Boletín xv6-2: Llamadas al Sistema en xv6

Ampliación de Sistemas Operativos

Dpto. Ingeniería y Tecnología de Computadores (DITEC)

Universidad de Murcia

Curso 2021/2022

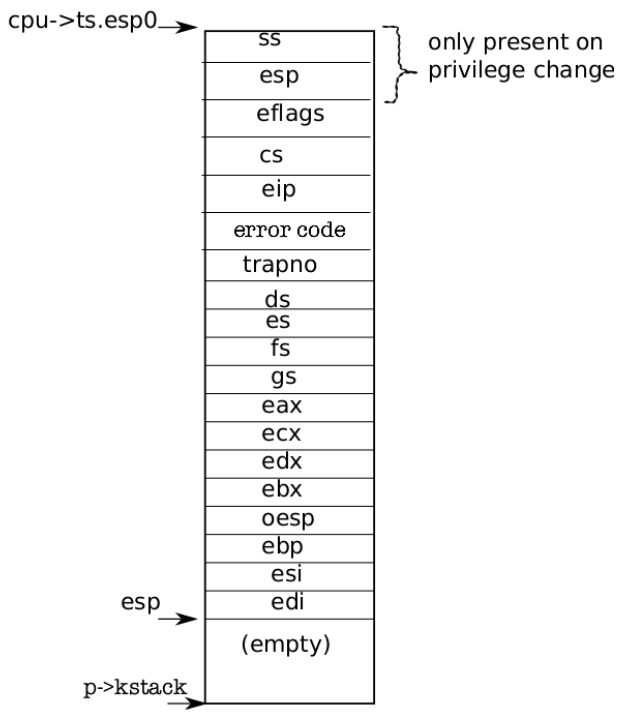
Índice

Las llamadas al sistema en xv6

- La arquitectura x86 permite 256 interrupciones diferentes:
  - Las interrupciones 0–31 se utilizan para las excepciones software (división por cero, acceso a un dirección no válida, etc.)
  - `xv6` asigna las interrupciones hardware al rango 32–63
  - La interrupción 64 (0x40) se usa 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 al *trap* 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 establece el nivel de privilegio a `DPL_USER`, lo que permite que un programa de usuario genere el *trap* con una instrucción `int`
  - `xv6` no permite a los procesos generar otro tipo de interrupciones con `int`. Si lo intentan se genera una excepción de protección general que es tratada a través del vector 13

Las llamadas al sistema en xv6 (cont.)

- Al cambiar de modo usuario a modo núcleo se realiza el cambio a la pila del núcleo
  - Cargando los registros `%esp` y `%ss` desde el segmento de estado de la tarea y almacenado los antiguos valores en la pila
- Se almacenan los registros `%eflags`, `%cs` y `%eip`. En algunos casos se almacena también una palabra de error
- Se actualizan los valores de `%eip` y `%cs`
- Se salta a `alltraps` que almacena varios registros de segmento y los de propósito general



Las llamadas al sistema en xv6 (cont.)

- A continuación **`alltraps`** establece los valores correctos de los distintos segmentos para poder ejecutar el manejador correspondiente en modo *kernel*

```
# Set up data and per-cpu segments
movw $(SEG_KDATA<<3), %ax
movw %ax, %ds
movw %ax, %es
movw $(SEG_KCPU<<3), %ax
movw %ax, %fs
movw %ax, %gs

# Call trap(tf), where tf=%esp
pushl %esp
call trap
```

Las llamadas al sistema en xv6 (cont.)

- Después de regresar elimina el argumento de la pila sumando 4 al puntero de pila y comienza a ejecutar el codigo a partir de la etiqueta **trapret** para regresar a modo usuario

```
  addl $4, %esp

  # Return falls through to trapret...
.globl trapret
trapret:
  popal
  popl %gs
  popl %fs
  popl %es
  popl %ds
  addl $0x8, %esp  # trapno and errcode
  iret
```

Implementación de llamadas al Sistema en xv6

- El propósito de esta segunda práctica es estudiar **cómo se implementan las llamadas al sistema en xv6**
- Recordemos del apartado configuración de las interrupciones visto en la práctica anterior que xv6 usa la interrupción 64 (0x40) como interrupción para las llamadas al sistema

Ejercicios de Implementación de llamadas al Sistema en xv6

Hecha - Falta comentarla en la memoria

- **EJERCICIO 1**: Añade una nueva llamada al sistema a xv6:

```
int date(struct rtcdate *d)
```

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 de usuario que llame a tu nueva llamada al sistema. Aquí tienes parte del código del programa `date.c`:

Ejercicios de Implementación de 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`

Ejercicios de Implementación de llamadas al Sistema en xv6 (cont.)

- 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)

Ejercicios de Implementación de 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. Añadir la llamada `date()` al fichero de definición de llamadas al sistema para los programas de usuario: `user.h`
  4. (con esto ya estaría todo listo para que los programas del S.O. puedan llamar a la nueva llamada)
  5. En `syscall.c` hay que añadir la definición de la función `sys_date()`
  6. 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
  7. 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

Ejercicio 2 – dup2()

Hay que hacerla - Tenemos el esqueleto (ver si hay algun video)

- **EJERCICIO 2**: 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)

Ejercicio 3 – `exit()` y `wait()`

hay que hacerlas de 0

- **EJERCICIO 3**: Modifica las llamadas `exit()` y `wait()` para que sigan la signatura de las funciones de POSIX, es decir:

```
int wait(int* status)
void exit(int status)
```

Para que el sistema siga funcionando, hay que cambiar las llamadas a `exit()` and `wait()` que hay a lo largo del sistema operativo para que ahora acepten un parámetro:

1. Hay que cambiar todas las llamadas `exit()` por `exit(0)`. Para eso hay que ejecutar en el código de xv6 la siguiente línea en el *shell* (se cambia en todos los ficheros C salvo en `sysproc.c` y `trap.c`):

```
$ sed -i -e 's/\bexit()/exit(0)/g' \
                    @(!(sysproc.c|trap.c)|(*.c))
```

Ejercicio 3 – `exit()` y `wait()` (cont.)

2. Hay que cambiar todas las llamadas `wait()` por `wait(0)` (equivale a `wait(NULL)`, salvo que la constante NULL no está definida). Para eso hay que ejecutar en el código de xv6 la siguiente línea en el *shell*:

```
$ sed -i -e 's/\bwait()/wait(0)/g' \
                    @(!(sysproc.c|trap.c)|(*.c))
```

3. Las funciones de llamada al sistema (en `user.h`) se tienen que adaptar para aceptar los argumentos
4. También hay que adaptar las correspondientes funciones de implementación de las llamadas dentro del núcleo, `sys_exit()` y `sys_wait()` para aceptar, respectivamente, un entero (el estado de salida del proceso) y un puntero a enteros (obsérvese el uso de `argptr()` dentro del núcleo)
5. La implementación en `sys_wait()` y `sys_exit()` debe ser tal que cuando un proceso llame a `wait(estado)` y reciba el estado de su hijo, éste sea exactamente el estado que el programa que terminó especificó en su llamada a `exit(estado)` (**Pista**: obsérvese las distintas llamadas a las funciones `exit()` y `wait()` dentro del núcleo en los ficheros `sysproc.c` y `trap.c`)

Ejercicio 3 – `exit()` y `wait()` (cont.)

6. Modificad la llamada al sistema `wait()` del shell (`sh.c`) en su función `main()` para que cada vez que ejecute un programa produzca una salida:

```
Output code: N
```

donde `N` es el código de salida del programa que se ejecutó

7. Implementa el código de salida de manera que funcionen los macros definidos en POSIX, con una pequeña diferencia, que como el error (trap) puede ser 0, al retornar el número de trap, se tiene que sumar uno a su valor (insértalos en `user.h`):

```
#define WIFEXITED(status)   (((status) & 0x7f) == 0)
#define WEXITSTATUS(status) (((status) & 0xff00) >> 8)
#define WIFSIGNALED(status) (((status) & 0x7f) != 0)
```

Y el siguiente macro específico de xv6:

```
#define WEXITTRAP(status)   (((status) & 0x7f) - 1)
```

Hay que probar esto