et **GetInt()**, on écrit, ou récupère, en réalité une chaine formatée, avec les fonctions **snprintf()** et **scanf()**, ensuite il suffit de faire un **SynchPutString()**, ou **SynchGetString()** sur la console, suivant l'appel système visé.

1.7 PartieVIII

Par manque de temps, nous n'avons pas implémenté la solution consistant à gérer le caractère EOF.

2 Points délicats

Le premier point délicat que nous avons rencontré est tout d'abord la compréhension de tout le code source de **NACHOS**. En effet nous avons passé beaucoup de temps à essayer de comprendre les liens entre tous les fichiers, entre toutes les classes ...

Mais dans ce projet il y a un point particulier qui nous a plus posé problème. En effet la fonction **GetString** que nous avons implémentée ne retounait pas toujours le même résultat alors que **NACHOS** est cencé être déterministe. Après plusieurs recherches dans le code, on a trouvé que la valeur de retour de la fonction **Main()** était bizarrement stockée dans le registre 3 au lieu du registre 4 de Exit. On a donc modifié le code assembleur **"start.S"** pour copier la valeur du registre 2, qui était déjà utilisé pour les valeurs des appels systèmes, dans le registre 4 comme expecté.

3 Limitations

Nous avons essayé de rendre le code le plus clair possible en endentent et en commentant tout celui ci et d'après les tests effectués il semble fonctionnel. Cependant à cause du manque de temps nous n'avons pas effectué des tests plus pousés. Nous avons surtout joué avec la taille du buffer,tester des chaines inférieures égale ou supérieur au buffeur pour voir si on n'avait pas un comportement suspect et non désireé.

Autre problème la lecture de chaine par **getString()** ne prend qu'un nombre limité de caractères passé en argument, et d'ailleurs un comportement qui peut être gênant a été détecté ; les caractères que **getString()** ne récupère pas, sont renvoyés dans le terminal et exécutés comme une commande.

4 Tests

Les tests effectués sont relativement sommaires, et se contentent généralement d'une chaine de caractères ou d'un entier. Les primitives n'agissant que sur ces objets, il teste seulement la possibilité de dépasser la taille maximale prévue ou d'insérer un caractère spécial.

Chaque primitive système dispose de son programme de test dans le dossier test et peut s'utiliser avec la commande :

```
./nachos -x ../test/nomDuTest
```