Material permitido: **Calculadora NO programable.** Tiempo: **2 horas.** N2	**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. Responda a las siguientes cuestiones:
a) *(0.5 p)*. Explique el resultado de ejecutar la orden:

```
$ cat < /proc/1/status.
```

b) *(0.5 p)*. Explique el funcionamiento del comando:

```
$ fsck [-opciones] [sistema...]
```

RESPUESTA:

a) (V. Apartado 2.9) /Proc/PID/status es un fichero especial del sistema de achivos /proc a través del cuál el UNIX expone información sobre el proceso cuyo *pid* se especifica en el path. En este caso el proceso 1 o "init". De este modo cat recibe por entrada estándar el contenido de /proc/1/status y lo muestra por salida estándar de modo que se muestra información acerca del proceso init.

b) (V. Apartado 8.9) `fsck` revisa y repara de forma interactiva las posibles inconsistencias que encuentra en los sistemas de ficheros UNIX. En el caso de que no existan inconsistencias, `fsck` informa sobre el número de ficheros, número de bloques usados y número de bloques libres de que dispone el sistema. Si el sistema presenta inconsistencias, `fsck` proporciona mecanismos para corregirlo.

2. a) *(1.5 p)* Describa qué son las señales, indicando sus fases. b) *(0.5 p)* Indique qué señal se emplea para finalizar un proceso.

RESPUESTA:

a) (V. Apartado 4.4.1) Las señales proporcionan un mecanismo para notificar a los procesos los eventos que se producen en el sistema. Los eventos se identifican mediante números enteros, aunque también tiene asignadas constantes simbólicas que facilitan su identificación al programador.

Algunos de estos eventos son notificaciones asíncronas (por ejemplo, cuando un usuario envía una señal de interrupción a un proceso pulsando simultáneamente las teclas `[control+c]` en el terminal), mientras que otros son errores síncronos o excepciones (por ejemplo, acceder a una dirección ilegal).

Las señales también se pueden utilizar como un mecanismo de comunicación y sincronización entre procesos.

En el mecanismo de señalización se distinguen dos fases principalmente: *generación* y *recepción* o *tratamiento*. Una señal es generada cuando ocurre un evento que debe ser notificado a un proceso.

La señal es recibida o tratada, cuando el proceso para el cual fue enviada la señal reconoce su llegada y toma las acciones apropiadas. Asimismo, se dice que una señal está *pendiente* para el proceso si ha sido generada pero no ha sido tratada todavía.

b) (V. Apartado 4.4. Tabla 4.1) Hay dos respuestas válidas: `SIGKILL o SIGTERM,` la diferencia entre ellas es que la primera no puede ser capturada.

3. *(2 p)* Describa las principales acciones que realiza el núcleo durante la ejecución del algoritmo **`sleep().`** ¿Qué parámetros de entrada requiere? ¿Qué ocurre cuando el proceso puede ser interrumpido por señales?

RESPUESTA:

(V. Apartado 4.5.1). El núcleo usa el algoritmo `sleep()` para pasar a un proceso A al estado dormido. Este algoritmo requiere como parámetros de entrada la *prioridad para dormir* y la *dirección de dormir o canal* asociada al evento por el que estará esperando el proceso.

La primera acción que realiza `sleep()` es salvar el nivel de prioridad de interrupción (*npi*) actual, típicamente en el registro de estado del procesador. A continuación eleva el *npi* para bloquear todas las interrupciones.

Posteriormente en los campos correspondientes de la entrada de la tabla de procesos asociada al proceso A marca el estado del proceso a *dormido en memoria principal*, salva el valor de la *prioridad para dormir* y de la *dirección de dormir*. Asimismo coloca al proceso en una lista de procesos dormidos.

A continuación compara la *prioridad para dormir* con un cierto valor umbral para averiguar si el proceso puede ser interrumpido por señales. Si la prioridad para dormir es mayor que dicho valor umbral entonces el proceso no puede ser interrumpido por señales. En caso contrario, el proceso sí puede ser interrumpido por señales. Se distinguen por tanto dos casos:

- Caso 1: *El proceso no puede ser interrumpido por señales*. En este caso el núcleo realiza un cambio de contexto, en consecuencia otro proceso B pasará a ser ejecutado. De esta forma la ejecución del algoritmo `sleep()` es momentáneamente detenida. Más tarde, cuando el proceso A sea despertado y planificado para ejecución, continuará su ejecución en modo núcleo en la siguiente instrucción del algoritmo `sleep()`, que consiste en restaurar el valor del *npi* al valor que tenía antes de comenzar a ejecutar el algoritmo. A continuación, el algoritmo finaliza.
- Caso 2: *El proceso puede ser interrumpido por señales*. En este caso el núcleo invoca al algoritmo `issig()` para comprobar la existencia de señales pendientes. Se pueden dar dos casos:
  - Caso 2.1: *Existen señales pendientes*. Entonces el núcleo borra al proceso A de la lista de procesos dormidos, restaura el valor del *npi* (al valor que tenía antes de comenzar a ejecutar el algoritmo) e invoca al algoritmo `psig()` para tratar la señal.
  - Caso 2.2: *No existen señales pendientes*. Entonces el núcleo realiza un cambio de contexto, en consecuencia otro proceso D pasará a ser ejecutado. De esta forma la ejecución del algoritmo `sleep()` es momentáneamente detenida. Más tarde, cuando el proceso A sea despertado (bien porque se produjo el evento por el que estaba esperando o porque es interrumpido por una señal) y planificado para ejecución, el núcleo invocará nuevamente al algoritmo `issig()` para comprobar la existencia de señales pendientes que han podido ser notificadas durante el tiempo que pasó dormido. Existen dos posibilidades:
    - Si no existen señales pendientes, entonces el núcleo restaura el *npi* al valor que tenía antes de comenzar a ejecutar `sleep()` y finaliza el algoritmo.
    - Existen señales pendientes, entonces el núcleo restaura el *npi* al valor que tenía antes de comenzar a ejecutar el algoritmo `sleep()` e invoca a `psig()` para tratar la señal.

4. *(2 p)* Escriba un programa en C que envíe el mensaje “hola” desde un proceso padre a su hijo y a través de una tubería sin nombre.

RESPUESTA:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
main()

{
	int tuberia[2];
	int pid;
	char mensaje[10];
	pipe(tuberia);
	if ((pid=fork())==-1)
	{
		perror("fork");
		exit(-1);
	}
	else if (pid==0) //proceso hijo
	{
		close(tuberia[1]);
		read(tuberia[0], mensaje, 10);
		printf("\nMensaje recibido: %s\n\n", mensaje);
		close(tuberia[0]);
	}

	else //proceso padre
	{
		write(tuberia[1],"hola",5);
		wait();
	}
}
```

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

a) *(1.5 p)* Explique el significado de las sentencias enumeradas (**[ ]**) del programa mostrado en la página siguiente.

b) *(1.5 p)* El programa es compilado por **root** en el fichero ejecutable `proc` y a continuación establece la máscara de modo de `proc` a 4711. Explique la ejecución del programa y muestre su salida si es ejecutado por el usuario sistemas con `uid`=1000 y `gid`=1000.

Nota: Recordamos que el fichero shadow tiene permiso para el usuario root pero no para el resto de usuarios.

La pregunta continua en la página siguiente

```
        #include <stdio.h>
        #include <stdlib.h>
        #include <unistd.h>
        #include <sys/types.h>

        int main(int argc,char *argv[])
        {
        uid_t uid, euid, gid, egid;
[1]     uid=getuid();
[2]     euid=geteuid();
[3]     gid=getgid();
[4]     egid=getegid();

[5]     if (euid==0 || egid==0) {
             printf("Tengo acceso al fichero shadow");
             printf(" (uid=%d euid=%d gid=%d egid=%d)\n",uid,euid,gid,egid);
             printf("Y voy a demostrarlo:\n");
[6]          if (fork()==0) {
[7]               if (execl("/bin/cat","cat","/etc/shadow", NULL)!=0){
                       perror("Error en la lectura:");
                  }
             }
             else {
[8]              seteuid(uid);
                 setegid(gid);
                 euid=geteuid();
                 egid=getegid();
[9]              wait();
                 printf("¿Y el usuario inicial?");
             printf(" (uid=%d euid=%d gid=%d egid=%d):\n",uid,euid,gid,egid);
                  if (execl("/bin/cat","cat","/etc/shadow", NULL)!=0) {
                       perror("Error en la lectura:");
                  }
             }
        }
        else {
        printf("No tengo acceso al archivo.\n");
        }
        }
```

RESPUESTA

Apartado a)

[1], **[2]**, **[3]** y **[4]** las llamadas al sistema `getuid`, `geteuid`, `getgid`, y `getegid` se encargan de obtener el identificador numérico de usuario `uid`, el identificador de usuario efectivo `euid`, el identificador de grupo `gid` y el identificador de grupo efectivo `egid` respectivamente. Cuando se accede a un recurso o se hacen llamadas al sistema que requieren privilegios UNIX comprueba los valores de `euid` y `egid` para asignar o denegar el acceso al recurso solicitado.

Cuando un proceso nuevo es creado mediante `fork()` los valores de identificador de usuario se heredan del proceso padre que los ha creado. Al invocar `exec` (o cualquiera de sus variantes) dichos valores se conservan a no ser que estén activados los bits `S_ISUID` o `S_ISGID`, en cuyo caso el usuario o grupo efectivo pasan a ser los del propietario del fichero.

[5] En esta sentencia, la condición del `if` se cumple si el usuario o el grupo efectivo es cero, recordemos que 0 es el identificador numérico del usuario `root`. De este modo la sentencia comprueba si se tienen privilegios de superusuario, en caso afirmativo se ejecuta el bloque de código siguiente al `if`, si no se ejecuta el último bloque de código que sigue al `else` que mostraría el mensaje "`No tengo acceso al archivo.`"

[6] `fork()` es la llamada al sistema que se utiliza para crear un nuevo proceso hijo duplicando el código del proceso padre. `fork()` devuelve el `pid` del proceso hijo al padre y 0 al proceso hijo, de este modo el hijo ejecutará el bloque de código a continuación del `if [6]` mientras que el padre ejecutará el código correspondiente a la cláusula `else` siguiente.

[7] `execl("/bin/cat","cat","/etc/shadow", NULL)` es una llamada al sistema que se encarga de ejecutar un fichero ejecutable. La memoria del proceso pasará a ser substituida por el código del ejecutable que se invoca. En este caso se ejecutará el programa `cat` que se encuentra en el directorio `bin`, Los parámetros que se le pasan al ejecutable como entrada son su propio nombre (la convención en UNIX es que el primer argumento sea el nombre del fichero ejecutable), y el parámetro "`/etc/shadow`". El parámeto NULL sirve para indicar a `execl` que no hay más parámetros de entrada. El resultado es similar a la invocación por línea de comandos de `$ /bin/cat /etc/shadow`, comando que intentará mostrar en pantalla el fichero de contraseñas del sistema. Si la llamada al sistema fallase devolvería -1 y se mostraría el mensaje de error con `perror`.

[8] La llamada al sistema `seteuid(uid)` intenta restablecer el usuario efectivo del fichero al identificador de usuario.

[9] `wait()` es una llamada al sistema que sincroniza la ejecución del proceso padre y del proceso hijo. Cuando la invoca, el proceso padre espera a que termine el proceso hijo antes de continuar.

Apartado b)

El programa ejecutable tiene como permisos `4711`, esto es lectura escritura y ejecución para su propietario (recordemos que el propietario es root puesto que él ha compilado el programa), y permisos de ejecución para los usuarios del grupo root y para el resto de usuarios. Además el archivo ejecutable se encuentra setuidado, es decir su bit `S_ISUID` está activo.

Esto significa que al ejecutarse el fichero por parte del usuario sistemas (que tiene permiso de ejecución) los identificadores de usuario serán `uid=1000` , `euid=0`, `gid=1000`, `egid=1000`. Nótese que el bit `S_ISUID` hace que al ejecutase el usuario efectivo sea root (el propietario del fichero) y no sistemas.

A continuación el programa obtiene estos valores con las instrucciones [1]-[4] y la condición del `if` se cumple (puesto que `euid=0`) de modo que se imprime por pantalla:

```
"Tengo acceso al fichero shadow (uid=1000 euid=0 gid=1000 egid=1000)"
Y voy a demostrarlo:
```

El proceso a continuación se bifurca dando lugar a un proceso padre y un proceso hijo.

El proceso padre restaura sus permisos efectivos a los del usuario sistemas mediante seteuid y setgid, de modo que el padre tendrá `uid=1000`, `euid=1000`, `gid=1000` y `egid=1000`. A continuación y vuelve a leerlos para refrescar el valor de las variables `euid` y `egid` y se queda esperando a que termine el proceso hijo mediante la llamada al sistema `wait()`.

El proceso hijo, por su parte, realiza un `execv` que substituye el proceso existente por el resultado de ejecutar `/bin/cat /etc/shadow`, puesto que el proceso tiene euid=0 sus permisos de acceso son los de root lo que le permite leer el fichero `/etc/shadow` sin problemas y mostrar su contenido por la pantalla, que será algo similar a esto:

```
root:!:15644:0:99999:7:::
…
sistemas:$6$eiuRucoi$HejAwO5TWlg2v7zjX5kkKLcsW9QLI4iXcpKu9IswI7/ALxm2BN4
XG34pz5o1A0xmPgp/BZ8jZ/a1Kn3iEpKXA/:15644:0:99999:7:::
…
```

A continuación el proceso hijo termina y el padre se despierta. El padre imprime por pantalla:

```
¿Y el usuario inicial? (uid=1000 euid=1000 gid=1000 egid=1000):
```

Y a continuación, cuando intenta leer el fichero con un `execv` similar al proceso hijo, se produce un error, puesto al comprobar los permisos de acceso de shadow el proceso `cat` que se ejecuta con `euid=1000` (sistemas) no tiene permisos de lectura sobre el mismo, de modo que se imprime algo como esto:

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

De modo que la salida completa del programa sería:

```
Tengo acceso al fichero shadow (uid=1000 euid=0 gid=1000 egid=1000)
Y voy a demostrarlo:
root:!:15644:0:99999:7:::
sistemas:$6$eiuRucoi$HejAwO5TWlg2v7zjX5kkKLcsW9QLI4iXcpKu9IswI7/ALxm2BN4
XG34pz5o1A0xmPgp/BZ8jZ/a1Kn3iEpKXA/:15644:0:99999:7:::
otro:$6$Ru1lg$EsKR30zwZMlhVswvkhqRDsUX4WhbQvCulZV1Yt2qZulIFSaCwZeKfKalb6
btySW5wma8.T0fHBxgfjYNue7cL/:15644:0:99999:7:::
```

... Resto del contenido del fichero shadow...

```
¿Y el usuario inicial? (uid=1000 euid=1000 gid=1000 egid=1000):
cat: /etc/shadow: Permiso denegado
```