

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 contiene el master boot record?

b) ¿Qué es un bootloader?

c) ¿Por qué se tiene que pasar a modo protegido antes de ejecutar por primera vez el proceso Init?

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

e) ¿Por qué la pila que se utiliza en modo sistema es diferente a la que se utiliza en modo usuario?

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

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

h) ¿Qué 3 formas hay de pasar parámetros a una llamada al sistema?

i) ¿Qué es una tabla de páginas?

j) ¿Para qué sirve el registro cr3?

k) En un procesador de 32 bits, ¿Cuántas entradas tiene la tabla de páginas?

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

m) ¿Por qué esta solapado el PCB de un proceso con su pila de sistema en Linux?

n) Escribe el código de la función task\_switch vista en teoría

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?

p) ¿Qué recursos son propios de un thread?

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

# SOA2 (20/06/2023)

Nombre:

DNI:

r) ¿Qué es un device descriptor?

s) ¿Qué es un driver de dispositivo?

t) ¿Para qué sirve un virtual file system?

## 2. (2.5 puntos) Moderadas

a) Calcula el espacio ocupado por las tablas de páginas de un proceso que ocupa 800 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 y con direcciones de 32 bits en las que los 10 bits de más pesos 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.

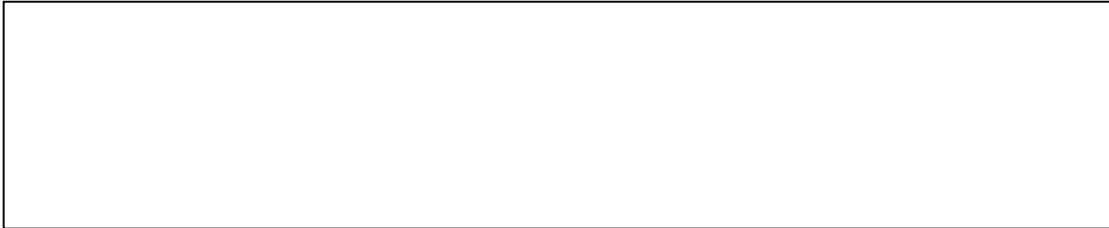
b) Calcula el porcentaje de disco que ocupa una FAT si el disco es de 2 GBs, los bloques de disco son de 4 KBs y los índices a bloques de datos son de 4 bytes.

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

d) Dibuja el contenido de la pila de sistema cuando se está ejecutando el proceso 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



### 3. (2.5 puntos) Complejo

Queremos modificar, en ZeOS, la llamada al sistema sys\_fork para que utilice el disco al crear un proceso en vez de utilizar memoria física. Así, cuando se creen un proceso, dentro del bucle de copia de sys\_fork de la zona de datos del proceso padre al hijo, en vez de asignar páginas de memoria física al proceso hijo, copiará las páginas de datos del proceso padre en un fichero y luego, cuando haya un fallo de página en el proceso hijo, mapeará una página física en el espacio de direcciones del proceso donde traerá los datos del disco. Así, en el caso de que una página no sea accedida, no se le asignará nunca una página de memoria física.

Para forzar estos fallos de página, en el bucle de copia marcaremos la página lógica como no presente.

Recuerda que, en la excepción de fallo de página, en el registro cr2 el procesador deja la dirección lógica en la que se ha producido el fallo de página. También recuerda que un proceso en ZeOS tiene 256 páginas de kernel, 8 de código y 20 de datos (en ese orden y sin agujeros en medio) y que las páginas de kernel y código están compartidas entre todos los procesos y, por tanto, solamente crearemos en disco las 20 páginas de datos.

El fichero que utilizaremos se accede mediante el file descriptor fd y para calcular el offset dentro del fichero donde se tiene que leer/escribir una página utilizaremos la fórmula:

$$\text{Offset} = (\text{PID} * \text{NUM\_PAG\_DATA} + (\text{PAG\_FALLO} - \text{PAG\_LOG\_INIT\_DATA})) * \text{PAGE\_SIZE};$$

En la que PAG\_FALLO es el identificador de la página lógica en la que se ha producido el fallo de página.

Para mover el puntero de lectura/escritura dentro de ese fichero para ponerlo al principio de la página que se quiere leer/escribir utilizaremos:

```
sys_seek(fd, offset, SEEK_SET);
```

Donde offset es el offset calculado con la fórmula anterior.

a) Escribe el bucle de copia de sys\_fork con esta modificación



# SOA2 (20/06/2023)

Nombre:

DNI:

- b) ¿Cómo se detecta, dentro de la rutina de servicio de fallo de página, que el acceso es o no correcto (segmentation fault)?

- c) ¿Cómo se calcula PAG\_FALLO en la rutina de servicio de fallo de página?

- d) Escribe el pseudocódigo de la rutina de servicio de fallo de página

- e) ¿En qué casos realmente se ahorra memoria física con esta nueva implementación?

- f) ¿En qué casos no se ahorra memoria física? Pon un ejemplo de código de usuario en el que no se ahorraría memoria física en el proceso.

- g) Imagina que tenemos un proceso P1, el cual ha accedido previamente a todo su espacio de direcciones lógicas, que ejecuta el siguiente código:

```
fork();  
fork();
```

¿Cuántos fallos de página se producirán, en total, al ejecutar estas dos llamadas al sistema?

- h) En el escenario descrito en g), cuanto espacio de disco se utilizará para crear los nuevos procesos.

- i) ¿Se tiene que modificar `sys_exit` para borrar la zona de disco correspondiente al proceso que ha finalizado?

- j) A parte de para ahorrar memoria, ¿esta implementación presenta alguna ventaja más?

#### 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. Este disco funciona de la siguiente forma: para programar una entrada/salida tiene 4 registros: `dma0`, `dma1`, `dma2` y `dma3`. En el registro `dma0` se tiene que indicar el índice del primer sector de disco al que se quiere acceder. En el registro `dma1` se tiene que indicar cuantos sectores se tienen que acceder. En el registro `dma2` se tiene que indicar la dirección de memoria con la que se va a trabajar y, por último, en el registro `dma3` se indica si la operación es de lectura o es de escritura. 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).

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 es multiproceso monohilo.

- 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

# SOA2 (20/06/2023)

Nombre:

DNI:

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

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

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

- f) Desde la interrupción de disco, ¿se podrían 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 está a 1, la página está a memoria principal; si está a 0, la página no está a memoria 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 página (frame).
- **user**: *User/Supervisor flag*, bit per indicar si la página és d'usuari o sistema
- **rw**: *Read/Write flag*, bit per indicar els permisos d'accés de la página (*Read/Write o Read*)

El sistema también 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.
- **int 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.