SOA (16/01/2024)

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. (2 puntos) Syscalls

Implementa la siguiente llamada al sistema:

```
int schedule(int pid);
```

esta llamada fuerza un cambio de contexto al proceso cuyo pid se pasa como primer parámetro. En caso de que no haya ningún proceso con el pid indicado, devolverá error. En cualquier otro caso, devolverá 0. La llamada al sistema tendrá como número de servicio el 11.

2. (2 puntos) Copy memory from process

Implementa la rutina de sistema

```
int copy_memory_from(struct task_struct* SRC, char* from, int size,
                     char* to)
```

que copia la región de memoria *[from, from+size)* del proceso *SRC* en el espacio lógico del proceso actual a partir de la dirección *to*. Las direcciones from y to deben estar alineadas a página. Esta función tiene que buscar 2 páginas consecutivas libres en el espacio lógico del proceso actual para realizar la copia.

3. (6 puntos + 0,5 puntos) ZZzzzz...

Queremos implementar una nueva llamada a sistema:

```
int sleep(int seconds);
```

que permita *dormir* un proceso durante *seconds* segundos. Esta llamada es bloqueante y, por lo tanto, durante este tiempo el proceso no se ejecuta. Al finalizar este tiempo, el proceso se despierta y puede volver a ejecutarse. Esta función retorna 0 si se despierta normalmente o un número negativo en caso de error (EINVAL, si el parámetro es <0 o EINTR, si se despierta antes de tiempo).

También queremos implementar otra llamada a sistema:

```
int wakeup(int pid, int NOW);
```

que permita *despertar* a un proceso dormido. Esta función tiene un parámetro (*NOW*) que, con un valor 1, indica que se tiene que pasar a ejecutar el proceso dormido inmediatamente. Si NOW es diferente de 1, el proceso debe respetar el orden de otros procesos que estén pendientes de ejecución.

Esta función retorna 0 si ha podido despertar al proceso correctamente o un número negativo en caso de error (EINVAL, algún parámetro no es válido o EEXIST, si el proceso no estaba dormido).

Para implementar estas nuevas funcionalidades del sistema **no se debe usar el mecanismo habitual de llamadas a sistema**. Debes usar las entradas 0x60 y 0x61 de la IDT para llamar de forma directa a las funciones de sistema *sleep* y *wakeup* respectivamente.

Para controlar el tiempo, puedes tener en cuenta que el reloj está programado a 18Hz (o sea, que genera 18 interrupciones por segundo).
Aunque actualmente haya una limitación de 10 procesos, la solución propuesta tiene que poder tratar de forma eficiente con miles de procesos.

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 estas nuevas funcionalidades.
d) Implementa el wrapper de la llamada al sistema *sleep*.
e) Implementa el wrapper de la llamada al sistema *wakeup*.
f) Implementa el handler para la llamada al sistema *wakeup*.
g) ¿Dónde puedes detectar que hay que despertar a un proceso? Escribe el pseudocódigo para detectar procesos dormidos que hay que despertar.
h) Indica el código de la llamada sistema *wakeup* para el caso de NOW=1. Puedes suponer que tienes implementada la siguiente función: *struct task_struct * find_task_by_pid(int pid)* que dado un *pid* te devuelve un puntero a su PCB (o NULL) si no existe.
i) Explica como detecta la llamada a sistema *sys_sleep* que debe devolver el valor –EINTR.
j) ***(0.5 puntos)*** Implementa correctamente en Zeos los cambios propuestos en los apartados anteriores.

4. (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 ejercicio3 respuestas.txt generic.txt
```