Nombre:
DNI:

Segundo control de teoría

Justifica todas las respuestas brevemente. **Una respuesta sin justificar se puntuará como no contestada.**

1. *(4 puntos) Preguntas cortas*

a) ¿Modifica la llamada a sistema *mmap* de Linux el espacio de direcciones del proceso que la invoca?

b) (0,5 puntos) Supón un hardware cuya memoria solo tiene 3 frames (4KB) disponibles para el usuario, que ejecuta un sistema operativo que ofrece memoria virtual. Un proceso de usuario genera los siguientes accesos a memoria:
0x128456, 0x12DFF4,0x128460, 0x128464, 0x12DFF0, 0x129CAC, 0x12BFFB, 0x12C000
Si todos los frames estaban disponibles al empezar esta secuencia, ¿cual seria el primer acceso que activaría el algoritmo de reemplazo?

c) ¿Para qué sirve el major i minor en Linux?.

d) ¿Cómo consiguen dos procesos el acceso compartido a un dispositivo?

Nombre:
DNI:

e) ¿Qué técnica se usa para minimizar la latencia de acceso a un dispositivo (por ejemplo un disco)?

f) En un sistema con memoria virtual ¿por qué es necesario bloquear un proceso que ha accedido a una página válida pero no presente?

g) ¿Qué es el descriptor de dispositivo?

h) ¿Cómo avisa la tarjeta de red al sistema operativo de que ha recibido un paquete nuevo?

i) (1 punto) Supón los siguientes fragmentos de código:

```
...
int main() {
        ...(1)...
        if (fork()==0)
            codeA()
        if (fork()==0)
            codeB()
        ⓪
        ...(2)...
        ③
        ...(3)...
        ⑥
}
```

```
...
void codeA() {
        ...(4)...
        ①
        ...(5)...
        ④
        ...(6)...
        exit();
}
```

```
...
void codeB() {
        ...(7)...
        ②
        ...(8)...
        ⑤
        ...(9)...
        exit();
}
```

Este código crea 2 procesos que ejecutan varios trozos de código (*A y B*). Usando el mínimo número de semáforos, indica el contenido de los puntos indicados en el código para garantizar que la secuencia de ejecución sea:

1) se ejecuta ⓪①②③, en este orden;
2) después ④ y ⑤, en cualquier orden;
3) y, finalmente, ⑥

Nombre:
DNI:

i. Código (1), (2) y (3)

ii. Código (4),(5) y (6)

iii. Código (7), (8) y (9)

2. *(2.5 puntos) Sistema de ficheros*

La compañía Shamsung nos ha donado un disco de última generación, con tecnología no volátil de 4Mbytes (4.194.304 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

Nombre:
DNI:

255 caracteres; un bloque de este sistema de archivos se corresponde con un sector del disco; y que un puntero a bloque lo guardamos como un entero (4 bytes). Responde a las siguientes preguntas:

a) Sabiendo que el primer bloque está reservado para el superbloque ¿Cuantos 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)

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 ¿cual 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 ¿Cuantos ficheros puede contener?

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

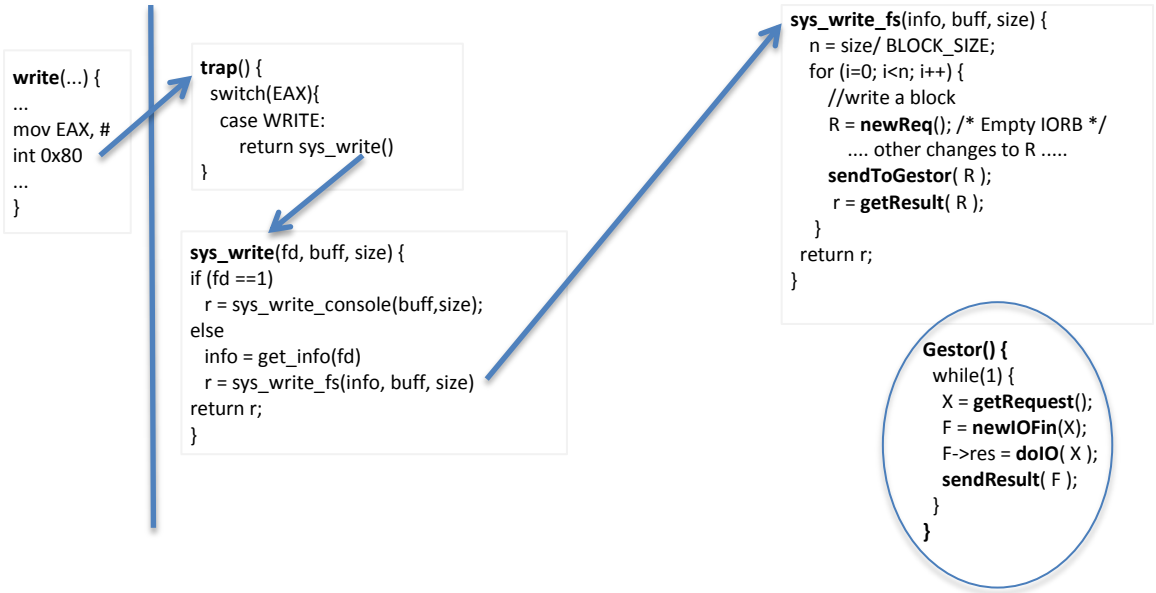
h) Suponiendo que solamente el superbloque se guarda en memoria para futuros usos ¿cuantos accesos a disco son necesarios para situarnos en el 4º bloque de un fichero?

3. *(2.5 puntos) Gestores*

Dado que tenemos un disco en el sistema, queremos añadir un gestor para tratar todas las operaciones sobre él. Para ello el señor BakaBaka crea un nuevo proceso de sistema con el código *Gestor*, añade 2 nuevas llamadas a sistema (*open* y *close*)

Nombre:
DNI:

para acceder a los ficheros, añade la función *get_info* que devuelve el bloque inicial de un fichero y actualiza las llamadas a sistema *read* y *write* de ZeOS de forma que, 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) ¿Cual/cuales de las funciones destacadas de la figura son bloqueantes?

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) Utilizando la nomenclatura O ¿Qué coste tiene la rutina anterior?

g) ¿Qué problema tiene esta solución cuando invocamos 2 veces la llamada a sistema *write* sobre el mismo fichero? ¿A qué se debe este problema?

h) También nos damos cuenta que la rutina *sys_write_fs* no tiene en cuenta el caso en que el tamaño del buffer de usuario no sea múltiplo del tamaño del bloque. Indica brevemente que debería hacerse para tratar este caso.

Nombre:
DNI:

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

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

4. (1 puntos) Memoria dinámica

Dado que esperamos que el gestor del apartado anterior va a recibir muchas peticiones (IORBs y IOFins), decidimos implementar un mecanismo de memoria dinámica específico para estas peticiones.

En concreto BakaBaka decide crear una región de memoria física de 4KB y usar el algoritmo de Buddy Allocator para gestionar las peticiones. Añade 2 rutinas *newReq*() y *newIOFin*() para crear una nueva petición de memoria para un IORB y un IOFin respectivamente. Suponiendo que el IORB ocupa 20 bytes y el IOFin ocupa 16 bytes, responde a las siguientes preguntas justificadamente:

a) Suponiendo la siguiente secuencia de llamadas:

p = newReq(); q = newReq(); r = newIOFin(); s = newReq();

Indica la cantidad de memoria libre disponible

b) ¿Cuantos IORBs se podrían crear después de la secuencia anterior?

c) En el peor caso, ¿qué porcentaje de esta región de memoria no se podria usar por culpa de la fragmentación?