

Nombre:

DNI:

Examen final de teoría

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

1. (3 puntos) Preguntas cortas

Contesta si son ciertas o falsas las siguientes afirmaciones justificando brevemente tu respuesta:

- a) Cualquier llamada a función desde modo usuario cambia el nivel de privilegio en un procesador de Intel.

- b) La estructura que nos permita encontrar la pila a usar cuando entramos a ejecutar en modo sistema es el PCB.

- c) En una llamada al sistema, el contexto hardware lo guarda automáticamente el procesador mientras que el contexto software se tiene que guardar en el handler de la llamada al sistema.

- d) La pila usada para guardar el contexto hardware podría ser la misma que la pila usada en modo usuario.

- e) En un procesador de 32 bits, con un sistema operativo que utiliza páginas de 4 KBs, el TLB utiliza los 20 bits de menos peso de la dirección lógica como identificador de página lógica.

SOA2 (17/06/2021)

Nombre:

DNI:

- f) Durante un cambio de contexto se tiene que modificar el registro esp del procesador para que apunte a la cima de la pila de sistema del siguiente proceso a ejecutar.

- g) Al crear un proceso nuevo con *fork* no hace falta copiar la pila de sistema del proceso padre al proceso hijo, ya que esta copia se realiza cuando se crea el espacio de direcciones lógico del proceso hijo.

- h) La finalidad del Inodo es almacenar el número de referencias a un fichero en un sistema de ficheros con una estructura de directorios en árbol.

- i) Gracias al Major y al minor, se consigue que el sistema operativo sea independiente del dispositivo físico.

- j) En un sistema operativo actual, la copia del espacio de direcciones del kernel de la tabla de páginas del proceso padre al hijo representa la mayor parte del tiempo de ejecución de *fork*.

- k) Al crear el proceso *idle* el valor asignado como EBP es 0 pero podría ser cualquier valor.

SOA2 (17/06/2021)

Nombre:

DNI:

- l) La página 1 contiene la dirección de memoria 0x1234.

- m) En Linux puedo usar el registro ESP para encontrar el PCB del proceso que está en ejecución.

- n) La tabla de canales guarda los dispositivos virtuales accesibles para el proceso.

- o) El sector es la unidad mínima de transferencia del sistema de ficheros

- p) La asignación contigua de bloques a ficheros tiene un mejor tiempo de acceso aleatorio que la asignación indexada multinivel.

- q) La buffer cache de un sistema de ficheros es solamente necesaria si existe un virtual file system.

SOA2 (17/06/2021)

Nombre:

DNI:

- r) El descriptor de dispositivos contiene, entre otras cosas, el código dependiente de un dispositivo físico.

- s) En un sistema operativo que permite varios threads por proceso, se tiene que implementar, necesariamente, un planificador a dos niveles.

- t) La implementación del wrapper de una llamada al sistema depende de como se ha implementado el handler de las llamadas a sistema.

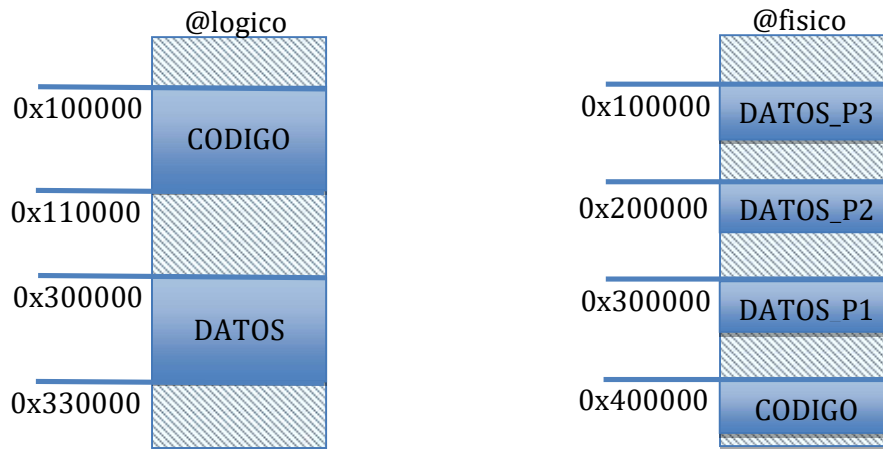
2. (2 puntos) Memoria

Tenemos una máquina, con arquitectura de 32 bits, que gestiona la memoria mediante una única tabla de páginas **sin directorio**, y donde las páginas ocupan 4KB. En esta máquina instalamos un sistema operativo tipo ZeOS, en la que todos los procesos de usuario tienen el mismo espacio de direcciones lógico (como el de la siguiente figura). En un momento determinado el proceso P2 está en ejecución y encontramos el siguiente contenido en el espacio de direcciones físico correspondiente a 3 procesos distintos:

SOA2 (17/06/2021)

Nombre:

DNI:



a) En esta máquina, ¿cuántas entradas tiene la tabla de páginas?

b) Indica el contenido de la tabla de páginas (posición->valor) para mapear la zona de datos del proceso actual en este espacio de direcciones (al menos la 1ª y la última dirección de la zona).

c) Completa el código siguiente para copiar la zona de datos del proceso actual a la zona de datos del proceso P3 usando una única página libre del espacio lógico. NOTA: Puedes usar las rutinas del final del ejercicio que creas convenientes.

```
for( i = 0; i < .....; i++) {
```

```
    copy_data(
```

Nombre:

DNI:

d) ¿Cuántos fallos de TLB generará el código del apartado anterior como mínimo?

3. (3 puntos) Sistemas de ficheros

Supón que en una máquina de 32bits tienes un sistema operativo como ZeOS y un fabricante muy conocido te deja un dispositivo de almacenamiento por bloques. Este fabricante te implementa las siguientes rutinas para acceder a este dispositivo:

int put (void *logical_adress, int block): Guarda 4096 bytes a partir de la dirección pasada como parámetro en el bloque del dispositivo pasado como segundo parámetro. El bloque es un identificador útil para su posterior recuperación.

int get (int block, void *logical_adress): Recupera el bloque del dispositivo identificado por el parámetro *block* y copia su contenido (4096 bytes) en la dirección lógica pasada como segundo parámetro.

Sobre este dispositivo, tienes que diseñar un sistema de ficheros que permita crear ficheros de diferentes tamaños (aunque no muy grandes), modificarlos y borrarlos, así como acceder de forma secuencial y/o directa a cualquier posición de estos ficheros de forma eficiente (minimizando el número de accesos al dispositivo). Los ficheros se organizan en una estructura en grafo mediante directorios que pueden contener ficheros y/o otros directorios. Para ello decides que lo mejor es usar un **sistema de ficheros indexado**. Para cada fichero tendrás, de forma separada, (1) sus bloques de datos y (2) una estructura, que llamaremos *inodo*, para gestionar estos bloques. Este inodo tendrá un **tamaño fijo de 4096 bytes**, contendrá información sobre el fichero y todos los índices directos a los bloques de datos del fichero.

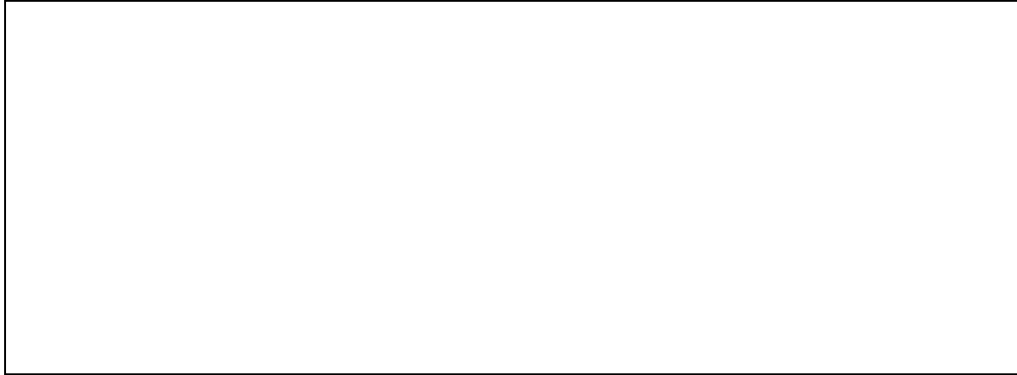
Los procesos de usuario pueden usar la llamada a sistema *open(nombre_de_fichero)* para abrir un fichero previamente creado (o crearlo si no existía), y a continuación usar las llamadas a sistema *read* o *write* para leer o escribir el contenido del fichero. Finalmente pueden usar la llamada a sistema *close* cuando quieran dejar de usar el fichero y borrarlo con la llamada a sistema *unlink(nombre_de_fichero)*.

SOA2 (17/06/2021)

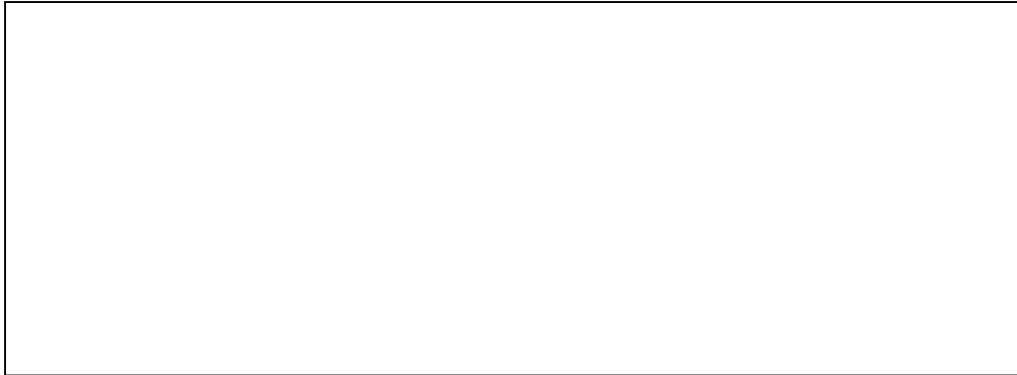
Nombre:

DNI:

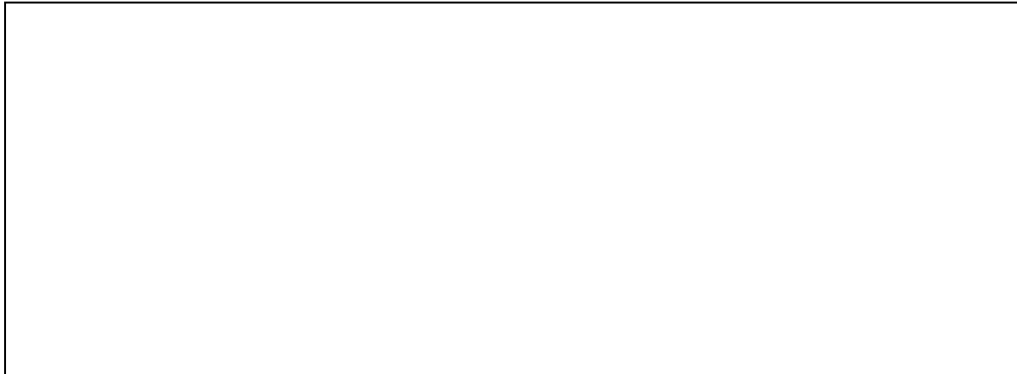
- a) ¿Qué información mínima debes guardar en el inodo para gestionar cada fichero?



- b) ¿Como guardas esta información en el dispositivo?



- c) ¿Cual es el tamaño máximo de un fichero en este sistema?

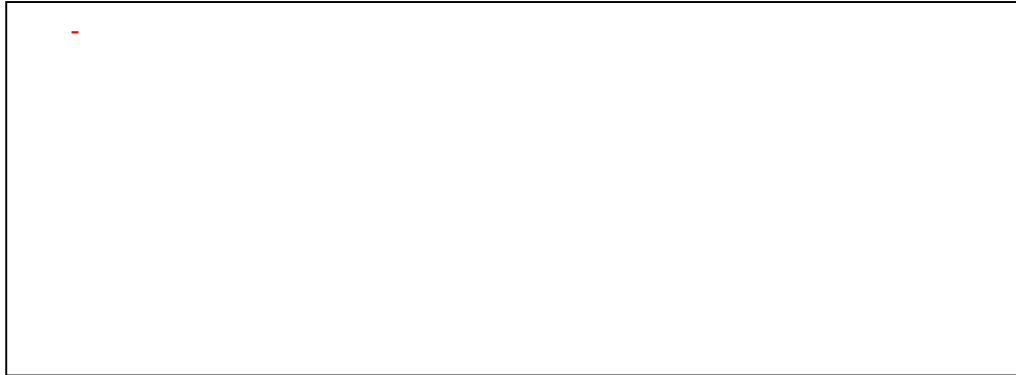


SOA2 (17/06/2021)

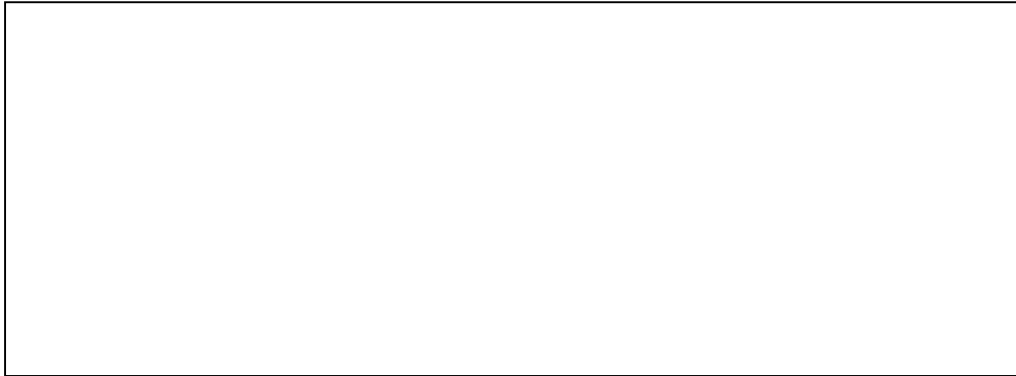
Nombre:

DNI:

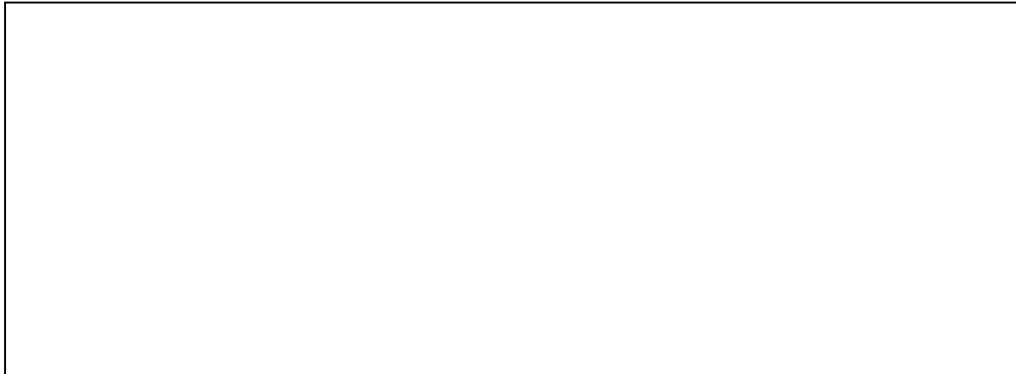
d) ¿Qué información debes guardar dentro del directorio?



e) ¿Como guardas esta información en el dispositivo?



f) El tamaño máximo de un directorio en este sistema ¿es el mismo que el de un fichero?



SOA2 (17/06/2021)

Nombre:

DNI:

- g) ¿Todos los directorios son iguales o necesitas tener alguno con alguna restricción especial?

- h) Cuando un proceso esté trabajando con un fichero ¿Qué información es indispensable tener actualizada en memoria?

- i) Supón el caso en que un usuario quiere escribir 4 bytes ("hola") en un fichero que ya había creado anteriormente y que esta escritura solo afecta al bloque de datos número 42, indica el pseudocódigo necesario de la rutina dependiente para implementarlo:

SOA2 (17/06/2021)

Nombre:

DNI:

4. (2 puntos) Entrada/Salida

Dada la llamada a sistema síncrona *read* de ZeOS queremos modificarla para que sea **asíncrona**. Para ello tienes que añadir un gestor.

a) ¿Qué estructuras de datos nuevas necesitas?

b) Completa el pseudocódigo del gestor, indicando si hay alguna parte bloqueante:

```
for(;;) {
```

```
}
```

c) ¿Cómo quedaría el pseudocódigo de la llamada dependiente de lectura? Indica si hay alguna parte bloqueante.

Nombre:

DNI:

d) ¿Seria necesario añadir alguna llamada a sistema adicional?

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 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*)

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.
- **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.