SOA (18/01/2019)

Final de laboratorio

Crea un fichero que se llame “respuestas.txt” donde escribirás las respuestas para los apartados de los ejercicios del control. Indica para cada respuesta, el número de ejercicio y el número de apartado (por ejemplo, 1.a, 1.b, ...).

Importante: para el primer ejercicio tienes que partir de la versión de Zeos original que te hemos suministrado.

*1. (6 puntos) **Spaceship...***

El señor Baka Baka quiere implementar un videojuego sobre nuestro sistema Zeos. Le gustaría diseñar un juego en el que se visualizara una nave espacial en la pantalla y poder controlarla a izquierda y derecha con el teclado, pero se da cuenta que actualmente no tiene acceso a los dispositivos necesarios (teclado y pantalla).
Para darle soporte tienes que implementar la llamada a sistema:

int get_key(char c);*

que permite al proceso de usuario obtener una tecla. Si no hay teclas disponibles el proceso debe bloquearse hasta que el usuario pulse una nueva tecla, momento en el que el proceso se desbloquea inmediatamente devolviendo en ´c´ la tecla recibida. Si diferentes procesos ejecutan esta llamada, las teclas deben servirse en un estricto orden secuencial (FIFO). Las teclas que el usuario vaya pulsando deben guardarse internamente en un buffer circular.

Sabiendo que la pantalla es una matriz de 25 filas por 80 columnas (*char pantalla[25][80]*), también tienes que implementar la llamada a sistema:

*int put_screen(char *s)*

que permite volcar a la pantalla el contenido de una matriz de 25 por 80 caracteres apuntada por ´s´.

Para implementar estas nuevas funcionalidades del sistema debes usar el mecanismo habitual con **sysenter** para *get_key* (número de servicio 7) y la **interrupción 0x60** para llamar en exclusiva a *put_screen*.

En el código suministrado hay un ejemplo de uso de estas rutinas.

La solución propuesta tiene que ser eficiente y escalable.

a) Indica qué estructuras de Zeos tienes que modificar para implementar estas nuevas funcionalidades.
b) Indica qué nuevas estructuras de Zeos tienes que añadir.
c) Escribe el código para habilitar get_key.
d) Escribe el código para habilitar put_screen.

e) Implementa el wrapper de la llamada al sistema *get_key*.
f) Implementa el wrapper de la llamada al sistema *put_screen*.
g) Implementa el handler para la llamada al sistema *get_key*.
h) Implementa el handler para la llamada al sistema *put_screen.*
i) Explica qué errores pueden devolver estas llamadas.
j) Implementa la llamada a sistema *sys_get_key*.
k) Implementa la llamada a sistema sys_put_screen.
l) Implementa la gestión del teclado.
m) ¿Podría morir con un *exit* un proceso que se ha bloqueado en la llamada a sistema *get_key*?

2. (4 puntos) Sockets

Queremos implementar un servidor centralizado de logs. Este servidor será multiprogramado. Su sintaxis es:

```
>log_server port
```

Donde *port* es el número de puerto por el cual recibirá mensajes.

Cuando un proceso cliente se conecte al servidor, le envía mensajes que el servidor de logs tiene que almacenar en un fichero. Cada uno de estos mensajes está compuesto por el PID del proceso que le envía el mensaje, la longitud en caracteres del mensaje, más una tira de caracteres con el texto del mensaje. Cuando el log_server recibe un mensaje, lo escribe en el fichero de logs. En este fichero, se tiene que escribir el PID del proceso que le ha enviado el mensaje junto con el texto del mensaje. Se tiene que asegurar que los mensajes que escribe el log_server no aparezcan mezclados. Dado que el servidor puede recibir mensajes de más de un cliente de forma simultánea, cada vez que un cliente se conecte al servidor, se creará un nuevo thread para procesar los mensajes que envíe ese cliente. Un cliente puede enviar más de un mensaje.

Dentro del código del cliente, tienes que implementar la función:

```
int send_msg(int socketfd, char *msg);
```

que sirve para enviar el mensaje cuyo texto está en *msg*, a través del socket *socketfd*.

a) Escribe la secuencia de llamadas al sistema para crear el puerto en el servidor de logs
b) Escribe la secuencia de llamadas al sistema para conectarse al puerto del servidor de logs desde un cliente
c) Escribe el código de la función *send_msg*
d) Escribe el código para recibir un mensaje en el servidor
e) (2 puntos) Implementa log_server y el cliente (llámalo log_client)

3. (1 *punto) Generic Competences Third Language (Development Level: mid)*

The following paragraph belongs to the book *Understanding the Linux Kernel* by D. Bovet and M. Cesati:

"The translation of linear addresses is accomplished in two steps, each based on a type of translation table. The first translation table is called the Page Directory, and the second is called the Page Table.
The aim of this two-level scheme is to reduce the amount of RAM required per-process Page Tables. If a simple one-level Page Table was used, then it would require up to 2^20 entries (i.e. at 4 bytes per entry, 4MB of RAM) to represent the Page Table for each process (if the process used a full 4GB linear address space), even though a process does not use all addresses in that range. The two-level scheme reduces the memory by requiring Page Tables only for those virtual memory regions actually used by a process"

Create a text file named "generic.txt" and answer the following questions (since this competence is about text understanding, you can answer in whatever language you like):

a) What is the name of the second translation table used for translating linear addresses?
b) Is it usual that a process uses 4GB of linear address space?
c) Why a two-level linear address translation reduces the amount of memory required to store all the translations?

Entrega

Sube al Racó los ficheros "respuestas.txt" y "generic.txt", junto con el código que hayas creado en cada ejercicio.

Para entregar el código utiliza:

```
> tar zcfv examen.tar.gz ejercicio1 ejercicio2
```