Boletín 7: Llamadas al Sistema en xv6

Ampliación de Sistemas Operativos

Dpto. Ingeniería y Tecnología de Computadores (DITEC)

Universidad de Murcia

Curso 2017/2018

Llamadas al Sistema en xv6

- El propósito de esta segunda práctica es estudiar **como se implementan las llamadas al sistema en xv6**
- Recordemos del tema de gestión de procesos visto en teoría que `xv6` usa la interrupción 64 como interrupción para las llamadas al sistema
- La función `tvinit()` definida en `trap.c` inicializa la IDT (*Interrupt Description Table*)
- Dicha función trata de manera especial a la interrupción de las llamadas al sistema
  - Especifica que es de tipo *trap* pasando un valor 1 como segundo argumento
  - Las *trap* no ponen a cero el flag IF, lo que permite que se produzcan otras interrupciones durante la ejecución del manejador asociado
  - También estable el nivel de privilegio a `DPL_USER` lo que permite que un programa usuario genere el *trap* a través de una instrucción `int`

Llamadas al Sistema en xv6 (cont.)

- **EJERCICIO 1**: ¿Cuál es la rutina de servicio asociada a la interrupción 64?
  (Nota: Utiliza el depurador para ver cómo se inicializa la IDT y que manejador se asocia al *trap* de las llamadas al sistema)

Llamadas al Sistema en xv6

- Veamos ahora cómo se implementan las llamadas al sistema en xv6
- Para ello iniciaremos `qemu` y `gdb`. A continuación cargaremos el fichero de símbolos del programa `ls` y añadiremos un breakpoint en la función `main()`:

```
(gdb) symbol-file _ls
Leyendo símbolos desde _ls...hecho.
(gdb) break main
Punto de interrupción 1 at 0x0: file ls.c, line 75.
```

- Una vez haya arrancado xv6 teclearemos en el *shell* la orden `ls` y depuraremos el código hasta llegar al momento en que se pasa a modo núcleo

Llamadas al Sistema en xv6 (cont.)

- **EJERCICIO 2**: ¿En que punto del núcleo entra el kernel cuando se realiza una llamada al sistema? (una vez se entra en el núcleo se puede volver a cargar los símbolos del núcleo escribiendo "`symbol-file kernel`") ¿Qué llamada al sistema se está ejecutando? ¿Qué se ha guardado en pila kernel del proceso desde que se ejecutó `int 64` hasta que se llama a la función `trap()` implementada en el fichero `trap.c`?
  (Nota: Repasa el código de los ficheros `usys.S`, `vectors.S` y `trapasm.S`)

Llamadas al Sistema en xv6 (cont.)

- **EJERCICIO 3**: Añade una nueva llamada al sistema (`int date(struct rtcdate* d)`) a xv6. El objetivo principal del ejercicio es que veas las diferentes piezas que componen la maquinaria de una llamada al sistema
  - La nueva llamada al sistema obtendrá el tiempo UTC actual y lo devolvera al programa usuario
    - Puedes utilizar la función `cmostime()` (definida en `lapic.c`) para leer el reloj en tiempo real
    - `date.h` contiene la definición de la estructura `rtcdate` que debe pasarse a `cmostime()` como un puntero
  - Debes crear un programa usuario que llame a tu nueva llamada al sistema. Aquí tienes parte del código del programa `date.c`:

Llamadas al Sistema en xv6 (cont.)

```c
#include "types.h"
#include "user.h"
#include "date.h"

int
main(int argc, char *argv[])
{
  struct rtcdate r;

  if (date(&r)) {
    printf(2, "date failed\n");
    exit();
  }

  // Pon aquí tu código para imprimir la fecha en el formato que desees

  exit();
}
```

- Para que tu programa esté disponible en el shell de xv6, añade `_date` a la definición `UPROGS` en el `Makefile`
- La estrategia a seguir para implementar la llamada al sistema es clonar todas las piezas de código de otra llamada al sistema existente (p.e. `uptime()`). Haz un `grep` buscando `uptime` en todos los ficheros fuente (`.c`, `.h` y `.S`) (o utiliza el mecanismo `:tag` de vim)

Llamadas al Sistema en xv6 (cont.)

- En particular, probar los siguientes pasos:
  1. En `syscall.h` hay que darle un nuevo número a la llamada
  2. En `usys.S` hay que añadir la llamada `date`
  3. En `syscall.c` hay que añadir la definición de la función `sys_date()`
  4. En `sysproc.c` es donde se implementan las llamadas al sistema que se realizan desde `syscall()`. Hay que añadir la función `sys_date()` con su implementación
  5. La implementación debe:
     1. Recoger el parámetro `struct rtcdate*` de la primera posición de la pila
     2. Llamar a la función `cmostime()` con ese puntero para obtener la fecha
     3. Por supuesto, hay que comprobar todos los errores y retornar `-1` en caso de error
  6. Añadir la llamada `date()` al fichero de definición de llamadas al sistema para los programas de usuario: `user.h`

Llamadas al Sistema en xv6

- **EJERCICIO 4**: Implementa la llamada al sistema `dup2()` y modifica el *shell* para usarla (usa como ejemplo la implementación de la llamada al sistema `dup()` y consulta cómo debe de comportarse `dup2()` según el estándar POSIX)