SOA2 (16/01/2024)

Nombre:

DNI:

Examen final de teoría

Justifica todas tus respuestas. Una respuesta sin justificación se considerará errónea.

1. (2.5 puntos) Básicas

Responde brevemente a las siguientes preguntas

a) ¿Qué es un master boot record?

Es el primer sector de todos los discos duros booteables, son 512B que contienen el codigo para bootear el sistema.

b) ¿Qué es un bootloader?

Hoy en dia los sistemas operativos son tan extensos que no se pueden cargar con 512B de código, el MBR apunta al bootloader, que es mas extenso y permite cargar el sistema

c) ¿Qué significa pasar a modo protegido?

**Pasar a modo sistema, donde hay privilegios que los usuarios no deben tener

d) ¿Qué son y para qué sirven los niveles de privilegios del procesador?

Si dejamos que los usuarios puedan modificar y ejecutar cualquier cosa, pueden modificar sectores de la maquina que la rompan, el modo protejido evita que sin querer o queriendo se modifique nada que no se deba

e) ¿Por qué cada proceso tiene dos pilas, una de usuario y otra de sistema?

Pila de Usuario: Es utilizada mientras el proceso ejecuta código en modo usuario, permitiendo aislar las operaciones del usuario sin comprometer el sistema.
Pila de Sistema: Se utiliza al cambiar a modo kernel (por ejemplo, en llamadas al sistema o interrupciones) para proteger y controlar el acceso directo al hardware y recursos críticos del sistema.

f) ¿Cuál es el contenido de la pila de sistema antes de ejecutar la primera instrucción de sys_fork?

**De abajo a arriba: cntx hw, cntx sw, retorno al handler de syscall, ebp

g) ¿Para qué sirven los wrappers de las llamadas al sistema?

Cada sistema operativo puede que implemente ciertas llamadas a sistema de manera distintaa internamente, los wrappers unifican las cabeceras y parametros para el codigo de los usuarios, para que sea mas portable

h) ¿Qué 2 formas hay de recoger el resultado de una llamada al sistema?

A través del registro de retorno %EAX, o a través de la variable errno en caso de que falle

i) Refiriéndonos al sistema de memoria, ¿Qué es un directorio?

Un directorio es una estructura de datos que contiene las direcciones de las tablas de página usadas por el proceso.
Este directorio permite que el sistema traduzca direcciones logicas a direcciones físicas para acceder a la memoria del proceso.

j) ¿Para qué sirve el registro cr3?

Apunta a la base del directorio de paginas, con la tabla de páginas del proceso actual, pudiendo calcular la id de la página física x:
x = %cr3 + id_pl * size entrada tabla páginas

k) En un procesador de 32 bits que no utiliza directorios para las tablas de páginas, ¿cuantos bits se utilizan de una dirección lógica para el identificador de página?

32 - log_2(medida pagina)
32 - log_2(4096) = 20 (bits 31..12)

l) ¿Qué mecanismo se utiliza para sincronizar el contenido del TLB con la tabla de páginas del proceso?

En caso de cambio de contexto a otro proceso, al cambiar el registro %cr3 se hace un flush de la TBL automáticamente. En el mismo proceso forzamos ese flush cuando cambiamos entradas en la tabla de paginas.

m) ¿Por qué el task_struct de un proceso está solapado con su pila de sistema?

??** Eficiencia memoria?

Nombre:

DNI:

n) Escribe el código de la función task_switch vista en teoría

```
push %ebp
%ebp <- %esp
%cr3 <- new_page_Table_baseAddr
tss.esp0 = &new->stack[1024]

current.task->kernel_esp = ebp
%esp <- new.task->kernel_esp

pop %ebp
ret
```

o) ¿Qué cambios se tienen que hacer en el contexto de ejecución del proceso hijo, en la llamada al sistema sys_fork, para que cuando pase por primera vez a RUN ejecute ret_from_fork?

Empilar el valor 0 (en la cima de la pila)
-Machacar el valor de ebp por @ret_from_fork, asi creamos un dinamic link falso que nos retornara a la funcion ret_from_fork

p) ¿Qué recursos del proceso comparten los threads?

q) Escribe el código de un gestor de E/S

r) ¿Qué es un device descriptor?

s) ¿Qué es un driver de dispositivo?

t) ¿Cuál es la función de un virtual file system?

2. (2.5 puntos) Moderadas

a) Calcula el espacio ocupado por las tablas de páginas de un proceso que ocupa 400 MBs de memoria en una arquitectura con una tabla de páginas de 2 niveles donde cada entrada del directorio y de la tabla de páginas ocupa 32 bits. Puedes suponer que los 10 bits de más peso de una dirección lógica indexan dentro del directorio, los siguientes 10 bits indexan dentro de la tabla de páginas y los 12 bits de menos peso corresponden al offset dentro de la página.

Espacio proceso en KB -> 400 MB = 400 * 1024KB = 409.600
Nº páginas = 409.600KB/ 4KB = 102.400 páginas
Nº de tablas de páginas = paginas/ tamaño pagina = 102.400 / 2^{10} = 100 tablas de paginas
Espacio 1 entrada 32 bits = 4B
Espacio total = entradas directorio * bytes/entrada +
tablas de paginas * entradas/tabla * bytes/entrada = 1024*4B + 100*1024*4B =
4KB + 400KB = 404KB

b) Calcula el porcentaje de disco que ocupa una FAT si el disco es de 4 GBs, los bloques de disco son de 2 KBs y los índices a bloques de datos son de 4 bytes.

c) ¿Por qué se tiene que crear el proceso Init en tiempo de boot y no cuando se ha acabado éste?

Para que cuando salte a modo usuario tenga un proceso al que saltar

d) Dibuja el contenido de la pila de sistema antes de ejecutar la primera instrucción de la clock_routine si el proceso actual es idle.

0
@cpu_idle

e) Escribe el código de la función task_switch optimizada para reducir el número de fallos de TLB durante el cambio de contexto en un sistema operativo multiproceso multihilo

Nombre:
DNI:

3. (2.5 puntos) Complejo

Queremos especializar una llamada al sistema cuyo comportamiento dependa de una característica propia del proceso. Por esta razón, diferenciaremos entre procesos "especiales" y procesos "no especiales"

Para hacer una primera implementación hemos introducido los siguientes cambios (para simplificar, si quieres, puedes considerar que estos cambios se han hecho en ZeOS):

-se ha añadido una variable, de tipo int, llamada *special* en el task_struct de los procesos.

-se ha creado un handler para esta llamada al sistema el cual se ejecuta a través de la posición 0x3e de la IDT. El código del handler es el siguiente:

```
ENTRY(special_handler)
        SAVE_ALL
        call sys_special
        EOI
        RESTORE_ALL
        iret
```

-también se ha añadido el código de la llamada al sistema:

```
int sys_special()
{
        if (current()->special == 0)
                printk("I am not special");
        else
                printk("I am special");
        return 0;
}
```

-para poder probar el nuevo mecanismo, se ha añadido una segunda llamada al sistema, que va por el mecanismo habitual de las llamadas al sistema, para poner el valor de la variable *special* del task_struct del proceso actual:

```
int sys_SetSpecial(int value)
{
        current()->special = value;
        return 0;
}
```

a) Nos hemos dado cuenta en esta implementación que los procesos hijos siempre heredan el valor de la variable special del task_struct de su proceso padre. ¿a qué se debe esto?

> Al hacer fork(), se copia la task_struct del padre, y si no se modifica los procesos hijos tendran el mismo valor special

Aunque esta implementación parece, a priori, sencilla, resulta que no lo es ya que en la función sys_special tiene muchas secciones de código especializadas con "if (current()->special == 0)" lo que hace que sea difícil de desarrollar y mantener.

Para solucionar esto, se proponen 2 implementaciones diferentes:

Implementación 1: IDT

Ahora disponemos de 2 handlers y 2 rutinas de servicio:

```
ENTRY(no_special_handler)
        SAVE_ALL
        call sys_no_special
        EOI
        RESTORE_ALL
        iret
int sys_no_special()
{
        printk("I'm not special");
}
```

```
ENTRY(special_handler)
        SAVE_ALL
        call sys_special
        EOI
        RESTORE_ALL
        Iret
int sys_special()
{
        printk("I'm special");
}
```

Ya no existirá una IDT global, sino que cada proceso tendrá su propia IDT. Así, los procesos "especiales" tendrán la entrada 0x3e redireccionada a *special_handler* mientras que los que no sean especiales, la tendrán redireccionada a *no_special_handler*. La IDT que tiene que tener configurada el procesador es la de current.

b) Indica que modificaciones se tienen que hacer en el task_struct de un proceso

c) Indica qué modificaciones se tienen que hacer en sys_fork

asumimos que cualquier proceso, al clonarse no es especial?
Despues de copiar la task_union del padre, modificar el campo de special para que valga 0

d) Escribe el código de la función int sys_SetSpecial(int value)

```
int sys_SetSpecial(int value) {
    current()-> special = value;
    return 0;

}
```

e) ¿Qué otra(s) función(es) del sistema operativo se tiene(n) que modificar? ¿Cómo se modifica(n)?

crear un wrapper: int special(){} que en funcion de si es especial o no, llame a un handler o el otro
crear otro wrapper para la llamada a sistema SetSpecial, crear un handler que llame a sys_SetSpecial

f) ¿Se tiene que modificar el boot del sistema? En caso afirmativo, ¿Cómo?

Implementación 2: Tabla de páginas

En esta implementación vamos a cambiar la tabla de páginas del proceso para que, dependiendo de si el proceso es o no especial, tenga mapeada la página física que contiene el código de la rutina de servicio correspondiente.

La entrada 0x3e ahora apunta a este handler:

```
ENTRY (maybe_special_handler)
       SAVE_ALL
       call 0x2E8000
       EOI
       RESTORE_ALL
       Iret
```

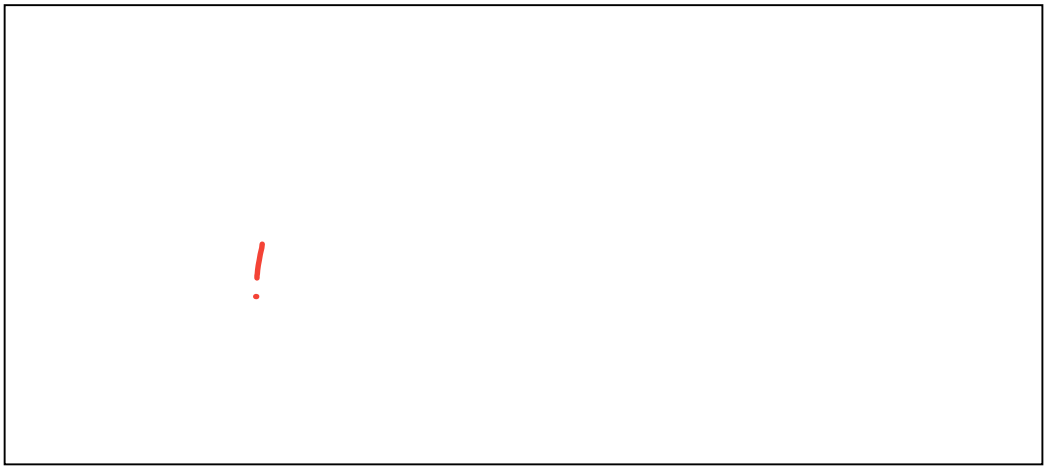
La dirección 0x2E8000 corresponde a la dirección lógica donde está mapeada la rutina de servicio correspondiente. Para los procesos “especiales”, la dirección 0x2E8000 estará mapeada a la dirección física correspondiente a la función “sys_special” (del apartado anterior). En caso de procesos “no especiales”, estará mapeada a la dirección física correspondiente a la función “sys_no_special” (del apartado anterior).

g) ¿Que característica(s) se tiene(n) que dar para que este mecanismo funcione?

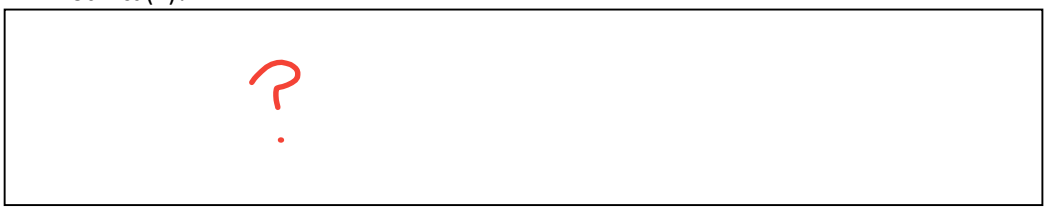
h) Indica que modificaciones se tienen que hacer en el task_struct de un proceso

i) Indica qué modificaciones se tienen que hacer en sys_fork

j) Escribe el código de la función int sys_SetSpecial(int value)



k) ¿Qué otra(s) función(es) del sistema operativo se tiene(n) que modificar? ¿Cómo se modifica(n)?



l) ¿Se tiene que modificar el boot del sistema? En caso afirmativo, ¿Cómo?

4. (2.5 puntos) Difícil

El Señor Baka Baka, aunque tiene una orden judicial internacional que le prohíbe comprar cualquier componente informático, ha conseguido un ordenador en el mercado negro y amenaza con hacer la vida imposible, otra vez, a los estudiantes de SOA2. En este caso pretende escribir el código de un gestor de disco.

Para programar una entrada/salida hacia este disco, se tiene que poner en el registro dma0 la dirección de memoria de una estructura que tiene los siguientes campos:

```
struct dma_IO
{
       int32_t sector;      // First accessed sector
       int32_t number_of_sectors; // Number of sectors to access
       void * buffer;       // Address of the data to read/write
       int8_t operation;    // Operation is read (1) or write (2)
};
```

Y en el registro dma1 se tiene que escribe un 1.

Una vez se ha acabado la entrada/salida, el disco duro envía una interrupción al procesador (la número 38 en decimal) indicando, en el registro dma0 el código de finalización de la E/S (0, correcto o 1, incorrecto).

Nombre:
DNI:

Puedes suponer que este gestor se está desarrollando para un sistema de ficheros que incluye todas las optimizaciones vistas en clase. Además, en el único disco duro que esta conectado a la máquina, los sectores son de 512 bytes y se ha formateado con EXT2 con un tamaño de bloque de 4KBs. Este disco duro es de 500MBs.

El sistema operativo donde se va a ejecutar este gestor tiene soporte para múltiples procesos pero no tiene soporte para threads.

a) ¿Qué componente del sistema operativo le envía peticiones al gestor de disco?

b) Escribe la sucesión de llamadas que se hacen desde que un usuario ejecuta read hasta que llega la petición al gestor de disco

c) Describe que campos tiene que tener una estructura IORB para hacer una lectura de

d) Escribe el código del gestor de disco teniendo en cuenta que el acceso a dispositivo físico funciona mediante interrupciones

e) Escribe el pseudocódigo que se tiene que poner en la rutina de servicio de la interrupción 38.

f) ¿Cómo se sabe desde la rutina de interrupción cual es la petición que se está procesando actualmente?

g) Desde la interrupción de disco, ¿se podría escribir directamente los datos traídos de disco en el buffer de usuario?

Información del Sistema Operativo

Cada proceso contiene una estructura para guardar su espacio de direcciones, consistente en un directorio de páginas con una única entrada de válida que es su tabla de páginas. Cada una de las entradas de la tabla de páginas usada por la MMU (*page_table_entry*) contiene, entre otros, los siguientes campos (codificados en una estructura de 32 bits):

- ***present***: *Present flag*, si este bit està a 1, la pàgina està a memòria principal; si està a 0, la pàgina no està a memòria principal i llavors la resta de bits de l'entrada es poden usar pel sistema operatiu.
- ***pbase_addr***: *Address field,* camp amb els 20 bits més significatius de l'adreça física d'un marc de pàgina (frame).
- ***user***: *User/Supervisor flag*, bit per indicar si la pàgina és d'usuari o sistema
- ***rw***: *Read/Write flag*, bit per indicar els permisos d'accés de la pàgina (*Read/Write* o *Read*)

La IDT es un vector de 256 posiciones apuntado por el registro del procesador rIDT (accesible solo en ensamblador). Cada una de las entradas de la IDT definidas en la estructura struct IDT, contiene:

- ***proc_address***: dirección de la función que se ejecutará cuando se produzca la interrupción o excepción asociada a esta entrada.
- ***priv_level***: nivel de privilegios mínimo desde el que se puede ejecutar proc_address.

El sistema tambien dispone de las siguientes rutinas:

- ***struct task_struct *current():*** Retorna la task_struct del proceso actual.
- ***int alloc_frame()*:** Reserva un frame de memoria física.
- ***void free_frame (int frame)***: Libera un frame de memoria.
- ***page_table_entry * get_PT(struct task_struct *t):*** Retorna la tabla de paginas del proceso *t*.
- ***page_table_entry * get_DIR(struct task_struct *t):*** Retorna el directorio de páginas del proceso *t*.
- ***set_CR3 (page_table_entry * dir):*** Sobreescribe el registro CR3 con el nuevo directorio de páginas *dir*, provoca una invalidación de la TLB.
- **i*nt get_frame (page_table_entry *pt, int logical_page)***: Retorna el frame asignado a una página lógica en la tabla de páginas de un proceso determinado.
- ***int set_ss_page (page_table_entry *pt, int logical_page, int frame):*** Asigna un *frame* a una página lógica de la tabla de paginas *pt*.
- ***int del_ss_page (page_table_entry *pt, int logical_page)*:** Borra una entrada de la tabla de páginas.