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 Partie 1 : Quel est le but ?

L'objectif de cette première partie est de créer un programme de test : putchar.c. Ce programme est placé dans le dossier code/test. Nous pouvons l'éxecuter depuis le dossier /code/userprog. La sortie attendue est "abcd".

```
$ ~/nachos/code/userprog ./nachos -x ../test/putchar
```

3 Partie 2 : Entrées-sorties asynchrones

Nachos offre une version primitive d'entrées-sorties par la classe Console dans machine/console.h. C'est une erreur de vouloir 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 fichiers de tests ne peuvent être plus executés à cause de la partie 4, nous ne pouvons pas instancier 2 consoles. Cependant, cette partie nous a permi de mieux prendre en main la partie console de Nachos.

4 Partie 3 : Entrées-sorties synchrones

TODO

5 Partie 4 : Appel Système Putchar

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.

Voici le code de la fonction PutChar(c), dans le fichier /test/start.S. Elle permet de lever une interruption SC_PutChar qui sera gérée par l'ExceptionHandler.

```
    .global PutChar
    .ent PutChar
PutChar:
    addiu $2, $0, SC_PutChar
    syscall
    j   $31
    .end PutChar
```

Voici le code du cas SC_PutChar de l'ExceptionHandler, dans le fichier /userprog/exception.cc. Elle récupère le caractère saisi dans le registre 4 avant de l'afficher.

```
    case SC_PutChar :
    {
        DEBUG('s', "PutChar\n");
        int ch = machine->ReadRegister(4);
        synchconsole->SynchPutChar(ch);
        break;
    }
```

Un fichier de test pour PutChar.c est disponible dans le dossier code/test/putchar.c. Pour l'éxecuter lancer la commande depuis code/userprog.
On attend "abcd" en sortie.

```
$ ~/nachos/code/userprog ./nachos -x ../test/putchar
abcd
```

6 Des caractères aux chaînes

Le but de cette 5ème partie est d'étendre le travail précédent afin de gérer des chaines de caractères.

Nous avons commencé par compléter la méthode SynchPutString. Elle prend en paramètre une chaine de caractères et va l'afficher caractère par caractère.

```
void SynchConsole::SynchPutString(const char s[]) {
    int i = 0;
    while (s[i] != '\0' && s[i] != EOF) {
      this->SynchPutChar(s[i]);
      i++;
    }
}
```

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.
Voici le code la fonction en question :

```
int copyStringFromMachine(int from, char *to, unsigned size) {
    if(size < 1)
      return 0;
    unsigned i = 0;
    int val = 0;
    do {
      machine->ReadMem(from+i, 1, &val);
      to[i] = (char)val;
      i++;
    } while(i < size && (char)val != '\0' && (char)val != EOF);
    if (i == size)
      to[i] = '\0';
    return i;
}
```

Par la suite, nous avons implémenté l'appel système PutString, utilisant copyStringFromMachine et SynchPutString. Il n'est pas raisonnable d'allouer un buffer de la même taille que la chaîne MIPS. Si un utilisateur malveillant rentrait une chaîne de taille démusurée, on se retrouverait à allouer un buffer de taille beaucoup trop importante, pouvant causer des problèmes critiques.
Voici le code l'appel PutString, dans le fichier /code/userprog/exception.cc :

```
case SC_PutString:
  {
        DEBUG('s', "PutString\n");
        int from = machine->ReadRegister(4);
        char* s = (char*)malloc(MAX_STRING_SIZE * sizeof(char));
        int n = 0, m = 0;
        do {
          n = copyStringFromMachine(from + m, s, MAX_STRING_SIZE);
          synchconsole->SynchPutString(s);
          m += n;
        } while (n == MAX_STRING_SIZE && s[n-1] != '\0');
        free(s);
        break;
  }
```

Un fichier de test est disponible pour PutString, dans le dossier /code/test/putstring.c. A lancer de façon similaire aux autres fichiers de tests. La taille de la chaîne entrée par l'utilisateur est bien plus longue que la taille du buffer MAX_STRING_SIZE. La sortie attendue est décrite en commentaire dans le fichier en question.

```
$ ~/nachos/code/userprog ./nachos -x ../test/putstring
```

7 Problèmes rencontrés

8 Conclusion