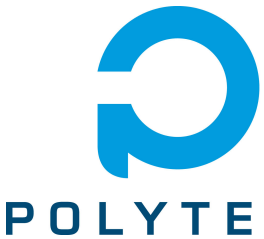

Polytech' Grenoble

RICM 4ème année

NachOS
Etape 2: Entrées/Sorties Console

Étudiants:
Paul LABAT
Rodolphe FREBY
Kai GUO
Zhengmeng ZHANG

Professeur:
Jean-françois MEHAUT

Février 2014

1 Entrées/Sorties asynchrones

L'objet Console est asynchrone, dans le programme de test (ConsoleTest) on gère manuellement la syncronisation 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

```
void ConsoleTest (char *in, char *out)
{
    char ch;
    delete synchconsole;
    console = new Console (in, out, ReadAvail, WriteDone, 0);
    readAvail = new Semaphore ("read avail", 0);
    writeDone = new Semaphore ("write done", 0);

    for (;;)
      {
      readAvail->P ();  // wait for character to arrive
      ch = console->GetChar ();

      if(ch == EOF){
        synchconsole = new SynchConsole(NULL, NULL);
        break;
      }

      if (ch != '\n'){  // On affiche pas les caracteres retour a la ligne et end of file
        console->PutChar ('<');
        writeDone->P ();    // Attente de la fin du write du <
        console->PutChar (ch);  // echo it!
        writeDone->P ();    // wait for write to finish
        console->PutChar ('>');
        writeDone->P ();    // Attente de la fin du write du >
      }else{
        console->PutChar('\n');
        writeDone->P ();
      }

      if (ch == 'q' || ch == EOF)
          return;       // if q, quit
      }
}
```

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

code/userprog/synchconsole.h

```
#ifndef SYNCHCONSOLE_H
#define SYNCHCONSOLE_H
#include "copyright.h"
#include "utility.h"
#include "console.h"
class SynchConsole {

    public:

    SynchConsole(char *readFile, char *writeFile);  // initialize the hardware console device

    ~SynchConsole();    // clean up console emulation

    void SynchPutChar(const char ch);   // Unix putchar(3S)
    char SynchGetChar();    // Unix getchar(3S)

    void SynchPutString(const char *s); // Unix puts(3S)
    void SynchGetString(char *s, int n);    // Unix fgets(3S)

    private:
    Console *console;
```

```
};
#endif // SYNCHCONSOLE_H
```

code/userprog/synchconsole.cc

```
#include "copyright.h"
#include "system.h"
#include "synchconsole.h"
#include "synch.h"
#include "console.h"


static Semaphore *readAvail;
static Semaphore *writeDone;

static void ReadAvail(int arg) { readAvail->V(); }
static void WriteDone(int arg) { writeDone->V(); }

SynchConsole::SynchConsole(char *readFile, char *writeFile)
{
    readAvail = new Semaphore("read avail", 0);
    writeDone = new Semaphore("write done", 0);
    console = new Console(readFile, writeFile, ReadAvail, WriteDone, 0); // Completer, initialisation des ↩
        sempahores a 0
}

SynchConsole::~SynchConsole()
{
    delete console;
    delete writeDone;
    delete readAvail;
}

void SynchConsole::SynchPutChar(const char ch)
{
    if(ch != '\n'){
        console->PutChar('<');
        writeDone->P();
    }
    //semPutChar->P();

    console->PutChar(ch);
        writeDone->P();
    //semPutChar->V();

    if(ch != '\n'){
        console->PutChar('>');
            writeDone->P();
    }
}

char SynchConsole::SynchGetChar()
{
    readAvail->P ();
    return console->GetChar ();

}

void SynchConsole::SynchPutString(const char s[]){
}

void SynchConsole::SynchGetString(char *s, int n){
}
```

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

```
void
SynchConsoleTest (char *in, char *out)
{
char ch;
    delete synchconsole;
    synchconsole = new SynchConsole(in, out);
    while ((ch = synchconsole->SynchGetChar()) != EOF ){
            synchconsole->PutChar(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
code/userprog/syscall.h

```
#define SC_PutChar  11
// ...
void PutChar(char c);
```

Un appel système entraine un changement d'environnement, du mode 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:
    addiu $2,$0,SC_PutChar
    syscall
    j     $31
    .end PutChar
```

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

```
void ExceptionHandler(ExceptionType which)
{
    int type = machine->ReadRegister(2);
    #ifndef CHANGED // Noter le if*n*def
    if ((which == SyscallException) && (type == SC_Halt)) {
        DEBUG('a', "Shutdown, initiated by user program.\n");
        interrupt->Halt();
    } else {
        printf("Unexpected user mode exception %d %d\n", which, type);
        ASSERT(FALSE);
    }
    #else // CHANGED
    if (which == SyscallException) {
        switch (type) {
            case SC_Halt: {
                DEBUG('a', "Shutdown, initiated by user program.\n");
                interrupt->Halt();
                break;
            }
            case SC_Exit: {
                break;
            }
            case SC_PutChar: {
                int lecture = machine->ReadRegister (4); // on lis le registre 4 ! c'est un int
                synchconsole->SynchPutChar ((char) lecture); // affichage
                break;
            }
        default: {
          printf("Unexpected user mode exception %d %d\n", which, type);
          ASSERT(FALSE);
        }
      }
    }
    #endif // CHANGED
    UpdatePC ();
}
```

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

```
#ifdef USER_PROGRAM         // requires either FILESYS or FILESYS_STUB
Machine *machine;         // user program memory and registers
SynchConsole *synchconsole; // On declare la synchconsole
#endif
// [..]
void Initialize (int argc, char **argv)
{
// ...
#ifdef USER_PROGRAM
machine = new Machine (debugUserProg);     // this must come first
synchconsole = new SynchConsole(NULL, NULL); // On declare la synchconsole sans aucun in/out
#endif
// ..
}

void Cleanup ()
{
// ..
#ifdef USER_PROGRAM
delete machine;
delete synchconsole;
#endif
// ..
}
```

code/threads/system.h

```
#ifdef USER_PROGRAM
#include "machine.h"
#include "synchconsole.h"
#define MAX_STRING_SIZE 50
extern Machine *machine;     // user program memory and registers
extern SynchConsole *synchconsole;
#endif
```

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

3.1.2 Test de PutChar()

code/userprog/putchar.c

```
#include "syscall.h"

int main() {
    PutChar('a');
    PutChar('\n');
    Halt();
}
```

```
$ ./nachos-userprog -x putchar

<a>
Machine halting!

Ticks: total 473, idle 400, system 50, user 23
Disk I/O: reads 0, writes 0
Console I/O: reads 0, writes 4
Paging: faults 0
Network I/O: packets received 0, sent 0

Cleaning up...
```

3.1.3 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 :

code/machine/mipssim.cc

```
void Machine::Run()
{
```

```
    // [...]
    interrupt->setStatus(UserMode);
    for (;;) {
        OneInstruction(instr);
        interrupt->OneTick();
        if (singleStep && (runUntilTime <= stats->totalTicks))
            Debugger();
    }
}
```

L'absence de *Halt()* provoque une interruption :

```
./nachos-userprog -x 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:
code/userprog/exception.cc

```
        case SC_Halt: {
          DEBUG('a', "Exit, initiated by user program.\n");
          interrupt->Halt();
          break;
        }
```

3.2 PutString

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

```
void copyStringFromMachine(int from, char *to, unsigned size)
{
    unsigned i = 0;
    int tmp;
    for(i = 0; i < size ; i++){
        if(machine->ReadMem(from + i, 1, &tmp))
        to[i]=tmp;
    }
    //si le message ne se fini pas par '\0'...
    if(i<size && tmp != '\0'){
        to[size-1] = '\0';
    }
}
```

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

```
        case SC_SynchPutString: {
                char *buffer = new char[MAX_STRING_SIZE];
                int recup = machine->ReadRegister(4);
                copyStringFromMachine(recup, buffer, MAX_STRING_SIZE);
                synchconsole->SynchPutString(buffer);
                delete [] buffer;
                break;
            }
```

Dans la SynchConsole on implémente *SynchConsole::SynchPutString* :
code/userprog/synchonsole.cc

```
void SynchConsole::SynchPutString(const char s[])
{
    int i;
        for(i=0;i<MAX_STRING_SIZE && s[i]!='\0';i++) {
                synchconsole->SynchPutChar((char)s[i]);
        }
        synchconsole->SynchPutChar('\n');
}
```

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 renvoit 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().

code/userprog/exception.cc

```
            case SC_SynchGetChar: {
                char recup = synchconsole->SynchGetChar();
                machine->WriteRegister(2, (int)recup);
                break;
            }
```

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

```
            case SC_SynchGetString: {
                char *buffer = new char[MAX_STRING_SIZE];
                int recup = machine->ReadRegister(4);
                int taille = machine->ReadRegister(5);
                synchconsole->SynchGetString(buffer, taille);
                copyStringToMachine(buffer, recup, taille);
                delete buffer;
                break;
            }
```

```
void copyStringToMachine(char *from, int to, unsigned int size)
{
  int tmp;
  unsigned int i;
  for(i = 0; i < size - 1; i++){
    tmp = from[i];
    machine->WriteMem(to + i, 1, tmp);
  }
  tmp = '\0';
  machine->WriteMem(to + i, 1, tmp);
}
```

code/userprog/synchonsole.cc

```
char SynchConsole::SynchGetChar()
{
    readAvail->P ();
    return console->GetChar ();
}

void SynchConsole::SynchGetString(char *s, int n)
{
    int i;
    char c;
    // attention a ne pas depasser la taille limite...
    for(i=0; i < n-1; i++){
        c = synchconsole->SynchGetChar();
        if(c == EOF || c == '\n'){
            break;
        }else{
            s[i] = c;
        }
    }
    s[i] = '\0';
}
```

3.4 PutInt et GetInt

Pour PutInt et GetInt on va utiliser les fonctions **sscanf** et **snprintf**. Pour facilier la saisie, on ajoute la fonction *SynchConsole::SynchGetString(char *buffer, int n, char delim)* qui permet de lire une chaine 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 :

code/userprog/synchonsole.cc

```
void SynchConsole::SynchPutInt(int n)
{
    char *s = new char[MAX_STRING_SIZE];
    snprintf(s, MAX_STRING_SIZE, "%d", n);
    synchconsole->SynchPutString(s);
    delete [] s;
}

void SynchConsole::SynchGetInt(int *n)
{

    int retour;
    // On travail sur des entiers...
    char *conversion = new char[12];
    SynchGetString(conversion, 12);
    sscanf(conversion, "%d", &retour);
    machine->WriteMem(*n, 4, retour);
    delete [] conversion;
}
```

code/userprog/exception.cc

```
            case SC_SynchPutInt: {
                int recup = machine->ReadRegister(4);
                synchconsole->SynchPutInt(recup);
                break;
            }
            case SC_SynchGetInt: {
                int *recup = new int;
                *recup = machine->ReadRegister(4);
                synchconsole->SynchGetInt(recup);
                delete recup;
                break;
            }
```

4 Test Nachos étape 2

Voici les programmes de test : *code/test/getChar.c*

```
#include "syscall.h"

int
main()
{
    char s = SynchGetChar();
    PutChar(s);
    return 0;
}
```

```
$ ./nachos-userprog -x getChar
a
<a>Machine halting!

Ticks: total 217479977, idle 217479891, system 50, user 36
Disk I/O: reads 0, writes 0
Console I/O: reads 2, writes 3
Paging: faults 0
Network I/O: packets received 0, sent 0

Cleaning up...
```

code/test/getString.c

```
#include "syscall.h"

int
main()
{
    char s [512];
    SynchGetString(s,12);
    SynchPutString(s);
    return 0;
}
```

```
$ ./nachos-userprog -x getString
abc
<a><b><c>
Machine halting!

Ticks: total 271282445, idle 271282258, system 150, user 37
Disk I/O: reads 0, writes 0
Console I/O: reads 4, writes 10
Paging: faults 0
Network I/O: packets received 0, sent 0

Cleaning up...
```

code/test/getint.c

```
#include "syscall.h"

int
main()
{
    int s;
    SynchGetInt(&s);
    SynchPutInt(s);
    return 0;
}
```

```
$ ./nachos-userprog -x getint
123
<1><2><3>
Machine halting!
```

```
Ticks: total 479159846, idle 479159659, system 150, user 37
Disk I/O: reads 0, writes 0
Console I/O: reads 4, writes 10
Paging: faults 0
Network I/O: packets received 0, sent 0

Cleaning up...
```