SOA (10/04/2025)

Primer control 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 cada uno de los ejercicios tienes que partir del código suministrado de Zeos.

1. *(3 puntos)* Comprensión de Zeos

a) (1 punto) Dada la rutina *funcionExamen* (no incluida en el código), indica los comandos necesarios (y como usarlos) para saber si se trata de una rutina de sistema o de usuario.
b) (1 punto) ¿En qué **página** se encuentra el PCB del proceso inicial? ¿Cómo lo has hallado?.
c) (1 punto) Dado el fichero *exam.*S (no incluida en el código) programado en ensamblador y que usa directivas del preprocesador de C. Indica **la línea (o líneas) de comandos** necesarias para compilar y linkar este objeto con el binario de sistema.

2. *(4 puntos) Where are my args?*

Nos hemos dado cuenta que, según la convención de llamadas a función en C, no es necesario mantener el valor de los registros ECX y EDX y esto simplificaría enormemente el código necesario para realizar las llamadas a sistema rápidas. La idea es que el *wrapper* guarde la dirección del código y pila de usuario a usar para volver de sistema en estos registros tal y como los necesitará el *sysexit*.
En particular, en el *wrapper*:

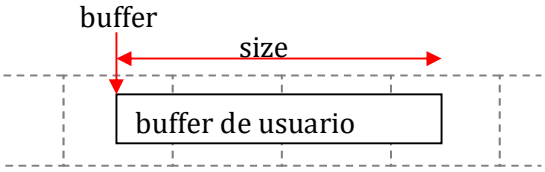
a) Modifica el código para guardar en estos dos registros, respectivamente, la dirección de retorno a código de usuario y la dirección de la pila de usuario, y elimina el código innecesario del falso enlace dinámico actual.
b) Esta modificación impide hacer el paso de parámetros como hasta ahora, por lo que en este caso dejaremos los parámetros en la pila de usuario y accederemos a ellos desde sistema mediante el registro EBP. Modifica el código del paso de parámetros para el wrapper de *write*.

En el *handler* hay que modificar varias cosas:

c) Cambia el código inicial para mantener el contenido de la pila como lo haría el procesador ante una interrupción hardware.
d) Dado que ahora los parámetros se encuentran en la pila de usuario, hay que apilarlos explícitamente en la pila de sistema justo antes de hacer la llamada a la rutina de servicio que corresponda. Como el número de parámetros a apilar depende de cada llamada, usaremos una rutina auxiliar en ensamblador que nos ayude, que se llamará *sys_XXX_wrap*, donde XXX corresponde a la llamada en cuestión. Por ejemplo, la rutina *sys_write_wrap* debe, usando el registro EBP, apilar los parámetros necesarios (3), hacer la llamada a sistema real (a *sys_write*) y retornar. Implementa el código de la rutina *sys_write_wrap*.
e) Usa estas rutinas auxiliares donde corresponda en la tabla *sys_call_table*.
f) Finalmente, gracias a estos cambios, simplifica el código para volver a modo usuario.

3. (3 puntos) Zero copy write

Queremos optimizar el código de nuestra rutina *sys_write* en ZeOS y evitar la copia del buffer de usuario a sistema. Para ello tienes que usar, como buffer de sistema, un rango de direcciones lógicas fijo (0x3FF000-0x3FFFFF) que mapeará el espacio físico del buffer de usuario. Como el buffer de usuario puede ser muy grande, habrá que ir mapeando y escribiendo bloques como el código original. Recuerda que la dirección del buffer puede ser cualquiera, y en particular no tiene por que estar alineada a página:



Reprograma la rutina *sys_write* con esta optimización en 2 pasos:

a) (2 puntos) Suponiendo que la dirección SIEMPRE está alineada a página.
b) (1 punto) Permitiendo cualquier dirección de usuario.

4. (1 punto) *Generic Competences Third Language (Development Level: mid)*

The following paragraph belongs to Wikipedia:

"A paged MMU also mitigates the problem of external fragmentation of memory. After blocks of memory have been allocated and freed, the free memory may become fragmented (discontinuous) so that the largest contiguous block of free memory may be much smaller than the total amount. With virtual memory, a contiguous range of virtual addresses can be mapped to several non-contiguous blocks of physical memory; this non-contiguous allocation is one of the benefits of paging.

However, paged mapping causes another problem, internal fragmentation. This occurs when a program requests a block of memory that does not cleanly map into a page, for instance, if a program requests a 1 KB buffer to perform file work. In this case, the request results in an entire page being set aside even though only 1 KB of the page will ever be used; if pages are larger than 1 KB, the remainder of the page is wasted. If many small allocations of this sort are made, memory can be used up even though much of it remains empty."

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

1.- Does ZeOS suffer from external fragmentation?
2.- Does ZeOS suffer from internal fragmentation?
3.- What is the ideal page size for the code section of a process in ZeOS to prevent internal fragmentation if present?

5. 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
```