SO2

Nombre:
DNI:

Segon control de teoría

Todas las respuestas se tienen que justificar brevemente. **Una respuesta sin justificar se dará como no contestada.**

1. *(4 puntos) Preguntas cortas*

a) Indica 2 estructuras de datos relacionadas con sistema de ficheros que estén replicadas en memoria

b) ¿Qué es un dispositivo virtual?

c) Nombra 2 llamadas a sistema que borren entradas de la tabla de canales de un proceso.

d) ¿Para qué sirve la Tabla de Ficheros Abiertos?.

e) Dado un fichero, ¿como sabe el sistema operativo Linux qué operaciones puede realizar sobre él?

Nombre:
DNI:

f) ¿Qué problema/s tiene el gestor de memoria *buddy system*?

g) Dada una petición de memoria (*malloc*) concreta y un bloque de memoria libre mucho mayor en el *Doug Lea allocator*, indica la secuencia de pasos que se producen para devolver una dirección válida a usar.

h) ¿Qué operación se optimiza al usar la llamada a sistema *clone*?

i) ¿Qué inconveniente/s tiene una asignación contigua para un sistema de ficheros?

j) (1 punto) Supón el siguiente fragmento de código:

```
1.  ...
2.  void th1(void) {
3.          ...(1)...
4.          code_A();
5.          ...(2)...
6.          code_B();
7.          ...(3)...
8.          exit();
9.  }
10. void th2(void) {
11.         ...(4)...
12.         code_A();
13.         ...(5)...
14.         code_C();
15.         ...(6)...
16.         exit();
17. }
```

```
18. ...
19. int main() {
20.         ...(7)...
21.         clone(th1, ...);
22.         clone(th2, ...);
23.         ...(8)...
24. }
```

Nombre:
DNI:

Este código crea 2 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 primero se ejecute el *code_A*, por un único thread a la vez (usando una exclusión mutua) y cuando ambos finalicen, se ejecute el *code_B*, y finalmente se ejecute el *code_C*.

i. Código (1) , (2) y (3)

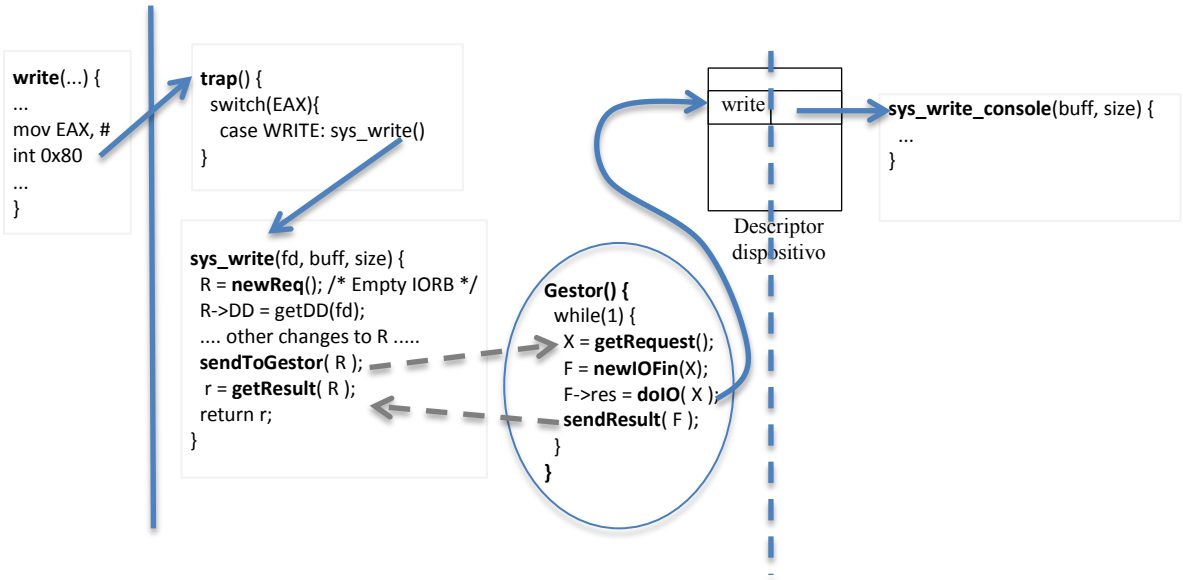
ii. Código (4) , (5) y (6)

iii. Código (7) y (8)

Nombre:
DNI:

2. (3 puntos) Gestores

Supongamos un sistema operativo tipo Zeos en el que para gestionar los dispositivos de forma genérica, añadimos **descriptores de dispositivo** para cada dispositivo y un **gestor** para tratar las llamadas a sistema que acceden a esos dispositivos. Cada descriptor tendrá información sobre el dispositivo y apuntadores a las operaciones válidas sobre él (*open, read, write* y *close* por el momento). Para facilitar la implementación también añadimos una función ***getDD***, que dado un canal retorna su descriptor de dispositivo asociado. En la figura siguiente puedes observar el descriptor *CONSOLA*, el nuevo gestor y el pseudocódigo para las funciones que intervienen en la llamada a sistema *write*:



Responde a las siguientes preguntas justificadamente:

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

b) La rutina *write* anterior ¿es bloqueante?

Nombre:
DNI:

c) ¿Como intercambiará información la llamada *sys_write* con este nuevo proceso gestor?

d) ¿Qué debe hacer la rutina *sendToGestor*? ¿Es bloqueante?.

e) La variable *R* es un puntero a una estructura IORB y contiene un campo *DD* con un puntero al descriptor de dispositivo a usar pero ¿qué otros campos son necesarios?

f) Indica qué debe hacer la rutina *getResult* y si es bloqueante.

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

Nombre:
DNI:

h) Indica qué debe hacer la rutina *getRequest* y si es bloqueante.

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

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

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

3. *(3 puntos) Sistema de ficheros*

Disponemos de un disco de 20MiB (20480KiB), cuyos sectores ocupan 512 bytes. Sobre este disco montamos un sistema de ficheros (SF) cuyo tamaño de bloque es de 4096 bytes.

Este sistema de ficheros es similar a FAT (encadenado en tabla) usando 32bits para cada elemento de la tabla. Su zona de metadatos está ubicada en el principio del disco y a continuación la zona de datos. En la zona de datos solo puede contener ficheros y directorios. Los nombres de los ficheros pueden tener una longitud máxima de 255 carácteres (bytes). Puedes asumir que los valores enteros ocupan 4 bytes. Responde a las siguientes preguntas:

Nombre:
DNI:

a) ¿Qué porcentaje del disco ocupa la FAT?

b) Suponiendo que cada fichero ocupa como mínimo 1 bloque, ¿cuantos ficheros diferentes puede contener este sistema de ficheros como máximo?

c) ¿Cuanto ocupa una entrada de un directorio? (Indica mínimo y máximo)

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

Nombre:
DNI:

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

f) ¿Qué contenido encontraremos siempre en un directorio cualquiera de este SF?

g) Suponiendo que el fabricante del disco te proporciona una función:

*int read_sector(int sec_num, char *buff)*

para leer un sector concreto del disco (512bytes) y guardarlo en *buff*. Indica el pseudocódigo de la función para leer un bloque de datos.

```
int read_block(int bnum, char *buff) {
```

h) ¿Cuantos sectores de la zona de datos se leerán al realizar la llamada a sistema siguiente?

read(fd, buff, 513);