	**SISTEMAS OPERATIVOS. CUESTIONES** **11 de Septiembre de 2015.**
	Nombre_____DNI_____ Apellidos_____ Grupo____

Contestad las cuestiones en las hojas de enunciado y a los problemas en hojas aparte.

Cuestión 1. (0,75 puntos) Cuando se pulsa un botón en el ratón o éste se mueve se escribe el incremento en **x** en el registro del controlador ubicado en `MOUSE_BASE`, el incremento en **y** en `MOUSE_BASE+1` y el código de botón pulsado en `MOUSE_BASE+2`. Se desea habilitar la comunicación con el ratón a través de un fichero de dispositivo, de forma que leyendo de dicho dispositivo se almacene en `buffer[0]` el botón pulsado, en `buffer[1]` el incremento en **x**, y en `buffer[2]` el incremento en **y**. Suponiendo todas estas constantes ya definidas, completar la función `device_read` del módulo correspondiente:

```
static ssize_t device_read(struct file *filp, char *buffer,
                           size_t len, loff_t *offp) {









  return len;
}
```

Cuestión 2. (0,75 puntos) Supongamos un módulo que implementa la siguiente función para un dispositivo tipo carácter, y que dicha función se configura como operación de lectura del driver del dispositivo:

```
static ssize_t device_read(struct file *filp, char *buffer,
                           size_t length, loff_t * offset);
```

Supongamos que el módulo se compila y se instala, y que se crea un fichero de dispositivo con este módulo (usando la orden `mknod`). (a) Poner un ejemplo de orden Linux en que se ejecutaría automáticamente esta función. (b) ¿Qué debe almacenar la cadena buffer? (c) ¿Qué sentido tiene utilizar la función `put_user`(…), o equivalentemente `copy_to_user`(…) dentro de `device_read`?

Cuestión 3. (0,75 puntos) Un disco tiene montado un sistema de ficheros tipo FAT con bloques de 1 KB. Se va a borrar el fichero `c:\tmp\coredump`, que reside en los bloques 10,12 ,14 y 16. La información del directorio `tmp` está en el bloque 9, y la información del directorio raíz (`c:\`) está en una zona reservada justo tras la tabla FAT (es decir, no ocupa ningún bloque de datos).

a. Indique qué estructuras (FAT, bloques de datos, directorio raíz) se modifican en el borrado del fichero (y en qué consiste la modificación).

b. Si la operación de borrado se queda a medias por un apagón del sistema (por ejemplo, se ha modificado la FAT pero no el resto de estructuras o viceversa), ¿en qué estado podría quedarse el sistema de ficheros? Razonar las consecuencias de cada escenario y si sería posible volver a dejar el sistema de ficheros en un estado correcto.

Cuestión 4. (0,75 puntos) Memoria En un sistema con memoria virtual paginada explicar cómo funciona el algoritmo LRU indicando la información que almacena para cada marco de página y cómo la utiliza, número de fallos de página, … Usar como ejemplo una memoria física de cuatro marcos de página (inicialmente los marcos de página están vacíos) y la siguiente cadena de referencias:

	a	d	a	b	e	d	c	e	d	a	b	c	f	g	d	e
M0																
M1																
M2																
M3																

(justificación)

Cuestión 5. (0,75 puntos) En el mapa de memoria de un proceso hay regiones con distintas características. Indique qué significa cada una de las siguientes y explique con un ejemplo cómo las trata el sistema operativo: a) región privada, b) región compartida, c) región con soporte en fichero, d) región sin soporte.

Cuestión 6. (0,75 puntos) En un sistema operativo se dispone de semáforos y cerrojos, pero no de variables condicionales. Respetando la semántica vista en clase, proponga una implementación de las funciones `varcond_init`, `varcond_wait` y `varcond_signal` usando semáforos y cerrojos (complete el tipo `varcond_t` como considere oportuno).

```
typedef struct {
     sem_t s;
     mutex_t m;




} varcond_t;
```

```
int varcond_init(varcond_t *vc) {




}
```

```
int varcond_wait(varcond_t *vc, mutex_t*
mtx) {









}
```

```
int varcond_signal(varcond_t* vc) {









}
```

Cuestión 7. (0,75 puntos). En un sistema tipo UNIX, se ejecuta el siguiente programa, que pretende buscar el valor máximo en un vector de enteros utilizando dos procesos concurrentes:

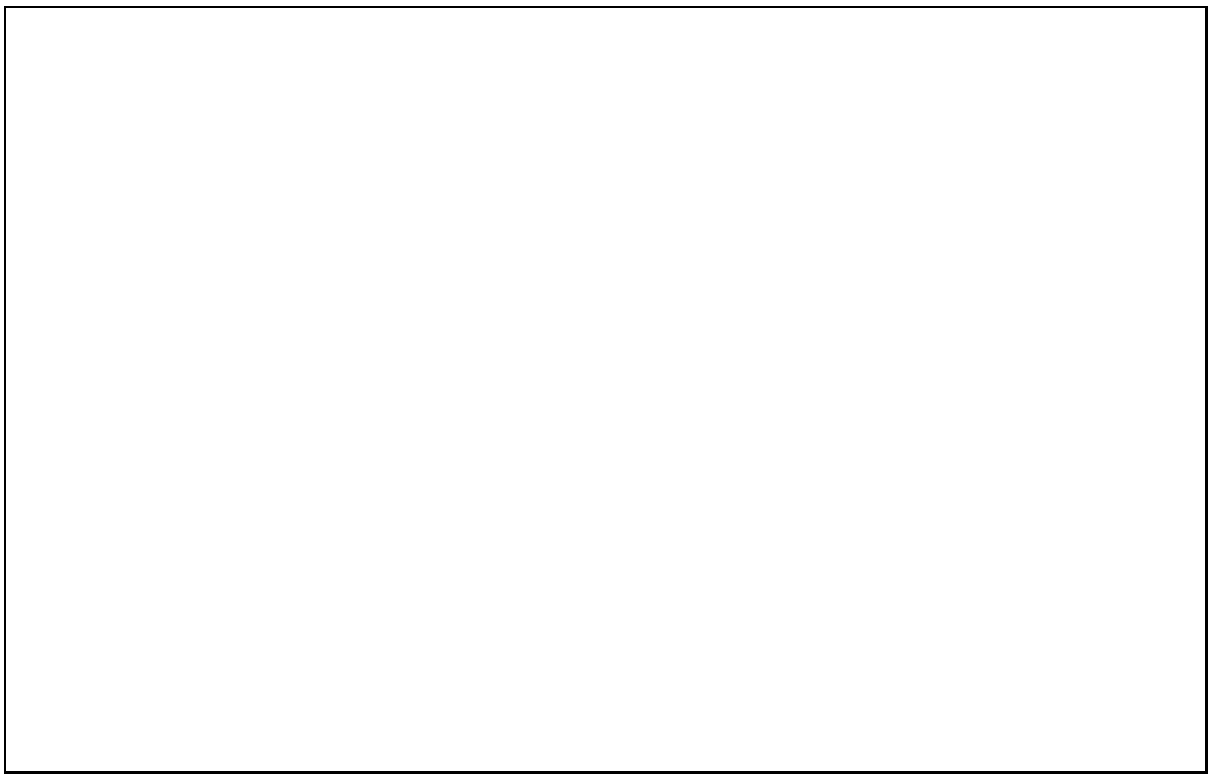
```c
#include <stdio.h>
#include <limits.h>

/*
 *  La función lee un vector del
 *  fichero cuya ruta se pasa como
 *  parámetro. Devuelve un puntero
 *  al vector y el # de elementos.
 */
int* lee_vector(char* fichero,
                int* n_elementos) { ... }

/* Devuelve el máximo del vector en el
rango [start_idx:end_idx) */
int busca_max(int* vector,
              int start_idx,int end_idx) {
  int max_value=-INT_MAX-1;
  int i=0;

  for (i=start_idx;i<end_idx;i++)
    if (vector[i]>max_value)
         max_value=vector[i];

       return max_value;
}

int main(void) {
       int n_elem;
       int* vector;
       pid_t pid;
       int max1=0,max2=0;

 vector=lee_vector("fichero_entrada",
  &n_elem);

 if ((pid=fork())==0) {
 /* Proceso hijo */
   max1=busca_max(vector,0,n_elem/2);
   exit(0);
 }
 else if (pid>0) {
 /* Proceso padre */
  max2=busca_max(vector,n_elem/2,n_elem);
 }
 else {
  fprintf(stderr,"Error en fork()\n");
  exit(2);
 }

 while(wait(NULL)!=pid) {}

 if (max1>=max2)
  printf("max de v es %d\n",max1);
 else
  printf("max de v es %d\n",max2);

 free(vector);

 return 0;
}
```

a. Explique razonadamente por qué este programa no funciona correctamente.
b. ¿Qué mecanismos de comunicación/sincronización del SO podrían utilizarse para conseguir que este programa multiproceso funcionara correctamente? (**Nota:** no es necesario proporcionar una versión corregida del código. Basta con describir una aproximación a la solución.)

Cuestión 8. (0,75 puntos). En un sistema tipo UNIX, se ejecuta el siguiente programa:

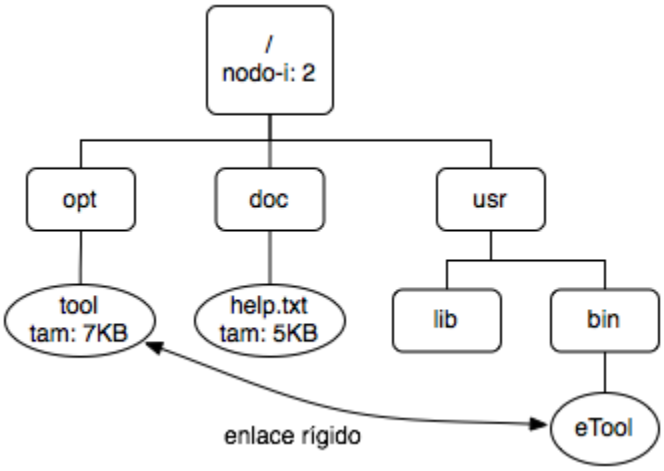
```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>

int main(void)
{
	int fd1,fd2,i,pos;
	char c;

	fd1 = open("outfile.txt", O_CREAT | O_TRUNC | O_RDWR, S_IRUSR | S_IWUSR);
	for (i=0; i < 4; i++) {
		write(fd1, "AAAA", 4);
		pos = lseek(fd1, 0, SEEK_CUR);
		if (fork() == 0) {
			/* Hijo */
			fd2 = open("outfile.txt", O_WRONLY);
			lseek(fd2, pos, SEEK_SET);
			write(fd2, "BBBB", 4);
			close(fd2);
			exit(0);
		} else {
			/* Padre */
			lseek(fd1, 4, SEEK_CUR);
		}
	}
	while (wait(NULL) != -1);
	lseek(fd1, 0, SEEK_SET);
	printf("File contents are:\n");
	while (read(fd1, &c, 1) > 0)
		printf("%c", (char) c);
	printf("\n");
	close(fd1);
	exit(0);
}
```

Explicar qué es lo que hace el programa y si depende del orden en que se ejecuten los procesos. Indicar lo que se mostrará por pantalla. **NOTA**: Recordar que la función `lseek()` devuelve la nueva posición del puntero de posición (bytes desde el comienzo del fichero).



Problema 1. (2 puntos) Un sistema de ficheros tipo UNIX utiliza bloques de disco de 2 Kbytes. Tanto el superbloque como el mapa de bits (para indicar los bloques ocupados) ocupan 1 bloque y la tabla de nodos-i ocupa 10 bloques. Para el direccionamiento de los bloques de datos se utilizan punteros de 32 bits. Cada nodo-i tiene 2 punteros de direccionamiento directo, 1 puntero indirecto simple y 1 puntero indirecto doble.

a. Dada la siguiente estructura de directorios, ¿cuál será el espacio total ocupado en el disco?
b. A partir de la información de la estructura de ficheros y directorios, indique un posible contenido de la tabla de nodos-i y de los bloques de datos ocupador por directorios.
c. Indique qué estructuras (mapa de bits, tabla de nodos-i, bloques de datos) se modificarían para borrar el fichero `/doc/help.txt` y cómo quedarían tras el borrado.



Los directorios se muestran como rectángulos con bordes redondeados y los ficheros regulares como óvalos.

Problema 2. (2 puntos) En un monasterio residen *N* monjes benedictinos que mantienen un riguroso protocolo a la hora de comer. Tras acceder al comedor, ningún monje, por educación, ha de empezar a comer hasta que el resto de monjes se hayan sentado a la mesa. Adicionalmente, al acabar de comer cada monje debe retomar sus quehaceres en estricto orden de llegada al comedor; es decir, aquel monje que llegó primero al comedor será el que comience antes con sus labores tras la comida. Los monjes dedicarán el resto del día a sus labores, por lo que no volverán a entrar al comedor hasta el día siguiente (esto es, en la codificación no hay que preocuparse de la posibilidad de que un monje trate de comer nuevamente antes de que todos los demás hayan terminado).

Cada monje, modelado como un hilo del programa concurrente, se comporta del siguiente modo:

```
void monje(){
    int turno_salida;

    while(1){
        <<labor_en_cuestion>>
        turno_salida=entrar_comedor_y_sentarse()
        <<comer>>
        salir_comedor(turno_salida);
    }
}
```

Implementar las funciones `entrar_comedor_y_sentarse()` y `salir_comedor()` para imponer las restricciones de sincronización anteriormente citadas empleando mútexes, variables condicionales o semáforos, y variables compartidas. La implementación debe permitir que todos los monjes coman simultáneamente.