Nombre:
DNI:

Segundo 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) Supón el siguiente código de usuario:

```
1:  int A=0;
2:  void th(void) {
3:      sem_wait(1);
4:      A++;
5:      exit();
6:  }
7:  int main() {
8:      sem_init(1, 0);
9:      ...
10:     clone(th, pila1);
11:     sem_signal(1);
12:     ...
13: }
```

Suponiendo que no hay errores. ¿Cual será el valor de la variable A en la linea 12?

b) Anton Baka Baka quiere ejecuctar el siguiente fragmento de código en un sistema Linux:

```
int main() {
        int * x = sbrk(sizeof(int)*2);
        x[2] = 42;
}
```

¿Es correcto?¿Qué ocurrirá al ejecutarlo?¿Por qué?

Nombre:
DNI:

c) ¿Qué tipo de comunicación estaremos usando en un programa que usa sockets pero no usa la llamada *accept*?

d) ¿Qué estructuras internas básicas encontramos en la implementación de sockets?

e) ¿Qué es un bloque de datos y para qué se usa?

f) ¿Para qué sirve y donde se encuentra el superbloque de un sistema de ficheros?

g) Indica 2 estructuras de datos relacionadas con sistema de ficheros que esten replicadas en memória

h) ¿Qué es un dispositivo virtual?

i) ¿En qué situación/es devolverá error una operación de E/S sobre una pipe sin nombre?

j) ¿Qué estructura de Linux relacionado con E/S permite a 2 procesos compartir el acceso a un mismo dispositivo?

2. *(2,5 puntos) Sistema de ficheros (SF)*

Supón que tienes un disco con sectores de 512 bytes y cuyo fabricante te proporciona el código para la rutina *read_sector(int sector_id, char *buffer)* que lee un sector concreto y lo almacena en la posición de memoria pasada como parámetro. Decides usar este disco en el sistema operativo y fijas tu tamaño de bloques de datos en 4096 bytes.

a) Indica el pseudo-código de la rutina *read_bloque(int bloque_id, char *buffer)* que te permita leer un bloque de datos de este disco.

Nombre:
DNI:

b) Sobre este disco implementas un sistema de ficheros con una estructura de directorios en arbol (no grafo) donde para cada fichero tienes **1 único bloque** de índices apuntando a los bloques de datos de este fichero.
   i. ¿Qué información debe contener cada entrada de directorio en este SF?

   ii. ¿Cual es el tamaño máximo de un fichero en este SF suponiendo que con un número de 32 bits puedo direccionar todos los bloques del disco?

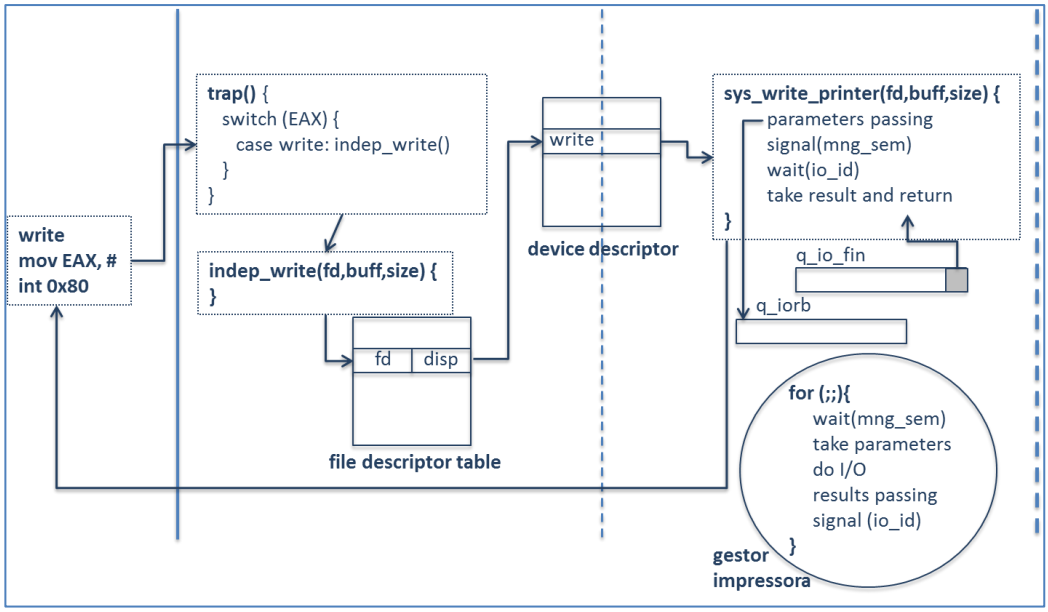
   iii. Si el disco tiene una capacidad de 50Mb ¿Cual es el número máximo de ficheros que podríamos tener?

   iv. Si los ficheros pudieran tener **2 bloques de indices** y **sin modificar el directorio** anterior ¿Cual sería el tamaño máximo de un fichero en este SF?

Nombre:
DNI:

3. *(2 puntos) Gestores*

Tenemos un sistema operativo con la gestión de E/S implementada, así como un dispositivo impresora gestionado mediante un gestor que utilizamos para realizar operaciones de escritura síncronas, tal y como se observa en la siguiente figura:



Además también tenemos las rutinas de bajo nivel necesarias para poder trabajar con dos tipos de sistemas de ficheros diferentes: FAT i EXT2.

En este sistema tienes que añadir dos gestores nuevos: uno para gestionar las peticiones de FAT y otro para gestionar las de EXT2.

a) ¿Cuantos semáforos en total será necesario añadir para sincronizar las rutinas dependientes con los gestores?

b) A la hora de escribir un valor en un fichero ¿qué rutina sería la encargada de localizar el bloque actual a modificar en el fichero?

Nombre:
DNI:

c) Para pasar las peticiones de acceso a fichero desde las rutinas dependientes al gestor, ¿cuantas colas de IORBs son necesarias?

d) Cuando estamos pensando el interfaz de usuario para acceder a estos sistemas de ficheros nos planteamos si usar una interfaz síncrona o asíncrona. Escoger una o la otra, ¿cambia la implementación de la rutina dependiente?

4. (1.5puntos) Memoria dinámica

a) Supón que en tu sistema operativo ZeOS implementas un *Buddy allocator* para gestionar memoria dinámica y como espacio de trabajo le asignas una región contigua de memoria de 16Kb. Dada la siguiente secuencia de llamadas a *malloc que se ejecutan una detrás de la otra*, señala las peticiones que el Buddy allocator podrá servir y las que no:

1) *p = malloc(4096);*
2) *q = malloc(9000);*
3) *r = malloc(5000);*
4) *s = malloc(5000);*
5) *t = malloc(3000);*

b) ¿Puedo realizar algun *malloc* más? ¿de cuanto?

c) Supón que ahora haces un *free(p).* ¿Qué puntero/punteros deberías liberar para activar la operación de *coalescing*?