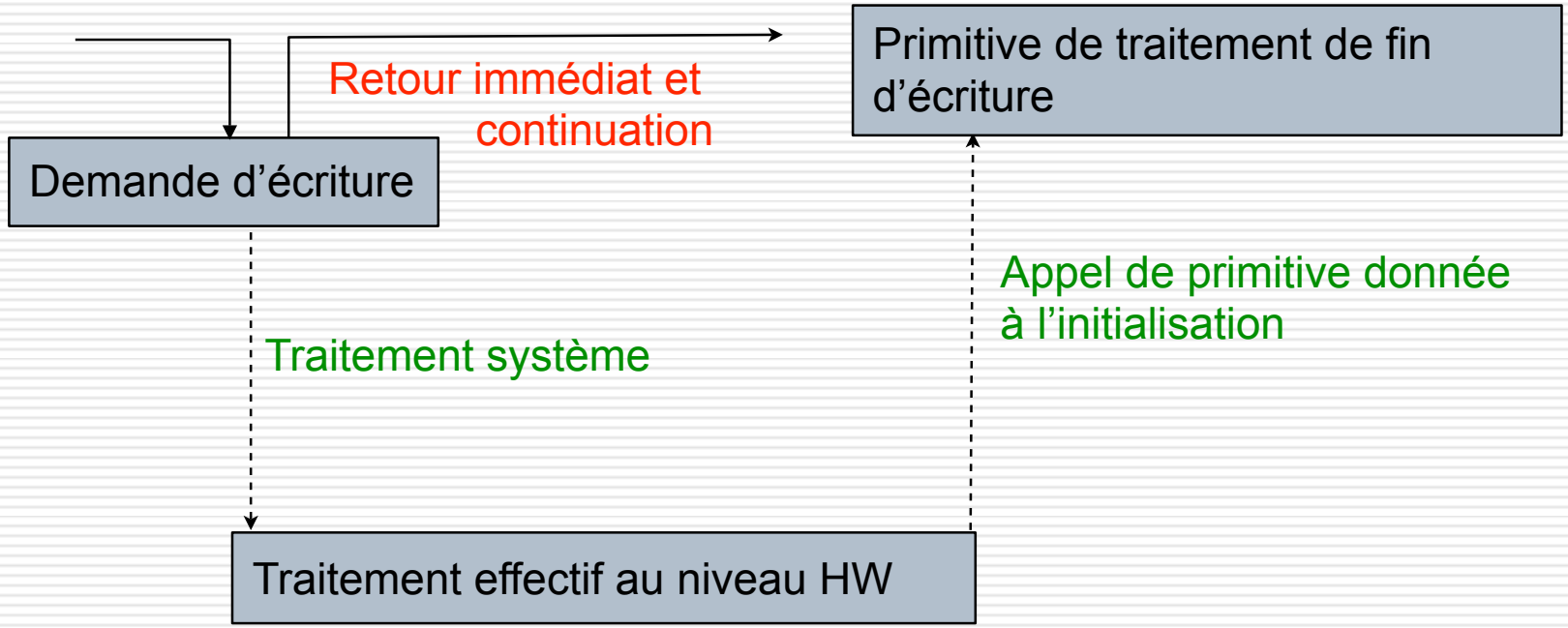
NachOS

Etape 2 : Entrées/Sorties Console

La console asynchrone

- NachOS fournit une console **asynchrone**
  - classe `Console` (`machine/console.h/c`)
  - méthodes `PutChar`, `GetChar`
- E/S asynchrones => le processus qui demande l'opération n'est pas bloqué pendant le traitement mais est informé à la fin



machine/console.h/c

```
   // console.h
//Data structures to simulate the behavior of a terminal
//I/O device.  A terminal has two parts -- a keyboard input,
//and a display output, each of which produces/accepts
//characters sequentially.
//
//The console hardware device is asynchronous.  When a character
//is written to the device, the routine returns immediately, and
//an interrupt handler is called later when the I/O completes.
//For reads, an interrupt handler is called when a character
// arrives.
//
//The user of the device can specify the routines to be called whe
//the read/write interrupts occur.  There is a separate interrupt
//for read and write, and the device is "duplex" -- a character
//can be outgoing and incoming at the same time.
```

Les primitives à appeler

- L'initialisation de la console prend deux paramètres qui sont les deux méthodes à appeler lors de la fin d'écriture et de lecture

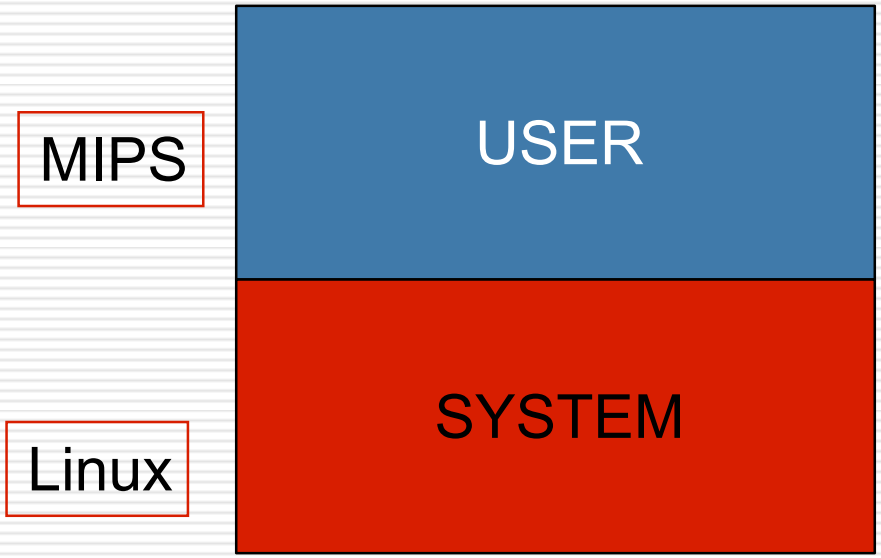
```
static Console *console;
static Semaphore *readAvail;
static Semaphore *writeDone;
static void ReadAvail(int arg) { readAvail->V(); }
static void WriteDone(int arg) { writeDone->V(); }
console = new Console(in, out, ReadAvail, WriteDone, 0);
```

- Synchroniser une lecture
  - `readAvail->P();// wait for character to arrive`
  - `ch = console->GetChar();`
- Synchroniser une écriture
  - `console->PutChar(ch); // echo it!`
  - `writeDone->P() ; //wait for character to go`

La console synchrone

- classe `SynchConsole`
- méthodes
  - `SynchPutChar`
  - `SynchGetChar`
  - `SynchPutString` : suite de caractères
  - `SynchGetString` : suite de caractères
- Une fois cette classe implémentée, NA PAS Y TOUCHER!
  - elle marche, le test -sc marche, pourquoi rajouter des traitements dedans?
- Dans quel mode s'exécutent les opérations de la synch console?

Les appels système



programmes user, dans ./test
Les appels système sont déclarés dans:
userprog/syscall.h

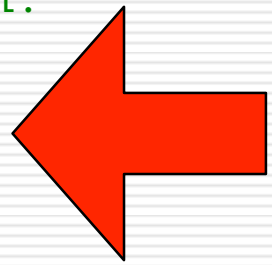
Extrait de userprog/syscall.h

```
    // LB: This part is read only on compiling the test/*.c files.
// It is *not* read on compiling test/start.S


/* The system call interface.  These are the operations the Nachos
 * kernel needs to support, to be able to run user programs.
 *
 * Each of these is invoked by a user program by simply calling the
 * procedure; an assembly language stub stuffs the system call code
 * into a register, and traps to the kernel.  The kernel procedures
 * are then invoked in the Nachos kernel, after appropriate error
checking,
 * from the system call entry point in exception.cc.
 */

/* Stop Nachos, and print out performance stats */
void Halt ();
```

Mais où est donc l'implém???

- Le corps des fonctions (appels système) est donné en assembleur dans test/start.S

```
/* ----------------------------------------------------------
 * System call stubs:
 * Assembly language assist to make system calls to the Nachos kernel.
 * There is one stub per system call, that places the code for the
 * system call into register r2, and leaves the arguments to the
 * system call alone (in other words, arg1 is in r4, arg2 is
 * in r5, arg3 is in r6, arg4 is in r7)
 *
 *      The return value is in r2. This follows the standard C calling
 *      convention on the MIPS.
 * ----------------------------------------------------------
 */
    .globl Halt
    .ent  Halt
Halt:
    addiu $2,$0,SC_Halt
    syscall
    j     $31
    .end Halt
```



Le code SC_Halt

- Dans syscall.h

```
/* system call codes -- used by the stubs to tell the
kernel which system call
 * is being asked for
 */
#define SC_Halt     0
#define SC_Exit     1
#define SC_Exec     2
#define SC_Join     3
#define SC_Create   4
...
Vous pouvez/devez rajouter des codes pour les appels que
vous allez implémenter.
```

Le déroulement d'un appel système

```
.globl Halt
   .ent Halt
Halt:
   addiu $2,$0,SC_Halt
   syscall
   j    $31
   .end Halt
```

instruction spéciale provoquant une interruption logicielle

interprétation dans `Machine::OneInstruction`

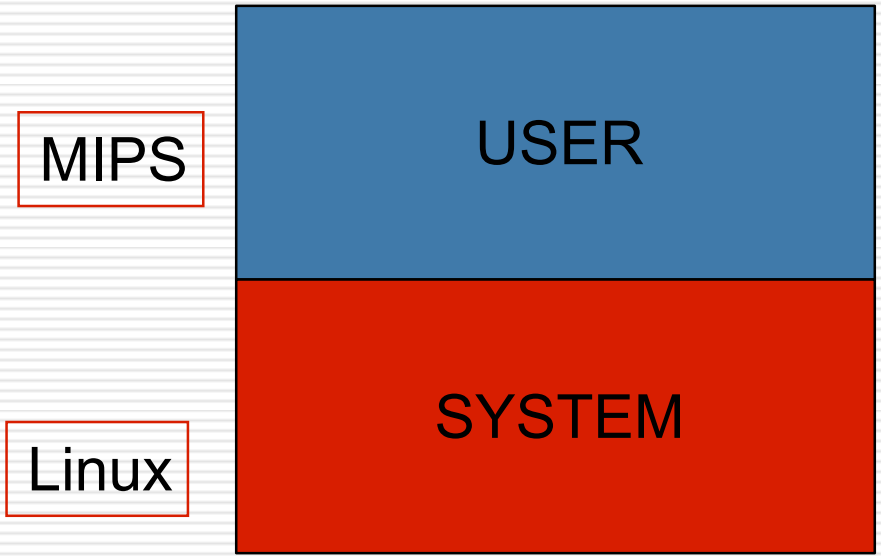
Machine::OneInstruction (machine/mipssim.cc)

```
    //----------------------------------------------------------------------
    ---
// Machine::OneInstruction
//      Execute one instruction from a user-level program
//
//      If there is any kind of exception or interrupt, we invoke the
// exception handler, and when it returns, we return to Run(), which
// will re-invoke us in a loop.  This allows us to
// re-start the instruction execution from the beginning, in
// case any of our state has changed.  On a syscall,
//      the OS software must increment the PC so execution begins
//      at the instruction immediately after the syscall.
//
void Machine::OneInstruction(Instruction *instr) {
...
 switch (instr->opCode) {
    ...
    case OP_SYSCALL:
     RaiseException(SyscallException, 0);
    return; ...
```

Interruption Hander : userprog/exception.cc

```
    void ExceptionHandler (ExceptionType which)
{
  int type = machine->ReadRegister (2);
  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);
  }
  // LB: Do not forget to increment the pc before returning!
  UpdatePC ();
  // End of addition
}
```

Alors, appel système PutChar?

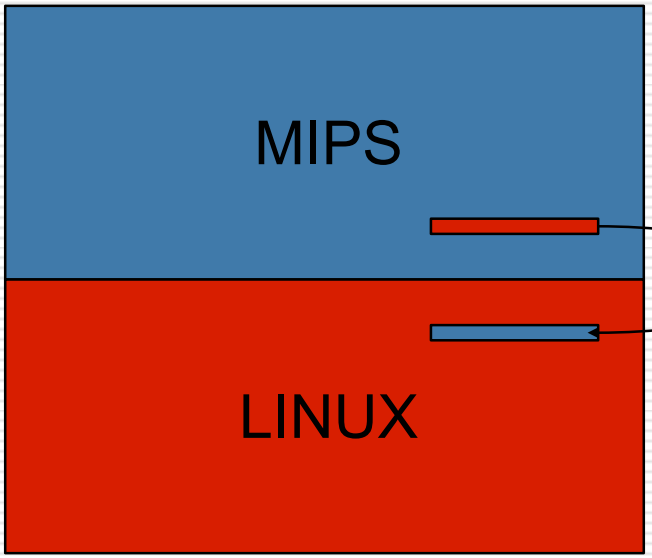


programmes user, dans ./test

appels système
SynchPutChar,
SynchGetChar,
SynchPutString,
SynchGetString

primitives des consoles
SynchConsole:PutChar
…

Et le `copyStringFromMachine`?



Appel de
PutString(pointeur MIPS)

`copyStringFromMachine`

Appel de
SynchPutString(pointeur Linux)

il faut bien un tampon, avec une certaine taille…
Lire la mémoire MIPS-> utiliser les primitives dédiées -> écrire dans tampon Linux

Erreurs courantes

- Plusieurs consoles à l'exécution de NachOS
  - S'assurer que pendant l'exécution il n'y a qu'une console asynchrone et qu'une console synchrone qui s'appuyie sur la console asycnrone
  - Les tests -c et -sc doivent marcher et se terminer proprement sans Ctrl-C
  - Que se passe-t-il si la chaîne à écrire est trop longue?
- Ctrl-D == EOF != ÿ
  - EOF est un entier
  - ÿ est un char
  - attention aux conversions qui pourraient faire que vous les confondiez