1.- Escribir un programa que haga las funciones de la orden **umask** del shell.

- Si se usa sin argumentos, muestra el valor de la máscara de usuario.
- Con un parámetro, cambia la máscara.

Nota. Se debe crear, dentro del programa, un nuevo fichero. Observar después que el nuevo fichero toma los permisos adecuados.

2.- Escribir un programa para "mover" ficheros. Usar dos argumentos, uno para indicar el fichero origen y otro para indicar el nuevo destino.
Nota.- Usar las llamadas link y unlink.

3.- Escribir un programa, llamado cambiar, que sea funcionalmente equivalente al comando del shell **chmod.**
Su uso sería:

cambiar numero fichero

donde *numero* sea un numero octal que representa los permisos.

4.- Escribir un programa para hacer una o más copias de un mismo fichero.
Su uso sería:

copiar fichero1 fichero2 [fichero3, ...]

5.- Escribir un programa que acepte como entrada un fichero regular y devuelva su tamaño en bytes.
Nota.- Usar las llamadas open, lseek y close, no stat.

6.- Escribir un programa en el que se cree un nuevo fichero y se escriba en una posición no inicial (lseek con un desplazamiento cualquiera). Comprobar después el tamaño del fichero creado.

7.- Implementar un programa para borrar uno o más ficheros que se pasen como argumentos.
Su uso sería:

borrar fichero1 fichero2 [fichero3, ...]

8.- Escribir un programa, llamado type, que permita mostrar el contenido de un fichero regular que se pase como argumento.

9.- Escribir un programa que reciba a través de la línea de órdenes el número de procesos hijo que debe crear. Cada proceso hijo deberá dormir un número aleatorio de segundos comprendido entre 0 y 30. Conforme vayan terminando los hijos, el padre presentará en pantalla el PID de cada uno de ellos, así como la cantidad de segundos que ha estado durmiendo. Este último dato se generará internamente dentro de cada proceso hijo, es decir, el padre no lo sabe hasta que el hijo no se lo comunica.
Nota. El hijo y el padre deben escribirse como programas diferentes.

10.- Escribir un programa en el que un proceso padre y un hijo utilicen un pipe para comunicarse mensajes. Usar una variable mensaje de longitud máxima declarada para enviar (padre) y recibir (hijo) los mensajes. La comunicación debe terminar cuando el padre envíe el mensaje "FIN".
Nota. El proceso hijo no tiene porqué saber que la cadena FIN termina la comunicación. El hijo terminará cuando no haya nada más que leer del pipe.

11.- Escribir un programa en C usando llamadas al sistema UNIX que permita copiar un fichero en otro a través de un *pipe*.
Se llamará con la siguiente sintaxis:

copiar fuente destino

Deberá resolverse en base a la estructura siguiente. Habrá dos programas diferentes, el programa principal (el padre) recibe como argumentos los nombres de los ficheros fuente y destino. El será el encargado de abrir el fichero fuente, leer su contenido y volcarlo en el *pipe*. El programa secundario leerá del *pipe* y escribirá en el fichero destino. El nombre de éste lo recibirá del padre como un argumento en la llamada de ejecución

Programa principal (parámetros: fichero fuente y fichero destino)
.....
Generar un nuevo proceso

Proceso padre:	Proceso Hijo:
Leer del fichero fuente	Llamar al programa secundario
Escribir en el *pipe*	(pasar el nombre del destino como argumento)

12.- Escribir un programa en C usando llamadas al sistema UNIX que permita simular la redirección de salida de una orden hacia un fichero.
Se llamará con la siguiente sintaxis:

redirigir orden fichero

El funcionamiento del programa será el siguiente. Se ejecutará la orden y su resultado se almacenará en el fichero. Por simplicidad, suponer que la orden no incluye parámetros. Se deberán controlar los errores (no existe la orden, no se puede crear el fichero, etc.)

13.- Escribir un programa en C usando llamadas al sistema UNIX que permita simular el mecanismo de tubería (pipe) entre dos órdenes.
Se llamará con la siguiente sintaxis:

entubar orden1 orden2

El funcionamiento del programa será el siguiente. Se ejecutará la orden1 y su salida resultante será utilizada como entrada estándar por la orden2. Por simplicidad, se supone que ni orden1 ni orden2 incluyen parámetros o argumentos.