SOA (18/06/2020)

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 numero de apartado (por ejemplo, 1.a, 1.b, ...).

Importante: para cada uno de los ejercicios tienes que partir de la versión de Zeos original que te hemos suministrado.

*1. (5 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* 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. (5 puntos) Memoria Dinámica

Para poder tener diferentes pantallas de juego, decidimos añadir un par de llamadas a sistema nuevas:

*char * getScreen(void);*
int freeScreen(char buff);*

getScreen reserva una región de 2000 bytes (80*25) en memoria de usuario (coge un frame físico libre y lo mapea en algun lugar vacio del espacio lógico del proceso) devolviendo la 1ª dirección disponible.
freeScreen libera una de estas regiones. El parámetro *buff* de *freeScreen* corresponde a un puntero devuelto por la función *getScreen* con anterioridad.
Se valorará la eficiencia y la eficacia de la solución, y que la solución propuesta sea escalable a sistemas con muchos más procesos.
Contesta a las siguientes preguntas:

a) ¿Qué estructuras nuevas se tienen que añadir al sistema? Indica también su utilidad.
b) ¿Qué estructuras actuales del sistema operativo se tienen que modificar?
c) Escribe el pseudocódigo de la función *busca_espacio_libre_user*() que retorna la dirección inicial de una zona libre del espacio de direcciones de usuario donde guardar un bloque de memoria.
d) Implementa la función *sys_getScreen*.
e) Implementa la función *sys_freeScreen*.
f) Implementa los cambios necesarios en el *sys_fork.*
g) Comenta si habría que modificar alguna/s rutina/s de sistema adicional/es y el motivo sin entrar en demasiados detalles.

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 zeos
```