J1INAW11: NachOS project

Due on Monday, October 12, 2015

NAMYST Raymond

VER VALEM Willian , RAKOTONIERA Hoby

Contents

Introduction

J'ai réalisé l'ensemble des appels systèmes qui étaient demandés : *PutChar*, *PutString*,*Exit*, *GetChar*, *GetString*. L'ensemble du projet a été réalisé sans difficultés particulières, hormis des difficultés de temps. C'est pour cette raison que je n'ai pas pu réaliser les bonus *printf*, *PutInt* et *GetInt*.

J'ai passé un certain temps, pour commencer, à essayer de comprendre le code, et de me faire une idée du fonctionnement du programme. Cela a été ma principale difficulté.

Après ce temps d'adaptation et de compréhension, j'ai pu terminer la totalité de la mise en oeuvre de ces exercices. La seule chose que je n'ai pas pu finir c'est, pour la raison de temps évoquée plus haut, celle des bonus.

Pour ce qui a été fait, tout fonctionne correctement. Après avoir testé les fonctionnements de ces appels système, je n'ai pas pu trouver de bug. J'espère simplement que ce n'est pas par manque de parce que je n'ai pas su les détecter...

En ce qui concerne les choses que je n'ai pas eu le temps de réaliser, je suppose qu'elles seraient du même ordre de difficulté que ce que j'ai déjà fait, et donc il me semble que je devrais y arriver de même, en espérant là aussi que je serais capable de trouver d'éventuels bugs.

Je pense qu'il me faudrait deux jours de travail de plus pour finir le bonus, à condition que je puisse m'y consacrer exclusivement.

Points délicats

Entrée-sorties synchrones

L'erreur commise dans cette partie du projet a été de ne pas inclure de *break* en cas de fin de fichier (*EOF*), fin de chaîne de caractères ou de nouvelle ligne au sein de la fonction *SynchGetString* ce qui a amené le programme à faire fuir la chaîne précédente dans la nouvelle.

Cette erreur a été réparée en insérant les lignes 4 à 6 :

Listing 1: SynchGetString.

```
1             for(int i = 0;i<n;i++){
2                 readAvail->P();
3                 s[i]=console->GetChar();
4                 if(s[i] == EOF || s[i]=='\0' || s[i] == '\n'){
5                     break;
6                 }
7             }
```

Appel système Putchar

Suivre les étapes fournies dans la description de l'exercice a été suffisant pour mettre en place ces *syscall*

Points délicats continued on next page...

Appel système PutString

Pour cet exercice, qui à mon avis était le plus compliqué, j'ai vraiment été bloqué au niveau de la partie qui consistait à manipuler des chaînes de caractères via un petit buffer, ce qui a posé quelques problèmes.

Afin de trouver une solution à ce problème, j'ai déclaré deux constantes : *BUFFER_SIZE* et *MAXI_SIZE*.

De ce fait, lorsqu'on lit une séquence de caractères dans *BUFFER_SIZE* ou jusqu'à ce qu'on trouve une commande '\0', et si un *BUFFER_SIZE* a été lu mais qu'il n'y avait pas de '\0', le programme répète la séquence jusqu'à ce qu'il trouve un '\0'. Ou si on lit un *MAXI_SIZE*, au cas où un *MAXI_SIZE* était lu mais sans '\0', le '\0' est ajouté à la fin de la chaîne.

Pour la fonction copyStringFromMachine, j'ai décidé de la placer dans le fichier *machine.h implement in machine.c* car l'usage de cette fonction semble étroitement relié aux autres fonctions de la machine.

Appel système Exit

Pour mettre en oeuvre cet appel système comme il est demandé dans l'exercice, il a fallu copier l'appel à fonction *halt* dans l'appel système *sc_Exit* ce qui fonctionne bien dans le cas d'un process unique.

Après avoir passé quelque temps à essayer d'imaginer comment gérer ce problème, j'ai reçu le conseil de l'enseignant qui m'a dit de laisser les choses en l'état, parce que la partie dans laquelle nous devons contrôler la quantité de process et mettre en oeuvre l'appel système approprié sera traité dans l'étape suivante du projet.
Donc, l'exit syscal a simplement été mis en oeuvre de la façon suivante :

Listing 2: SC_Exit.

```
1                                   interrupt->Halt();
2                                   break;
```

En ce qui concerne la récupération de la valeur *main*, elle peut être récupérée en lisant le registre 4 après avoir modifié le fichier *start.S* pour enregistrer le registre 2 dans le registre 4 au lieu du registre 0, de la façon suivante :

Listing 3: ___start at start.S

```
1                   .globl __start
2                   .ent    __start
3               __start:
4                   jal main
5                   move    $4,$2
6                   jal Exit
7                   .end __start
```

Ainsi, la valeur du registre est enregistrée dans le registre 4, mais jusqu'à maintenant, je ne sais pas vraiment si je devrais réinscrire cette valeur dans le registre 2 comme résultat de l'appel système *Exit*.

Appel système GetChar

Pour la mise en oeuvre de cet appel système, il a été suffisant d'appeler la fonction *SynchGetChar* et d'inscrire la valeur résultat dans le registre 2.

Points délicats continued on next page...

Appel système GetString

Inspiré par l'implementation de *PutString* cet appel système appelle le *SynchGetString* puis écrit le résultat dans l'adresse de l'espace utilisateur employant l'appel de la fonction *CopyStringToMachine* ; il faut se rappeler de toujours insérer un '\0' à la fin.

Bonus printf, PutInt, GetInt

Cette partie n'a pu être mise en oeuvre, car mon binôme a été malade. J'ai donc dû travailler seul, et n'ai pas eu le temps de tout faire. Mais je vais le faire, même seul, car mon objectif est d'apprendre au maximum.

tests

Pour le test, j'ai mis en oeuvre quelques fonctions supplémenaires dans *putchar.c*. Afin de tester les appels système *GetString* et *PutString*, j'ai créé une boucle qui travaille comme le *consoleTest* sous *userprog* qui lit une chaîne de caractères et la reproduit, ce qui offre la possibilité de tester différentes longueurs de chaîne pour les deux appels système.