SOA/SO2

Nombre:
DNI:

Examen final de teoría

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

1. *(2 puntos) Preguntas cortas*

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

a) 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.

b) La estructura TSS solamente se modifica en el BOOT del sistema para apuntar a la cima de la pila del proceso *Init*.

c) Una de las finalidades del *wrapper* de una llamada al sistema es asegurar la portabilidad del código.

d) La IDT es la estructura hardware en la que se almacena el código de las llamadas al sistema.

Nombre:
DNI:

e) Durante la generación de un ejecutable, todas las direcciones que se crean son lógicas ya que indican la dirección, dentro del fichero ejecutable, donde se pueden encontrar las variables y el código.

f) La MMU es un mecanismo hardware cuyo único propósito es traducir direcciones lógicas a físicas.

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

h) El número de frames disponibles en un ordenador depende, exclusivamente, del tamaño de la tabla de páginas de un proceso.

i) El objectivo de tener sistemas operativos multiprogramados es solapar el tiempo de E/S de unos procesos con la ejecución de otros procesos.

j) La dirección de memoria 1234 (en decimal) pertenece a la página 1.

Nombre:
DNI:

2. (2 puntos) Preguntas cortas (II)

Responde justificadamente las siguientes preguntas:

a) El acceso a disco es lento. ¿Qué hace el sistema de ficheros de forma general para evitar acceder a él?

b) ¿Qué mecanismo es fundamental para gestionar la memoria virtual?

c) ¿Qué modificaciones se tienen que hacer en un sistema operativo con entrada/salida sincrona, para que también pueda ofrecer entrada/salida asincrona?

d) ¿Para qué sirve el *major* y *minor* en Linux?.

e) ¿Cómo consigues un acceso compartido a un dispositivo?

f) Comenta una ventaja del sistema de ficheros con asignación contigua.

Nombre:
DNI:

g) ¿Qué es el descriptor de dispositivo?

h) ¿Cómo se asegura que la ejecución de sem_wait es atómica es ZeOS?

i) (,5 puntos) Supón el siguiente fragmento de código:

```
...
int main() {
        ...(1)...
        for(i=0; i<10;i++) {
                pthread_create(...,th1, ...);
        }
        ...(2)...
}
```

```
void th1(void) {
        ...(3)...
        code_A();
        ...(4)...
        code_B();
        ...(5)...
        code_C();
        ...(6)...
        exit();
}
```

Este código crea 10 threads que ejecutan varios trozos de código (*A, B y C*). Usando el mínimo número de semáforos, indica el contenido de los puntos indicados en el código para garantizar que:

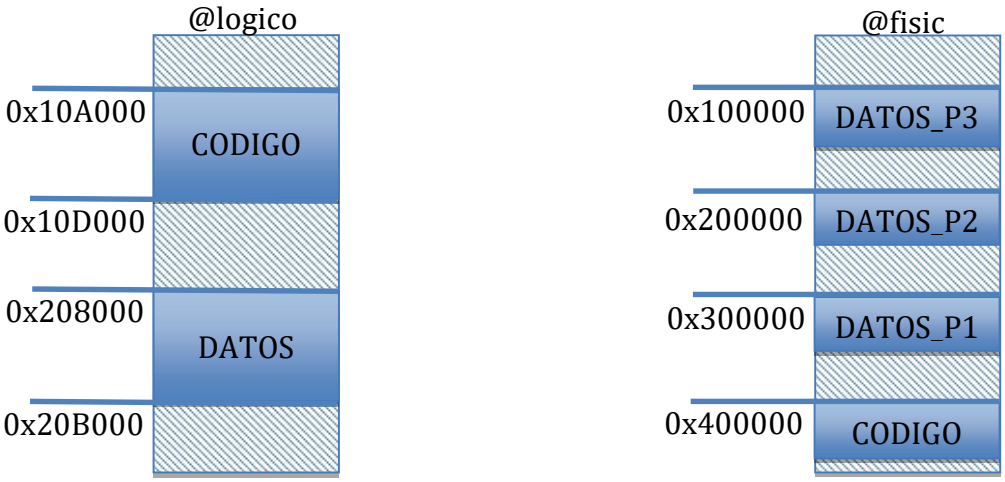
1) los códigos A y C puedan ejecutarse en paralelo con cualquier otro código de otro thread,
2) pero 2 threads no pueden ejecutar el código *B* a la vez.

i. Código (1) y (2)

ii. Código (3), (4), (5) y (6)

3. (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) y en un momento determinado encontramos el siguiente contenido en el espacio de direcciones físico correspondiente a 3 procesos distintos:



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

Nombre:
DNI:

b) Indica el contenido de la tabla de páginas (posición->valor) para mapear la zona de código del proceso 2 en este espacio de direcciones.

c) Indica el código necesario para mapear la zona física de datos del proceso 3 en alguna zona libre del espacio lógico del proceso 2. NOTA: Puedes usar las rutinas del final del examen que creas convenientes.

d) Completa el código siguiente para copiar la zona de datos del proceso actual a la zona de datos del proceso 3 mapeada en el apartado anterior.

```
copy_data(
```

e) ¿Cuantos fallos de TLB generará el código del apartado anterior como mínimo?

Nombre:
DNI:

4. *(2 puntos) Sistemas de ficheros*

La compañía Shamsung nos ha donado un disco de Nueva Generación[1], con tecnología no volátil de 8Mbytes (8.388.608 bytes) de capacidad, cuyos sectores ocupan 4096 bytes. Además nos ha facilitado las siguientes rutinas **síncronas** para acceder al dispositivo:

```
int shame_open();  // Para inicializar el disco
int shame_read(int n, char *buff);
      // Para leer el contenido del sector ´n´ del disco en el buffer ´buff´
int shame_write(int n, char *buff);
      // Para escribir el contenido de ´buff´ en el sector ´n´ del disco
int shame_close();  // Para dejar de usar el disco
```

Sobre este disco queremos montar un sistema de ficheros similar a FAT (encadenado en tabla) con las siguientes características: puede contener ficheros y directorios; los nombres de los ficheros pueden tener una longitud máxima de 255 caracteres; un bloque de este sistema de archivos se corresponde con un sector del disco; y los bloques se identifican con un número de 32 bits. Responde a las siguientes preguntas:

a) Sabiendo que el primer bloque está reservado para el superbloque ¿Cuántos bloques ocupa la FAT?

b) ¿Cómo guardas el espacio libre del disco y donde lo guardas?

c) ¿Qué debe contener una entrada del directorio y cuanto ocupa? (Si su tamaño es variable indica mínimo y máximo)

[1] Concretamente del rodaje de la conocida serie de ciencia ficción *ST: La Nueva Generación* (1987).

d) Suponiendo que el contenido del directorio raíz (/) se encuentra en el primer bloque de datos disponible del SF ¿en qué sector del disco encontramos este directorio?

e) Suponiendo que el directorio raíz contiene un único fichero ¿cuál es el tamaño máximo de este fichero en este disco?

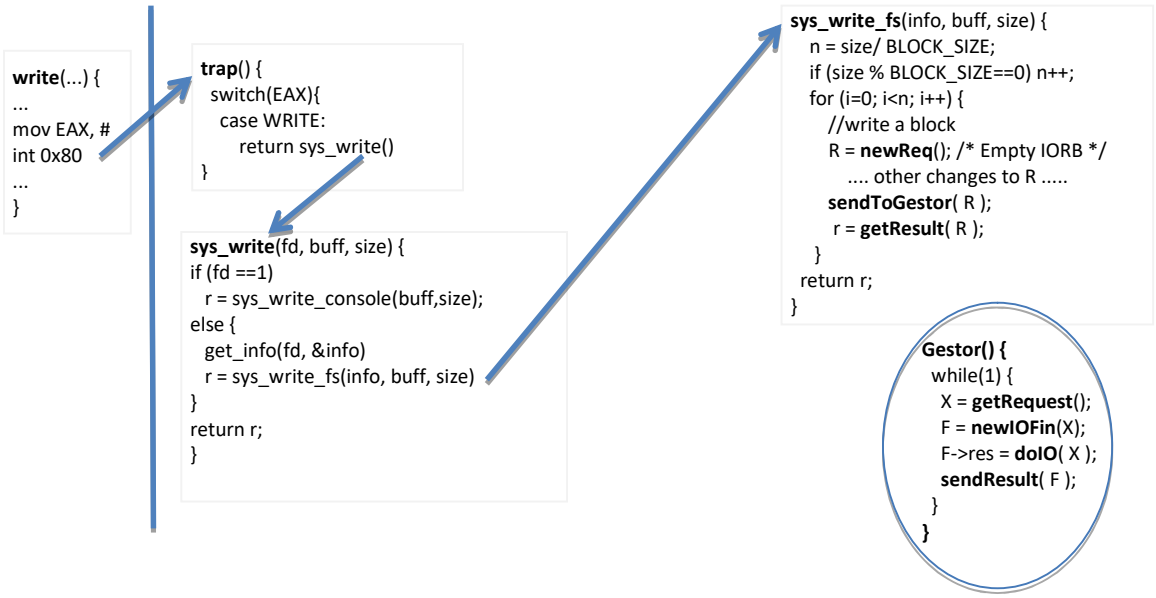
f) Si limitamos el tamaño de los directorios para que ocupen un bloque como máximo ¿Cuál es el número máximo y mínimo de ficheros que puede contener?

g) ¿Este sistema de ficheros sufre el problema de la fragmentación externa? ¿Y de fragmentación interna?

Nombre:
DNI:

5. *(2 puntos) Gestores*

Con el disco del ejercicio anterior en el sistema, el señor BakaBaka quiere implementar un sistema de ficheros para facilitar su uso. Para ello crea un nuevo proceso de sistema que ejecuta la rutina *Gestor*; añade 2 nuevas llamadas a sistema (*open* y *close*) para acceder a los ficheros; añade la función *get_info* que devuelve una estructura con la información dinámica y el bloque inicial de un fichero; y actualiza las llamadas a sistema (*read* y *write*) de ZeOS de forma que pueda acceder a estos ficheros. Por ejemplo, la llamada *write* queda así:



Responde a las siguientes preguntas justificadamente:

a) La rutina *write* de la figura ¿implementa una llamada síncrona o asíncrona?

b) ¿Cuál/cuáles de las funciones destacadas de la figura NO son bloqueantes?

Nombre:
DNI:

c) ¿Qué campos son necesarios en el IORB para enviar una petición al gestor?

d) ¿Cómo intercambiará información la llamada *sys_write_fs* con este nuevo proceso gestor?

e) Indica qué debe hacer la rutina *getResult*.

f) Indica el pseudocódigo de la rutina *doIO* del gestor.

```
int doIO(struct IORB * iorb) {
```

g) ¿Sería correcto mover la instrucción "r = getResult(R);" de la rutina *sys_write_fs* fuera del bucle? ¿Por qué?

h) Suponiendo que las llamadas a sistema read/write definidas fueran síncronas ¿sería posible tener un proceso de usuario que hiciera lecturas/escrituras asíncronas?

i) ¿Qué información debemos devolver en el IOFin?.

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 en memoria principal; si está a 0, la página no está en memoria principal y el resto de bits de la entrada se pueden usar para el sistema operativo.
- ***pbase_addr***: *Address field,* campo con los 20 bits más significativos de la dirección física de un marco de página (frame).
- ***user***: *User/Supervisor flag*, bit para indicar si la página es de usuario o de sistema
- ***rw***: *Read/Write flag*, bit para indicar los permisos de acceso de la página (*Read/Write* o *Read*)

El sistema también dispone de las siguientes rutinas:

- ***struct task_struct *current():*** Devuelve 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):*** Sobrescribe 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.