Boletín 8: Reserva de páginas bajo demanda en xv6

Ampliación de Sistemas Operativos

Dpto. Ingeniería y Tecnología de Computadores (DITEC)

Universidad de Murcia

Curso 2017/2018

Reserva de Páginas Bajo Demanda en xv6

- Una de las muchas optimizaciones que los SS.OO. implementan es la reserva bajo demanda de páginas en la memoria *heap* (*lazy allocation*)
- En `xv6`, las aplicaciones solicitan memoria al kernel a través de la llamada al sistema `sbrk()`
- En el kernel que estamos utilizando `sbrk()` reserva las páginas físicas necesarias y las mapea en el espacio de direcciones virtuales del proceso
- Existen programas que reservan memoria pero nunca la utilizan (p.e. para implementar arrays dispersos de gran tamaño)
- Por esta razón los kernels suelen retrasar la reserva de cada página de la memoria hasta que la aplicación intenta usarla

Reserva de Páginas Bajo Demanda en xv6 (cont.)

- **EJERCICIO 1**: Implementa esta carácteristica de reserva diferida en xv6. Para ello:
  1. Elimina la reserva de páginas de la llamada al sistema `sbrk()` (implementada a través de la función `sys_sbrk()` en `sysproc.c`)
     - La nueva función debe incremantar el tamaño del proceso (`proc->sz`) y devolver el tamaño antiguo pero no debe reservar memoria
     - No se debe llamar a `growproc()` en caso de que el proceso crezca
  2. Realiza las modificaciones, arranca xv6, y teclea `echo hola` en el *shell*:

```
init: starting sh
$ echo hola
pid 3 sh: trap 14 err 6 on cpu 0 eip 0x12f1 addr 0x4004--kill proc
$
```

Reserva de Páginas Bajo Demanda en xv6 (cont.)

3. El mensaje `pid 3 sh: trap...` proviene del manejador de *traps*, definido en `trap.c`
   - Ha capturado un fallo de página (trap 14 o `T_PGFLT`) que el kernel de xv6 no sabe cómo manejar
   - `addr 0x4004` indica la dirección virtual que generó el fallo de página
4. Modifica el código en `trap.c` para responder a un fallo de página en el espacio de usuario mapeando una nueva página física en la dirección que generó el fallo, regresando después al espacio de usuario para que el proceso continúe
   - Debes añadir tu código justo antes de la llamada a `cprintf()` que produce el mensaje
   - No es necesario que el código cubra todas las posibles situaciones (es suficiente con que permita al *shell* ejecutar comandos simples como `echo` y `ls`)
   - Reutiliza código de `allocuvm()` en `vm.c` y usa `PGROUNDDOWN(va)` para redondear la dirección virtual a límite de página
   - Necesitarás llamar a `mappages()`. Para ello borra la declaración de función estática en `vm.c` y declara `mappages()` en `trap.c`

Reserva de Páginas Bajo Demanda en xv6 (cont.)

- La implementación realizada en el ejercicio 1 no es totalmente correcta ya que no contempla varias situaciones:
  - El caso de un argumento negativo al llamarse a `sbrk()`
  - Manejar el caso de fallos en la página inválida debajo de la pila
  - Verificar que `fork()` y `exit()` funciona en el caso de que haya direcciones virtuales sin memoria reservada para ellas
  - Asegurarse de que funciona el uso por parte del kernel de páginas de usuario que todavía no han sido reservadas (p.e., si un programa pasa una dirección de la zona de usuario todavía no reservada a `read()`)
- **EJERCICIO 2**: Modifica el código del primer ejercicio para que contemple las dos primeras situaciones anteriormente descritas
- **OPCIONAL**: Modifica el código para que funcione correctamente en todas las situaciones anteriormente descritas (Nota: para obtener la puntuación máxima en este boletín no es necesario realizar esta parte opcional)