Material permitido: **Calculadora NO programable.** Tiempo: **2 horas.** N1	**Aviso 1:** Todas las respuestas deben estar razonadas. **Aviso 2:** Escriba sus respuestas con una letra **lo más clara posible.** **Aviso 3: No use *Tipp-ex*** o similares (atasca el escáner).

ESTE EXAMEN CONSTA DE 5 PREGUNTAS

1. (*2 p*) Explique **razonadamente** si las siguientes afirmaciones son verdaderas o falsas.

i) *(1 p)* Las interrupciones son atendidas en modo usuario dentro del contexto del proceso que se encuentra actualmente en ejecución, aunque dicha interrupción no tenga nada que ver con la ejecución de dicho proceso.

ii) (*1 p*) Si un usuario normal ejecuta la orden:

```
$ cat /etc/shadow
```

se le solicitará la contraseña de administración.

RESPUESTA:

APARTADO *i)*: FALSA

(Ver 1.6.3 Interrupciones y Excepciones, p. 15) Las interrupciones software o traps, se producen al ejecutar ciertas instrucciones especiales y son tratadas de forma síncrona. Son utilizadas, por ejemplo, en las llamadas al sistema, en los cambios de contexto, en tareas de baja prioridad de planificación asociadas con el reloj del sistema, etc.

Las excepciones hacen referencia a la aparición de eventos síncronos inesperados, típicamente errores, causados por la ejecución de un proceso, como por ejemplo, el acceso a una dirección de memoria ilegal, el rebose de la pila de usuario, el intento de ejecución de instrucciones privilegiadas, la realización de una división por cero, etc. Las excepciones se producen durante el transcurso de la ejecución de una instrucción.

Tanto las interrupciones (hardware o software) como las excepciones son tratadas en modo núcleo por determinadas rutinas delnúcleo, no por procesos.

APARTADO *ii)*: FALSA

(*V. Ejemplo 2.11*)**.** Si un usuario normal ejecuta la orden

```
$ cat /etc/shadow
```

Se produce un error ya que dicho archivo es propiedad del superusuario y un usuario normal no puede leerlo:

```
cat: /etc/shadow: Permiso denegado
```

Mientras que si hace:

```
$ sudo cat /etc/shadow
```

Se le solicitará la contraseña de administración, y en caso de introducirla correctamente si tiene los permisos necesarios de administración configurados en `/etc/sudoers` se ejecutará la orden `cat /etc/shadow` como si la hubiese hecho el superusuario mostrando por pantalla el contenido del fichero `/etc/shadow`.

2. (*1.5 p*) Conteste razonadamente las siguientes cuestiones:
a) (*1 p*) ¿Qué es el superbloque?
b) (*0.5 p*) Explique cómo procede núcleo cuando la lista parcial de nodos-i libres del superbloque se vacía.

RESPUESTA:
(*V. 8.8.4. El superbloque*)

APARTADO a).
El *superbloque* es una región del disco que contiene metadatos sobre el propio sistema de ficheros. Hay un único superbloque por cada sistema de ficheros y reside al comienzo del sistema de ficheros en el disco, a continuación del área de arranque. El núcleo lee el superbloque cuando monta el sistema de ficheros y lo almacena en memoria hasta que el sistema de ficheros es desmontado. El superbloque contiene básicamente información administrativa y estadística del sistema de archivos, como por ejemplo:

- Tamaño en bloques del sistema de ficheros.
- Tamaño en bloques de la lista de nodos-i.
- Número de bloques libres y nodos-i libres.
- Comienzo de la lista de bloques libres.
- Lista parcial de nodos-i libres.

APARTADO b).
Puesto que el sistema de ficheros puede contener muchos nodos-i libres o bloques de disco libres, es poco práctico mantener en el superbloque una lista de nodo-i libres completa y una lista de bloques libres completa. En el caso de los nodos-i, el superbloque mantiene una lista parcial de nodos-i libres.

Un nodo-i se considera que está libre cuando su campo di_mode contiene el valor 0. Cuando la lista se vacía, el núcleo busca en el disco nodos-i libres para rellenar la lista comenzando por el *nodo-i recordado* y en sentido ascendente de número de nodo-i. El *nodo-i recordado* se define como el nodo-i de mayor número de nodo-i que se ha almacenado en la lista parcial de nodos-i libres desde la última vez que ésta fue rellenada.

3. (*1.5 p*) Responda razonadamente a las siguientes cuestiones:
a) (*0.75 p*) ¿Qué es un proceso ligero? ¿Puede un proceso ligero aprovechar el paralelismo en un sistema multiprocesador?
b) (*0.75 p*) ¿Cuáles son las principales limitaciones de los procesos ligeros?

RESPUESTA:
(Ver 4.9.3 Procesos ligeros)

APARTADO a)
Un proceso ligero es una hebra de usuario soportada en el núcleo. Es una abstracción de alto nivel basada en las hebras del núcleo; puesto que un sistema debe soportar hebras del núcleo antes de poder soportar procesos ligeros. Cada proceso puede tener uno o más procesos ligeros, cada uno soportado por una hebra del núcleo separada (ver Figura 4-8).

Los procesos ligeros son planificados independientemente y comparten el espacio de direcciones y otros recursos del proceso. Pueden hacer llamadas al sistema y bloquearse en espera de una operación de E/S o por el uso de algún recurso. En un sistema multiprocesador, un proceso puede disfrutar de los beneficios de un verdadero paralelismo puesto que cada proceso ligero puede ser encaminado para ser ejecutado en un procesador diferente. Hay ventajas significativas incluso en un sistema monoprocesador, puesto que los bloqueos en espera de un recurso o de una operación de E/S, son para un proceso ligero determinado y no para el proceso entero.

Además de la pila del núcleo y el contexto a nivel de registro, un proceso ligero también necesita mantener el contexto a nivel de usuario, que debe ser salvado cuando el proceso ligero es expropiado. Mientras que cada proceso ligero está asociado con una hebra del núcleo, algunas hebras del núcleo pueden ser dedicadas a tareas del sistema y no tener un proceso ligero.

APARTADO b)
Los procesos ligeros tienen algunas limitaciones. La mayoría de las operaciones de los procesos ligeros, tales como creación, destrucción y sincronización, requieren del uso de llamadas al sistema. Cada llamada al sistema requiere de dos cambios de modo, uno de modo usuario a modo núcleo en la invocación y otro de vuelta a modo usuario cuando se completan. En cada cambio de modo, el proceso ligero cruza una frontera de protección. El núcleo debe copiar los parámetros de la llamada al sistema desde el espacio del usuario al espacio del núcleo y validarlos para protegerlos contra procesos maliciosos. Asimismo, a la vuelta de la llamada al sistema, el núcleo debe copiar los datos de nuevo en el espacio de usuario.

Cuando los procesos ligeros acceden a datos compartidos frecuentemente, la sobrecarga debido a la sincronización puede anular cualquier beneficio que supone el uso de estos procesos.

Cada proceso ligero consume recursos del núcleo de forma significativa, incluyendo memoria física para la pila del núcleo. Por lo tanto un sistema no puede soportar un gran número de procesos ligeros. Además, puesto que el sistema tiene una única implementación de proceso ligero, ésta debe ser lo suficientemente general para soportar la mayoría de las aplicaciones más comunes.

Finalmente, los procesos ligeros deben ser planificados por el núcleo. Aplicaciones que deben transferir a menudo el control de una hebra a otra no pueden hacerlo tan fácilmente usando procesos ligeros.

En resumen, aunque el núcleo suministra los mecanismos para crear, sincronizar y gestionar procesos ligeros, es responsabilidad del programador usarlos juiciosamente. Muchas aplicaciones se pueden atender mejor mediante el uso de hebras a nivel de usuario.

4. *(2 p)* Responda razonadamente a las siguientes cuestiones. Supóngase que en el directorio de trabajo se ejecuta la orden:

```
$ ls -l
```

Que muestra en pantalla la siguiente línea:

```
drw-r----- 1 addison users 1028 Lu 30 12:40 editoriales
```

a) *(0.5 p)* ¿Cuál es el significado de cada uno de los elementos de la línea que se muestra en pantalla?

b) *(1 p)* Explique los permisos de acceso del fichero `editoriales` para propietario, los miembros del grupo al que pertenece el propietario del fichero y para otros usuarios.

c) *(0.5 p)* Muestre y explique la expresión en modo octal de la máscara en modo simbólica.

RESPUESTA
(*Ver 2.6. Configuración de los permisos de acceso a un fichero, p. 67).*

APARTADO a)
El significado de los elementos de esta línea es el siguiente:
`drw-r-----` es la máscara de modo simbólica,
`1` es el número de enlaces duros,
`addison` es el propietario del fichero,
`users` es el grupo al cual pertenece el propietario,
`1028` es el tamaño del fichero en bytes,
`Lu 30 12:40` es la fecha de la última modificación del fichero
y `editoriales` es el nombre del fichero.

APARTADO b)

La máscara de modo simbólica `drw-r-----` (ver Tabla) informa, por orden, de los permisos para el propietario, los miembros del grupo al que pertenece el propietario del fichero y otros usuarios.

	6			4			0		
	110			100			000		
s_9=0	s_8	s_7	s_6	s_5	s_4	s_3	s_2	s_1	s_0
d	r	w	–	r	–	–	–	–	–
Tipo de fichero	Propietario			Grupo			Otros usuarios		

El primer carácter de la cadena de permisos s9=d representa el tipo de fichero. El d significa que es un directorio.

Las siguientes tres letras (s8s7s6) = (rw-) representan los permisos para el propietario del directorio `addison`. Luego `addison` tiene permisos de lectura y escritura para el directorio `editoriales`. Además como s6=-, significa que `addison` no tiene permiso para ejecutar (entrar en) ese directorio y el bit S_ISUID no está activado.

Los siguientes tres caracteres, (s5s4s3)=(r--) representan los permisos para los miembros del grupo `users`. Como sólo aparece una r cualquier usuario que pertenezca al grupo `users` puede leer este fichero, pero no escribir en él o ejecutarlo y además el bit `S_ISGID` no está activado.

Los últimos tres caracteres, también (s2s1s0)=(---), representan los permisos para cualquier otro usuario del sistema (diferentes del propietario o de los pertenecientes al grupo `users`). En este caso los demás usuarios no pueden leer ni escribir en el directorio (consultar/modificar las entradas del mismo), ejecutarlo (entar) y además el bit S_ISVTX no está activado.

APARTADO c)

La máscara en modo octal del fichero es `0640`.
El 0 inicial procede del hecho de que los bits `S_ISUID, S_ISGID y S_ISVTX` están desactivados, el resto de cifras se obtienen mediante el procedimiento mostrado en la Tabla.

5. *(3 p)* Conteste razonadamente a los siguientes apartados:

a) (2 p) Explicar el significado de las sentencias enumeradas (**[ ]**) del programa que se muestra en la página siguiente.

b) *(1 p)* Explicar el funcionamiento del programa si se compila en el ejecutable `programa`, se invoca con la orden `./programa DyASO` y no se produce ningún error.

La pregunta continua en la página siguiente

```
        #include <stdio.h>
        #include <stdlib.h>
        #include <sys/types.h>
        #include <sys/stat.h>
        #include <fcntl.h>
        #include <unistd.h>
        #include <string.h>
        #include <sys/wait.h>

        void manejador(int sig)
        {
[1]        execl("/bin/cat","cat","texto.txt",NULL);
           perror("Error execl");
           printf("¡Incorecto!\n");
           exit(1);
        }

        int main(int argc, char *argv[])
        {
           int fd, pid, salida, status;
           if (argc!=2) {printf("Argumentos incorrectos\n"); exit(2);}

[2]        if ( (pid=fork())==-1 ) {printf("Error fork\n"); exit(3);}
           if ( pid==0 )
               {
[3]            if (signal(SIGUSR1,manejador)==SIG_ERR)
[4]                {perror("Error signal"); exit(4);}
[5]            pause();
               printf("¡Erroneo!\n");
               }
           else
                {
                sleep(1);
[6]             fd=open("texto.txt", O_RDWR|O_CREAT|O_TRUNC, 0666);
                if (fd==-1) {perror("Error open"); exit(5);}
                write(fd,"Ejercicio 5 ",12);
[7]             write(fd,argv[1],strlen(argv[1]));
[8]             close(fd);
[9]             kill(pid,SIGUSR1);
[10]            wait(&status);
                printf(" ¡Correcto!\n");
[11]            unlink("texto.txt");
                }
           exit(0);
        }
```

RESPUESTA

APARTADO A):

[1] `execl` es una llamada al sistema que carga en la memoria el programa ejecutable que se le indica en el primer parámetro con las opciones que se le pasan a continuación. En este caso se ejecutará el programa `cat` del directorio `bin` (`/bin/cat`) al que se le pasan como argumentos su propio nombre (convenio de UNIX) así como el argumento `texto.txt`. `NULL` indica que ya no hay más argumentos. Si la llamada al sistema tiene éxito, el contexto y la memoria del proceso que la invoca es sustituida por el nuevo ejecutable `cat`.

[2] La llamada al Sistema `fork()` crea un proceso hijo que es una copia idéntica del proceso padre. Al proceso padre le devuelve el `pid` del hijo mientras que al proceso hijo le devuelve 0. En caso de error, `fork` devuelve -1, de tal modo que se ejecuta la cláusula `if`, se imprime "`Error fork`" y se termina el proceso con la llamada al sistema `exit` y el código de retorno 3.

[3] `signal` asocia una función `(manejador)` que actuará como manejador a la señal de usuario número uno (`SIGUSR1`). Si se produce un error devolverá `SIG_ERR` ejecutándose la sentencia **[4]** que monstrará el error que se ha producido.

[4] `perror` imprime "`Error signal:`" seguido del descriptor textual del mensaje de error que se encuentra almacenado en la variable `errno`.

[5] `pause` suspende la ejecución del proceso que la invoca hasta que se atienda una señal que el proceso no ignore.

[6] `open` abre el fichero `"texto.txt"` con permiso de lectura y escritura (`O_RDWR`) truncándolo de modo que elimina el contenido del archivo (`O_TRUNC`). Si el fichero no existe lo crea (`O_CREAT`) con la máscara de permisos `0666`. Si no hay errores devuelve el descriptor de fichero `fd`, en caso contrario devuelve `-1`.

[7] `write` escribe en el fichero anterior el segundo argumento de entrada del ejecutable `argv[1]` (recordemos que el primer argumento `argv[0]` es el nombre del ejecutable). El número de bytes a escribir es la longitud de dicho argumento que se obtiene con `strlen`.

[8] La llamada al sistema `close` cierra el fichero cuyo descriptor es `fd`.

[9] `kill` es una llamada al Sistema que permite enviar una señal (`SIGUSR1`) al proceso hijo cuyo identificador de proceso es `pid`.

[10] `wait` hace que el proceso padre se suspenda hasta la terminación de su hijo. Obteniendo en la variable `status` el código de retorno del mismo.

[11] `unlink` es una llamada al sistema que elimina el fichero `texto.txt`.

APARTADO B:

El funcionamiento del programa es el siguiente:

En primer lugar, se define una función `manejador` que recibe una señal y al hacerlo ejecuta el proceso `cat` con el argumento `texto.txt`. Esto significa que el proceso que la invoque se convertirá en el ejecutable que se obtiene al hacer "`cat texto.txt`". Dicho proceso muestra el contenido del fichero texto.txt por la salida estándar.

La función principal en primer lugar comprueba que recibe dos argumentos de entrada (el nombre de la función y el argumento adicional `DyASO`), puesto que así es, no se produce ningún error. A continuación se realiza un fork que bifurca el proceso original en un proceso padre y un proceso hijo:

-HIJO: El hijo asocia la función manejador con la señal `SIGUSR1` y a continuación queda a la espera de una señal con `pause()`.

-PADRE: El padre en primer lugar duerme (`sleep`) durante un segundo, seguidamente abre el fichero `texto.txt`. Si no se produce ningún error, tal como indica el enunciado, escribe en dicho fichero el texto "`Ejercicio 5 `" y, a continuación el segundo argumento de entrada que es `DyASO`. Hecho esto el padre cierra el fichero cuyo contenido final es "`Ejercicio 5 DyASO`".

Posteriormente, el padre envía una señal `SIGUSR1` a su hijo y queda a la espera de la terminación del mismo con `wait`.

-HIJO: El proceso hijo recibe la señal `SIGUSR1` y ejecuta en su contexto la función `manejador`. Al hacerlo el contexto del hijo queda sustituido por el programa `cat texto.txt` que muestra "`Ejercicio 5 DyASO`" por la salida estándar y termina. Nótese que la instrucción `printf("¡Erroneo!\n")` nunca se ejecuta.

-PADRE: Cuando el hijo termina, el padre escribe por salida estándar "`¡Correcto!`" seguido de un salto de línea y termina. De este modo la salida completa del programa es:

```
Ejercicio 5 DyASO ¡Correcto!
```

NOTA: Sea cual sea el orden en el que el SO planifique los procesos padre e hijo, la salida es siempre la misma ya que el hijo quedará suspendido hasta que el padre envíe la señal `SIGUSR1` (`pause`) y el padre no terminará de escribir "¡ Correcto!" hasta que el hijo haya terminado (`wait`).