SOA (20/06/2023)

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. (4 puntos) Show me your pars

Queremos implementar la llamada a sistema:

```
int show_me_your_pars(int par1, int par2, int par3)
```

esta llamada imprime por pantalla el valor de los 3 parámetros y retorna 0. Si el primer parámetro (*par1*) es negativo debe retornar un -1 y modificar el valor de la variable de usuario *errno* a 666. Esta llamada a sistema sólo debe poder invocarse mediante la interrupción 110.

El paso de parámetros y la devolución de resultados de esta llamada se realizará mediante una nueva zona de memoria de usuario [0x3FF000-0x400000) dedicada única y exclusivamente a ello. Esta zona de memoria debe estar presente en todos los procesos del sistema.

Los parámetros deben copiarse a esta zona de la siguiente forma:

0x3FF000	par1
0x3FF004	par2
0x3FF008	par3

El handler accederá a esta zona de memoria para coger los parámetros de la llamada y guardará el resultado en la dirección 0x3FF000.

El wrapper en C de esta llamada es el siguiente:

```
int show_me_your_pars(int par1, int par2, int par3){
      copy_pars(par1,par2,par3);
      call_syscall();
      return value_or_error();
}
```

a) (1 punto) Implementa la función

```
      int map_parameters(page_table_entry* PT)
```

para buscar un frame físico y mapearlo a la zona de paso de parámetros en una tabla de páginas determinada.

b) (0,5 puntos) Indica dónde añadir la función anterior para inicializar los espacios de direcciones de todos los procesos (y cómo invocarla).

c) (0,5 puntos) Indica la línea de código necesaria para activar esta llamada a sistema.

d) (1 punto) Indica el código de la función *call_syscall*.

e) (0,5 puntos) Indica el código de la función *copy_pars*.

f) (0,5 puntos) Indica el código de la función *value_or_error*.

2. (6 puntos) Look over your shoulder

Queremos poder mapear una zona de memoria de un proceso en el espacio de direcciones del proceso actual. En particular queremos la llamada a sistema

```
void* share(int pid, void* address, int size)
```

Esta llamada al sistema mapea las páginas que contienen el rango de direcciones [*address*, *address+size*) en un rango consecutivo del espacio de direcciones del proceso actual no usado y devuelve la dirección que correspondería a *address* en el nuevo espacio.
Si no hay espacio para realizar el mapeo, debe devolver NULL. El número de regiones mapeadas por un proceso debe limitarse a 10.

Igualmente debemos añadir una función para desmapear una de esas zonas:

```
int unshare(void* address)
```

Donde address es una de las direcciones devueltas por *share*.
Si un proceso muere pero tienes regiones mapeadas por otro proceso, no puede liberar las páginas físicas asociadas.

a) (1 punto) Indica qué estructuras de Zeos tienes que modificar para implementar esta nueva funcionalidad, así como las funciones que deberían cambiar. Justifica la necesidad de cada campo.
b) (0,5 puntos) Indica qué nuevas estructuras de Zeos tienes que añadir.
c) (1 punto) Implementa la rutina

```
int find_pages(int n)
```

para encontrar la página del espacio de direcciones de usuario en la que encontrar *n* páginas consecutivas.
d) (1 punto) Implementa la rutina

```
int map_pages(struct task_struct* SRC, int start, int end,
              int from)
```

que mapea las páginas *[start, end)* del proceso *SRC* en el espacio lógico del proceso actual a partir de la página *from*.
e) (0,5 puntos) Es necesario en la rutina anterior llamar a la función set_cr3? ¿Por qué?
f) (0,5 puntos) Implementa la llamada a sistema sys_*share*
g) (0,5 puntos) Indica los cambios necesarios en la rutina *sys_exit*.
h) (1 puntos) Implementa todo el mecanismo descrito.

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 respuestas.txt
generic.txt
```