Rapport Devoir 1

Système d'exploitation

NACHOS : Entrées/Sorties

Guillaume NEDELEC et Alfred Aboubacar SYLLA

Rendu le : mardi 16 Octobre 2018

I. Bilan

Ce TP consiste à la mise en place de mécanismes d'entrées/sorties sur la distribution Nachos. Tout d'abord, nous avons commencé par prendre en main la version primitive d'entrées/sorties de la classe Console nous permettant d'écrire un caractère avec sa méthode `Console::PutChar(char ch)`, et d'en lire avec sa méthode `Console::GetChar()`.
Pour avoir un mécanisme d'entrée-sortie synchrone , nous avons implémenté la classe `SynchConsole` avec les méthodes :

- `SynchPutChar` : qui prend en paramètre un caractère et l'affiche à la console en utilisant PutChar de console .
- `SynchGetChar`: qui lit un caractère dans la console .
- `SynchPutString`: Permet d'écrire une chaine de caractère dans la console utilisateur en utilisant la méthode `SynchPutChar` : pour écrire caractere par caractere .
- `SynchGetString`: Permet de lire une chaine de caractère caractère par caractère depuis la console
- `SynchPutInt`: permet d'écrire un entier sur la console utilisateur à l'aide de `snprintf` et de `SynchPutString`
- `SynchGetInt`: permet de lire un entier dans la console utilisateur grâce à `sscanf` et `SynchGetString` afin de transformer l'entier en string.

Afin de tester la console synchrone, nous avons implémenté la fonction `SynchConsoleTest` dans threads/main.cc qui permet de lancer cette console avec l'option `-sc`.

Pour utiliser ces mécanismes d'entrée sortie il a fallu déclarer les appels système `PutChar`, `GetChar`, `PutString`, `GetString`, `PutInt`, `GetInt` dans le fichier `userprog/syscall.h` et les déclarer en assembleur dans le fichier `test/start.S`.

Pour finaliser ces mécanismes il a ensuite fallu implémenter les actions à effectuer lors de l'appel système. Pour ce faire nous avons rajouté des cas pour chaque appel système dans la fonction `ExceptionHandler` de `userprog/exception.cc` .

II. Points Délicats

Nous avons rencontré quelques points délicats au cour de ce TP, lors de l'implémentation de l'appel système Exit , on avait du mal à avoir la bonne valeur de retour de Exit car on avait pas changer la valeur par défaut du registre dans le fichier start.S .

Nous avons rencontré des difficultés dans l'implémentation des méthodes `copyStringFromMachine` et `copyStringToMachine` qui permettent de copier une chaine de caractère, caractère par caractère, de la machine vers notre mécanisme d'entrée sortie et inversement. Nous avons eu des problème dans la gestion de la taille du buffer `MAX_STRING_SIZE` pour copier des chaînes plus grande que ce buffer.

III. Limitations

Avantages:

- `GetString`: Nous l'avons implémenté en respectant le fait que même si size (taille en paramètre de `GetString`) est supérieur ou inférieur à `MAX_STRING_SIZE`, seul size sera retenu par la fonction.
- `GetChar`: On peut arrêter la saisie au console utilisateur par le caractère **"!" .**

Inconvénients:

- `GetInt` : l'implémentation de la méthode ne gère pas le cas où l'utilisateur met une chaine de caractère avec un entier dans la console. La méthode ne gère pas non plus les entiers supérieur à 1 milliard (environ). Dans ces cas, la valeur retournée dans ce cas sera fausse.
- `PutInt` : ne gère pas les cas où la valeur de l'entier est supérieur ou égal à ~1 000 000 000.
- l'appel système `Exit` appelle la fonction `Halt()` pour éteindre la machine au lieu de fermer le processus en cours.
- Entrées/sorties limitées à des caractères ASCII, les accents ne sont pas supportés par la classe `Console` fournie.
- la fonction bonus `printf` n'a pas été implémentée.

IV. Tests

Pour effectuer les tests, nous avons créé des programmes de tests pour chaque fonctionnalités développées. Nous avons ainsi créé les fichiers suivants : `putchar.c`, `putstring.c`, `getchar.c`, `getstring.c`, `putint.c`, `getint.c`, `exit.c`.
Pour effectuer les tests il faut lancer la commande suivante depuis le répertoire "`code/userprog`" :

```
./nachos -x ../test/<nom du programme>
```

Le fichier `putchar.c` (donné dans l'énoncé) permet d'afficher "`abcd`" puis un retour à la signe à son exécution. Cette fonction ne teste pas les limites de la fonction `PutChar()` en essayant d'écrire des caractères spéciaux par exemple.

Le fichier `putstring.c` permet d'afficher une chaîne de caractères. Nous avons décidé d'afficher d'abord une chaine de caractère de taille inférieure au buffer `MAX_STRING_SIZE`, puis après un retour à la ligne d'en afficher une autre supérieure à `MAX_STRING_SIZE` pour vérifier que l'on peut bien écrire une chaîne supérieure à la taille du buffer local.

Le fichier `getchar.c` permet de lire un caractère. Nous avons choisi de faire une boucle infinie permettant de lire un caractère avec la fonction `GetChar()` et de l'écrire juste après avec `PutChar()` pour vérifier qu'il a bien été lu. Cette boucle infini s'arrête lorsque le caractère lu est un point d'exclamation (lorsque le code ASCII est 33).

Le fichier `getstring.c` permet de lire une chaine de caractère. Cette fonction prend en compte une taille maximale de lecture. C'est à dire que si la taille maximale est de 15, la fonction `GetString()` lira au maximum 15 caractères. Tout comme la fonction `PutString()`, `GetString()` utilise un buffer local `MAX_STRING_SIZE`.

Le fichier `getint.c` permet de lire un entier sur le même principe que la fonction `GetString()` avec le stockage en tant qu'entier grâce à la fonction `sscanf()`.

Le fichier `putint.c` permet d'écrire un entier en utilisant `PutString()` en convertissant cet entier en une chaîne de caractère grâce à la fonction `snprintf()`. Nous avons choisi de passer à cette fonction un buffer de taille 100 afin de pouvoir écrire des entiers de grande taille.

Le fichier `exit.c` mais plus précisément l'appel système `SC_Exit` permet d'effectuer automatiquement la fonction `Halt()` permettant d'arrêter la machine. Cela permet de pouvoir écrire "`return 0;`" plutôt que `Halt()`, à la fin des fichiers de tests sans avoir d'erreurs.
C'est dans le fichier `code/test/start.S` avec le code :

```
__start:
     jal  main
     move $4,$2
     jal  Exit  /* if we return from main, exit(0) */
     .end __start
```

que l'appel vers `Exit` se fait automatiquement.