Sistemas Operativos

Nachos: Syscalls

Septiembre 2006

Profesor Jonathan Makuc // jmakuc@gmail.com

Índice

1. Introducción

1.1. Generalidades

En el presente documento se detallara el funcionamiento de las llamadas al sistema de NachOS, y el procedimiento que se debe realizar para agregar syscalls.

1.2. Syscall, llamada al sistema

La interfaz entre el sistema operativo y los programas de usuario está definida por el conjunto de llamas al sistema (syscalls) ofrecidas por el sistema operativo. Estas realizan las tareas que requieren los programas de usuarios, que deben ser administradas por el sistema como la lectura de un archivo, apertura de un puerto, etc.

En NachOS las syscalls se encuentran completamente implementadas y permite visualizar el ciclo completo de acciones que se gatillas cuando se realiza una llamada a sistema.

2. Syscall en NachOS

2.1. Generalidades

A continuación se verán las modificaciones a Nachos necesarias para la inclusión de dos syscalls:

```
int Cuadrado(int x);

int Duplicar(char* origen, char* destino);
```

La primera retorna el cuadrado del número entregado, mientras que la segunda coloca en destino el string de origen repetido 2 veces, retornando negativo en caso de error.

2.2. Asignación de código

De manera que el sistema sea capaz de identificar cada llamada de forma única, se asigna un número entero a cada una de ellas.

Esta declaración se encuentra en el archivo ***userprog/syscall.h***. Originalmente NachOS contiene un grupo de syscalls ya creadas:

```
#define SC_Halt          0
#define SC_Exit          1
#define SC_Exec          2
(...)
#define SC_LockRelease   20
#define SC_ThreadExit    21
```

Agregaremos a continuación la syscall con el código 22:

```
#define SC_Cuadrado    22
#define SC_Duplicar    23
```

2.3. Declaración de prototipo

De manera de poder realizar la compilación del programa de usuario, se debe tener el prototipo de la función a llamar en espacio de usuario.

Esta declaración del nombre, parámetros y retorno de la función se encuentra en el archivo userprog/syscall.h. En este se encuentran prototipos como:

```
/* Write "size" bytes from "buffer" to the open file.
 * Return the number of bytes actually written on success.
 * On failure, a negative error code is returned.
 */
int Write(char *buffer, int size, OpenFileId id);
```

Incluiremos la declaración de *Cuadrado* y *Duplicar* al final del archivo, antes del final lógico del mismo:

```
/* Retorna el cuadrado del valor entregado */
Int Cuadrado(int x);

/* Retorna el cuadrado del valor entregado */
int Duplicar(char* origen, char* destino);

#endif /* IN_ASM */        ← Lineas existentes de final lógico del archivo
#endif /* SYSCALL_H */     ←
```

2.4. Implementación del trap

En NachOS el trap al kernel (el paso del control desde el programa de usuario al sistema), debe realizarse en lenguaje de máquina para el procesador MIPS simulado.

El archivo a modificar es test/start.s. En este existen llamadas implementadas en assembler de MIPS como:

```
        .globl LockRelease
        .ent    LockRelease
LockRelease:
        addiu $2,$0,SC_LockRelease
        syscall
        j       $31
        .end LockRelease
```

Aquí se esta declarando la implementación en assembler de la función “LockRelease”. Esta copia al registro #2 el codigo de la syscall que se esta llamando; a continuación se realiza trap al sistema, para luego retornar a la instrucción que sigue a continuación en el programa de usuario.

- En los procesadores MIPS, el codigo de las llamadas al sistema siempre se almacena en el registro #2 para su recuperación en espacio de kernel.
- En este ejemplo SC*_LockRealease* es reemplazado por el código declarado en syscall.h al momento de compilar.
- En los procesadores MIPS el registro #31 es el registro de dirección de retorno. La instrucción ***j $31*** es equivalente a escribir “***return***”

Agregamos al final del archivo las implementaciones de las llamadas Cuadrado y Duplicar:

```
        .globl Cuadrado
        .ent    Cuadrado
Cuadrado:
        addiu $2,$0,SC_Cuadrado
        syscall
        j       $31
        .end Cuadrado

        .globl Duplicar
        .ent    Duplicar
Duplicar:
        addiu $2,$0,SC_Duplicar
        syscall
        j       $31
        .end Duplicar



/* dummy function to keep gcc happy */      ← Lineas existentes
.globl  __main                              ← al final del archivo
        .ent    __main                      ←
__main:                                     ←
        j       $31                         ←
        .end    __main                      ←
```

Importante: el espacio en blanco que se muestra antes de las instrucciones es un TAB y NO ESPACIOS EN BLANCO.

2.5. Recepción de la syscall en espacio de Kernel

Nachos provee un punto único de entrada al kernel desde los programas de usuario. Este se encuentra en el archivo ***userprog/exception.cc***, en la función ***ExceptionHandler***. Esta maneja syscalls, fallas de página, fallas de direccionamiento e instrucciones inválidas.

Analizando en detalle la función, se puede ver claramente la recuperación del código de la syscall desde el registro #2 y su almacenamiento en la variable "type":

```
void
ExceptionHandler(ExceptionType which)
{
    int type = kernel->machine->ReadRegister(2);
```

Registros MIPS importantes:

- #2 → Al entrar la llamada contiene el código de la syscal. Al retorna, contiene el valor de retorno de la función.
- #4 → valor del primer parámetro de la función
- #5 → valor del segundo parámetro de la función
- #6 → valor del tercer parámetro de la función

El procesador MIPS simulado solo puede recibir un máximo de 3 parámetros.

A continuación se encuentra la implementación de la tabla de syscalls donde cada llamada tiene un segmento de código para la recuperación de variables e invocación del manejador correspondiente. Por ejemplo la syscall exec tiene el siguiente código:

```
      case SC_Exec:
        vaddr = kernel->machine->ReadRegister(4);
        DEBUG(dbgSysCall,"System Call: Exec vaddr=" << vaddr);
        returnval = ExecHandler(vaddr);
        break;
```

La syscall ***exec*** recibe como parámetro un puntero a un string, y este valor el que es pasado al kernel como parámetro de la función. Dado que es un programa de usuario, este solo conoce direcciones virtuales, y es este virtual address el que se recibe. Una vez recuperada la dirección virtual donde se encuentra el string se llama al manejador. Aquí la función retorna el pid del nuevo proceso creado; este valor es colocado en la variable "*returnval*" para su paso al espacio de usuario vía el registro #2 de la maquina MIPS.

Para la syscall ***Cuadrado*** se debe agregar el siguiente código dentro del switch/case de syscalls:

```
    switch(which) {
    case SyscallException:
      switch(type) {

      case SC_Cuadrado:
        vaddr = kernel->machine->ReadRegister(4);
        DEBUG(dbgSysCall,"System Call: Cuadrado valor=" << vaddr);
        returnval = CuadradoHandler(vaddr);
        break;
```

Aquí hemos reutilizado la variable ***vaddr*** para almacenar el número al cual se le quiere calcular el cuadrado. En el caso de los tipos primitivos (int, char, etc) el valor se encuentra en el mismo registro #2. A continuación se realiza la llamada al manejador de Cuadrado y se coloca el valor entregado en la variable de retorno.

Para la syscall Duplicar, el caso es un poco mas complejo. Esta contiene 2 punteros a string, uno desde el cual leer y otro donde escribir.

```
      case SC_Duplicar:
        int vaddrOrigen = kernel->machine->ReadRegister(4);
        int vaddrDestino = kernel->machine->ReadRegister(5);
        DEBUG(dbgSysCall,"System Call: Duplicar vaddrOrigen =" <<
vaddrOrigen << "  vaddrDestino=" << vaddrDestino );
        returnval = DuplicarHandler(vaddrOrigen, vaddrDestino);
        break;
```

Aquí son recuperadas las direcciones virtuales donde se encuentran los 2 strings desde el registro #4 y #5 respectivamente y se invoca al manejador.

2.6. Implementación de Manejador de llamada

En la recepción de la syscall en el punto 2.5 se realizaron las llamadas a las funciones *CuadradoHandler* y *DuplicarHandler*. Estas son las que realmente implementan las acciones que debe realizar cada syscall.

Para *CuadradoHandler* la implementación es simple:

```
//----------------------------------------------------------------------
// CuadradoHandler
// Retorna el cuadrado del valor entregado
//----------------------------------------------------------------------
 int CuadradoHandler(int numero)
{
        return (numero*numero);
}
```

La implementación de *DuplicarHandler* es más complicada. Aquí se requiere recuperar los string desde espacio de usuario para poder trabajar con ellos en espacio de kernel.

Primero se crearan 2 buffers, uno para recibir el string del usuario y otro para escribir el duplicado. Se recupera el string, se realiza la copia duplicada y luego via la dirección virtual del string de destino, se pasa a espacio de usuario el resultado.

```
int DuplicarHandler(int vOrigen, int vDest)
{
        char* origen = new char[MaxStringArgLength];
        char* destino = new char[MaxStringArgLength];
        int len = 0;

        // se limpia el buffer origen y destino
        bzero(destino, MaxStringArgLength);
        bzero(origen, MaxStringArgLength);

        // recuperacion del string desde espacio de usuario a espacio de kernel
        if ((len = kernel->currentThread->space->UserStringToKernel(vOrigen,origen)) < 0) {

                return -1;
        }

        // copiamos la primera vez el string
        strcpy(destino, origen);
        // segunda vez
        strcpy(destino+len-1, origen);



        // luego debemos pasar el string resultado al espacio de usuario
        if (kernel->currentThread->space->KernelToUserBuf(vDest, MaxStringArgLength,
destino)) {

                return -1;
        }

        return MaxStringArgLength;
}
```

2.7. Inclusión en programa de usuario

La inclusión de las syscalls en un programa de usuario, tiene 2 partes. La primera es la programación misma, y la segunda es la modificación del makefile del directorio test para la inclusión de los headers syscalls.

Un programa de usuario básico que llame a ambas syscall e imprima en pantalla los resultados, es:

```
#include "syscall.h"
#include "io_lib.h"

int main() {
        char destino[256];

        printInt(Cuadrado(10));
        Duplicar("hola", destino);
        print(destino);
       Halt();
}
```

Luego para agregarlo al Makefile de usuarios, se debe editar el archivo test/Makefile, agregando el programa como se muestra en la sección 3.2.3, en el documento 01 de NachOS.

Al compilar y ejecutar el programa se debe apreciar la siguiente salida:

```
user@linux:/usr/local/nachos/code/test$ ./nachos -x ejemplo
100holaholaMachine halting!

Ticks: total 1383, idle 140, system 1070, user 173
Disk I/O: reads 0, writes 0
Console I/O: reads 0, writes 8
Paging: faults 0
Network I/O: packets received 0, sent 0
```