Compte Rendu
NACHOS

EUDES Robin, ROSSI Ombeline, BARTHELEMY Romain, MORISON Jack

7 mars 2015

Table des matières

1 Introduction **2**

2 Étape 2 **3**
2.1 Entrées-sorties asynchrones 3
2.2 Entrées-sorties synchrones 4
2.3 Le syscall PutChar 4
2.4 La fonction SynchPutString 4
2.5 Le syscall Halt 4
2.6 Les Fonctions de lecture/écriture 4
2.7 Jeu de test 4
2.8 Partie II : Entrées-sorties asynchrones 4
2.8.1 Observation de progtest.cc 4
2.8.2 Modifications de progtest.cc 4
2.9 Entrées-sorties synchrones 5

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

2.3 Le syscall PutChar

2.4 La fonction SynchPutString

2.5 Le syscall Halt

2.6 Les Fonctions de lecture/écriture

2.7 Jeu de test

A l'exécution de putchar.c, on s'attends à avoir le retour suivant :

```
$ ./nachos -x ./putchar
abcd
[...] # retour du syscall halt
```

2.8 Partie II : Entrées-sorties asynchrones

2.8.1 Observation de progtest.cc

```
 ./nachos-userprog -c
test
test # retour de la console
arret si on tape sur q et rien ensuite
arret si on tape sur qMachine halting! # retour de la console
[...]
```

La console se ferme sur la lecture du caractère "q" , et ignore tout ce qu'on a put saisir ensuite. (#Action II.1)

2.8.2 Modifications de progtest.cc

Trace d'exécution :

```
 $ ./nachos-step2 -c
test
<t><e><s><t>


test<t><e><s>< # test suivi de ^D
t> # dernier caractère qui s'affiche après entréé
Machine halting! # ^D en debut de ligne bien pris en compte
```

Après quelques modifications, la terminaison de la console s'effectue sur fin de fichier ou, sur un tty, D̂ en début de ligne. Cependant, nous notons un léger disfonctionement suite à nos modifications : lorsque l'on effectue un D̂ en fin de ligne, le dernier caractère tapé ne sera pas affiché avant que l'on appuie sur entrée. Ce problème se manifeste uniquement dans ce cas, nous pensons à un soucis de buffer. (#Action II.2)

Par ailleurs, on peut observer que chaque caractère est bien encadré de < > . (# Action II.3)

Le test avec un fichier d'entrée et de sortie fonctionne correctement. (#Action II.4)

2.9 Entrées-sorties synchrones

Nous allons maintenant dévelloper une console synchrone. Dans un premier temps, nous copions un squelette fourni. (#Action III.1)

Ensuite, les fonctions de manipulations des caractères sont complétées :

```
 void SynchConsole::SynchPutChar(const char ch)
{
  SemPutChar->P();
  console->PutChar (ch);
  writeDone->P (); // wait for write to finish
  SemPutChar->V();
}

char SynchConsole::SynchGetChar()
{
  SemGetChar->P();
  char ch;
  readAvail->P (); // wait for character to arrive
  ch = console->GetChar ();
  SemGetChar->V();
  return ch;
}
```

La manipulation de String n'est rien de plus qu'une itération d'appels aux fonctions de manipulation des char, après tout, une string n'est qu'une suite de caractères. Explications du fonctionnement sur PutChar :

Dans un premier temps, on prend le sémaphore afin d'assurer qu'un seul caractère est traité à la fois (section critique). L'opération "put" est ensuite effectuée. On attend que le write se termine pour libérer le sémaphore de la section critique, grâce à un second sémaphore (writeDone), qui sera libéré l'écriture effectuée.

On retrouve ce principe sur la fonction getchar.

Par ailleurs, on peut observer quelques subtilitées dans la fonction getString, afin de conserver un fonctionnement cohérent : prise en compte du D̂ uniquement en début de ligne, entre autres. (#Action III.2)