



POLYTECH' GRENOBLE

RICM 4ÈME ANNÉE

NachOS

Etape 2: Entrées/Sorties

Étudiants:
Elizabeth PAZ
Salem HARRACHE

Professeur:
Vania MARANGOZOVA

février 2012

1 Entrées/Sorties asynchrones

L'objet Console est asynchrone, dans le programme de test (ConsoleTest) on gère manuellement la synchronisation avec deux sémaphores (un pour l'écriture et un autre pour la lecture) et des handlers qui seront appelés par le traitant une fois la tâche de lecture/écriture effectuée. Dans cette partie, on ajoute la prise en compte du caractère de fin de fichier (EOF).

code/usrprog/progtest.cc

```
//..
    if (ch == EOF or ch == 'q')
        return;
//..
```

Comme on le voit, à chaque fois qu'on écrit ou lit un caractère il faut utiliser le sémaphore explicitement pour obliger le programme à se bloquer et à ne reprendre qu'une fois le caractère lu/écrit.

2 Entrées/Sorties synchrones

On va maintenant gérer la synchronisation directement dans la Console. Pour ça on va implémenter la classe SynchConsole qui utilisera de façon transparente les sémaphores. On ajoute les deux fichiers *code/userprog/synchconsole.h* et *code/userprog/synchconsole.cc*

On peut tester la synchconsole depuis le fichier *code/userprog/progtest.c*

```
void SynchConsoleTest(char *in, char*out)
{
    char ch;
    synchconsole = new SynchConsole(in, out);
    while ((ch = synchconsole->SynchGetChar()) != EOF )
        synchconsole->SynchPutChar(ch);
    fprintf(stderr, "Solaris: EOF detected in SynchConsole!\n");
}
```

3 Appels systèmes

Dans cette partie on va implémenter les appels système qui utiliseront notre synchconsole.

3.1 PutChar

3.1.1 Mise en place

On commence par déclarer la fonction d'appel système *void PutChar(char c)* et son numéro

Un appel système entraîne un changement d'environnement, du mode user au mode noyau. Il faut donc écrire (en assembleur) le code qui permet de faire cette interruption pour basculer en mode noyau et prévoir le retour au programme utilisateur :

code/test/start.S

```
.globl PutChar
.ent PutChar
PutChar:
    /* On place le signal dans le registre R2
       Il va servir au handler d'exceptions pour qu'il puisse
       savoir qui sera le traitant de cette exception.
    */
    addiu $2,$0,SC_PutChar
    /* syscall provoque un d routement et place le compteur de
       programme (PC) la premi re instruction du traitant :
       ExceptionHandler
    */
    syscall
    /* Maintenant on revient au programme appelant
       Le registre R31 sauvegarde l'adresse de retour de la
       fonction appelante
    */
    j $31
.end PutChar
```

Le traitement sera fait dans *code/userprog/exception.cc*. Pour faciliter l'ajout de nouveaux appels systèmes on utilisera un switch/case qui associe un traitement à chaque numéro d'appel système.

L'objet *synchconsole* appartient au noyau, il faut donc l'initialiser dans le fichier *code/threads/system.cc*.

Maintenant on peut utiliser *synchconsole* en important *system.h*.

3.1.2 Terminaison

Un programme est obligé d'appeler *Halt()* pour dire qu'il s'est terminé, ce qui n'est pas pratique en temps normal. Un programme est lancé par la méthode *Machine::Run()* qui ne termine pas. L'absence de *Halt()* provoque une interruption :

```
./build-origin/nachos-userprog -x ./build/putchar
a
Unexpected user mode exception 1 1
```

L'interruption 1 correspond à l'appel système *Exit()*, il faut donc implémenter cet appel système, qui se contentera, dans un premier temps, d'éteindre explicitement la machine.

3.2 PutString

La différence entre *PutChar* et *PutString* c'est que *PutString* prend un pointeur sur une chaîne de caractères en mémoire **user**. On va donc devoir par précaution, préalablement la copier dans un buffer en mémoire **noyau**.

La mise en place de l'appel système est similaire à *PutChar()*. Il faut simplement spécifier la taille du buffer de copie.

On utilise un mutex (Sémaphore initialisé à 1) pour assurer l'atomicité de *PutString*. En effet, on souhaite avoir tous les caractères dans le bon ordre, et deux appels à *PutString* doivent se faire l'un après l'autre.

3.3 GetChar GetString

Ces deux appels systèmes sont symétriques à *PutChar* et *PutString*. Dans le cas de *GetChar*, rien de plus simple, étant donné qu'il renvoie directement la valeur (C'est le registre 2 qui est utilisé pour les valeurs de retour). On écrit directement dans le registre la valeur de retour de *SynchGetChar()*.

Pour *GetString*, on écrit dans un buffer intermédiaire, puis on copie ce buffer dans la mémoire *user* à l'adresse donnée à l'appel système.

3.4 PutInt et GetInt

Pour *PutInt* et *GetInt* on va utiliser les fonctions **sscanf** et **snprintf**. Pour faciliter la saisie, on ajoute la fonction *SynchConsole::SynchGetString(char *buffer, int n, char delim)* qui permet de lire une chaîne de caractères et de s'arrêter dès qu'on rencontre un délimiteur (*delim*). Dans notre cas, on va utiliser `'\n'` comme délimiteur lors de la saisie de nombres entiers.

4 Test Nachos étape 2

Voici le programme de test :

```
#include "syscall.h"

int main() {
    PutString("Veuillez saisir un nombre : \n");
    int nombre = GetInt();
    PutString("Nombre +10 = "); PutInt(nombre+10); PutChar('\n');
    PutString("Veuillez saisir une lettre : \n");
    char c = GetChar();
    PutString("Voici la lettre : ");
    PutChar(c);
    PutString("\nVeuillez saisir une phrase (max = 100) : ");
    char buffer[100];
    GetString(buffer,100);
    PutString("\nVoici la phrase : ");
    PutString(buffer);
    PutChar('\n');
    return 0;
}
```

```
$ ./build-origin/nachos-userprog -x ./build/etape2
Veuillez saisir un nombre :
5
Nombre +10 = 15
Veuillez saisir une lettre :
g
Voici la lettre : g
Veuillez saisir une phrase (max = 100) : le mot de la fin

Voici la phrase :
le mot de la fin

Machine halting!

Ticks: total 1136142117, idle 1136139819, system 2170, user 128
Disk I/O: reads 0, writes 0
Console I/O: reads 22, writes 174
Paging: faults 0
Network I/O: packets received 0, sent 0

Cleaning up...
```