Compte Rendu
NACHOS

BARTHELEMY Romain, EUDES Robin, MORISON Jack, ROSSI Ombeline

7 mars 2015

Table des matières

1 Introduction

Ce projet a été réalisé dans le cadre de notre 4ème année d'étude à Polytech, avec la spécialisation "systèmes et réseaux". En réalisant ce projet, nous avons put mettre en pratique l'ensemble des connaissances engrangés au cours de nos parcours et ainsi comprendre les concepts entrant en jeu lors de la réalisation d'un système d'exploitation.

Dans un premier temps, nous allons nous intéresser à la gestion de l'affichage de chaines de caractères de façon synchrone par le système. Nous allons également nous intéresser à la mise en place d'un appel système (syscall). Par la suite, nous nous interesserons à la gestion des threads utilisateur, au multithreading.

2 Étape 2

2.1 Entrées-sorties asynchrones

Une version élémentaire de gestion des entrées-sorties nous est fournie par NachOS, au travers de la classe *Console*. Le code fournit effectue une gestion asynchrone des entrées-sorties. Nous devons donc gérer la synchronisation grâce à deux sémaphores (pour gérer l'écriture et la lecture) ainsi que deux handlers. Ceux-ci libèreront le sémaphore et nous informeront de la fin de l'opération de lecture/écriture. Ainsi, la synchronisation est assurée par ce mécanisme.

Voici un extrait de code permetant cette gestion des entrées/sorties. Si le caractère est EOF, la machine s'arrête : syscall Halt() :

```
void ConsoleTest (char *in, char *out){
    char ch;
    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 ();

          #ifdef CHANGED
          if(ch!='\n' && ch!=EOF){
                  console->PutChar ('<');
                  writeDone->P ();
          }
          #endif

          // Original code
          #ifndef CHANGED
          console->PutChar (ch);
          writeDone->P (); // wait for write to finish
          if (ch == 'q')
                  return;
          #else

          // Now, we prefer to exit on EOF,
          // only if it's at the begin of a new line.
          if(ch!=EOF){
                  console->PutChar (ch);
                  writeDone->P ();
                  if(ch!='\n'){
                          console->PutChar ('>');
                          writeDone->P ();
                  }
          }
          else{
                return;
          }
          if (ch=='\0'){ // EOT
                  return;
          }
      #endif
      }
}
```

2.2 Entrées-sorties synchrones

Nous devons maintenant créer une classe *SynchConsole* afin de réaliser les opérations de synchronisation d'entrées/sorties automatiquement :

```
static Semaphore *readAvail;
static Semaphore *writeDone;
static Semaphore *SemPutChar;
static Semaphore *SemGetChar;
static Semaphore *SemPutString;
static Semaphore *SemGetString;

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);

        SemPutChar = new Semaphore("PutChar", 1);
        SemGetChar = new Semaphore("GetChar", 1);
        SemPutString = new Semaphore("PutString", 1);
        SemGetString = new Semaphore("GetString", 1);
}

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

void SynchConsole::SynchPutChar(const char ch){
        SemPutChar->P();
        console->PutChar (ch);
        writeDone->P ();
        SemPutChar->V();
}

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

Le test de ces méthodes est réalisé dans *progtest.cc* par la fonction *SynchConsoleTest*.

```
 void SynchConsoleTest (char *in, char *out){
        char ch;
        SynchConsole *synchconsoletest = new SynchConsole(in, out);

        while ((ch = synchconsoletest->SynchGetChar()) != EOF){
                if(ch!='\n'){
                        synchconsoletest->SynchPutChar('<');
                        synchconsoletest->SynchPutChar(ch);
                        synchconsoletest->SynchPutChar('>');
                }
                else{
                        synchconsoletest->SynchPutChar(ch);
                }
        }
        fprintf(stderr, "Solaris: EOF detected in SynchConsole!\n");
}
```

Note : Chaque caractère est par ailleurs encadré par < >

Le fichier *main.cc* est modifié pour prendre en compte l'option -sc qui permetra l'éxecution de la console synchrone. Initialement, la création de la console était effectuée dans *system.cc , fonction Initilize*, mais suite à des erreurs rencontrées dans les phases de test, l'instanciation de SynchConsole a été déplacée dans le main.

```
#ifdef CHANGED
...
else if (!strcmp (*argv, "-sc")){
        if (argc == 1)
            SynchConsoleTest (NULL, NULL);
        else
          {
              ASSERT (argc > 2);
              SynchConsoleTest (*(argv + 1), *(argv + 2));
              argCount = 3;
          }
        interrupt->Halt ();
}
#endif // CHANGED
```

Deplus, la fonction *Cleanup() , fichier exception.cc* est modifiée, pour suprimer cette nouvelle console lors de l'arrêt de la machine :

```
...
#ifdef CHANGED
    delete synchconsole;
#endif //CHANGED
...
```

2.3 Création d'un syscall, Putchar

Pour réaliser cet appel système, nous modifions *syscall.h*, afin d'y ajouter putchar :

```
 #define SC_PutChar              11
 ...
/* Print the character c on the terminal */
void PutChar(char c);
...
```

Le syscall est ensuite définit dans *start.S* (en assembleur), en nous inspirant des syscall existants :

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

Le syscall *PutChar* défini, il nous reste à mettre en place de handler qui se chargera de la prise en compte des exceptions relatives à PutChar (fichier *exception.cc, fonction ExceptionHandler*) :

```
if (which == SyscallException){
        switch(type){
                case SC_Halt:{
                        DEBUG ('a', "Shutdown, initiated by user program.\n");
                        interrupt->Halt ();
                        break;
                }
                case SC_PutChar:{
                        int c = machine->ReadRegister (4);
                        synchconsole->SynchPutChar((char)c);
                        break;
                }
                ...
                default:{
                        printf ("Unexpected user mode exception %d %d\n", which,
                            type);
                        ASSERT (FALSE);
                }
        }
}
```

2.4 La manipulation de chaine de caractères

La manipulation des string nous permet d'étudier les spécificités de la simulation d'un système d'exploitation par NachOS. En effet, nous devons jongler entre 2 espaces mémoire : l'espace MIPS (NachOS) et l'espace Linux.

```
// Used for SynchPutString
// get string from mips memory space , put it in linux memory space
void copyStringFromMachine( int from, char *to, unsigned size){
        unsigned int i;
        int tmp;
        for(i=0;i<size;i++){
                if(machine->ReadMem(from+i,1,&tmp))
                to[i]=tmp;
        }
        if(tmp!='\0'){
                to[size-1]='\0';
        }
}
// Used for SynchGetString
// get string from linux memory space, put it to mips memory space
void copyStringToMachine( char *from, int to, unsigned size){
        unsigned int i;
        int tmp;
        for(i=0;i<size-1;i++){
                tmp=from[i];
                machine->WriteMem(to+i,1,tmp);
        }
        tmp='\0';
        machine->WriteMem(to+i,1,tmp);
}
```

Nous devons ensuite ajouter les syscall associés SynchPutString et SynchGetString (*start.S*) :

```
//...
 SynchPutString:
        addiu $2,$0,SC_SynchPutString
        syscall
        j       $31
        .end SynchPutString

        .globl SynchGetChar
        .ent    SynchGetChar
//...
SynchGetString:
        addiu $2,$0,SC_SynchGetString
        syscall
        j       $31
        .end SynchGetString

        .globl SynchPutInt
        .ent    SynchPutInt
// ...
```

Et mettre en place les Handler associés, comme pour les précédants appels système. (fichier *exception.cc , fonction ExceptionHandler*) :

```
 ...
case SC_SynchPutString:{
        char *buffer=new char[MAX_STRING_SIZE];
        int s = machine->ReadRegister (4);
        copyStringFromMachine(s, buffer, MAX_STRING_SIZE);
        synchconsole->SynchPutString(buffer);
        delete buffer;
        break;
}
...
case SC_SynchGetString:{
        char *buffer=new char[MAX_STRING_SIZE];
        int s = machine->ReadRegister (4);
        int size = machine->ReadRegister (5);
        synchconsole->SynchGetString(buffer,size);
        copyStringToMachine(buffer, s, size);
        delete buffer;
        break;
}
```

Note : MAX_STRING_SIZE , SC_SynchPutString, SC_SynchGetString sont définis dans system.h

Enfin, les fonctions SynchPutString et SynchGetString sont définies dans SynchConsole.cc :

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

void SynchConsole::SynchGetString(char *s, int n){
SemGetString->P();
char c;
int i;

c = synchconsole->SynchGetChar ();
if(c==EOF || c=='\n'){
        s[0]='\0';
        SemGetString->V();
        return;
}
else
        s[0] = c;
for (i=1;i<n;i++){
        c = synchconsole->SynchGetChar ();
        if(c==EOF && s[i-1]=='\n')
                break;
        else{
                if(c==EOF)
                        i--;
                else
                        s[i] = c;
        }
}
s[i]='\0';
SemGetString->V();
}
```

La méthode SynchGetString est un peu plus complexe que SynchPutString car nous devons maintenir un comportement : Si EOF est vu en début de ligne, on termine la console, sinon ce dernier est ignoré (comme dans un système Linux).

3 Étape 3

3.1 Thread Nachos

3.2 Thread Utilisateur