1. Con llamadas al sistemas relativas a procesos

1 Programa una aplicación en C en la cual un proceso padre crea exactamente 2 hijos. El hijo 1 debe imprimir en pantalla "Soy el hijo 1 y mi PID es xxx" y el hijo 2 algo parecido "Soy el hijo 2 y mi PID es xxxx". El número de cada proceso se puede averiguar invocando `getpid`. El padre debe imprimir "Soy el padre y he creado dos hijos, cuyos PID son xxxx y xxxx".

2 Programa una aplicación en C con 2 procesos. El proceso padre le pasa dos sumandos al proceso hijo, el hijo realiza la suma y le pasa el resultado al padre a través de la llamada `exit`. El padre debe esperar a que el hijo finalice (`wait`) e imprimir el resultado en consola.

3 Programa una aplicación en C que usando llamadas al sistema imprima en pantalla los ficheros que hay en el directorio desde el que lo ejecutas. Recuerda que ya existe un programa en el sistema hace justo eso.

2. Con llamadas al sistemas relativas a ficheros

1 Programa una aplicación en C en la cual un proceso escriba por pantalla "Hola mundo" sin utilizar `printf`, empleando la llamada al sistema `write`.

2 Programa una aplicación en C que copia el contenido del fichero *origen* en un nuevo fichero *destino*. Crea con tu editor favorito el fichero origen y rellenalo con varios caracteres. Prueba tu programa copiador con ficheros de origen grandes como canciones o videos.

3 Escribe un programa que vuelque en pantalla el contenido de `argc` y `argv` cuando lo invocas con un parámetro en la línea de comandos, o dos, o tres... Modifica el programa del ejercicio anterior pero adáptalo para que los nombres de los ficheros origen y destino se pasan como argumentos del ejecutable

4 Programa una aplicación en C que usando llamadas al sistema imprima en pantalla tres veces la frase que se escribe por entrada estandard. Hazlo sin utilizar `scanf`, sino empleando la llamada al sistema `read`. Si quieres asume que la frase ocupa como máximo 30 bytes.

3. Con llamadas al sistemas relativas a redirecciones y manejo de tuberías

1 Modifica el programa que copia un fichero *origen* en un nuevo fichero *destino* para se comporte como `cat` cuando se le invoca sin argumentos, usalo para que se pueda redireccionar su salida y entrada estandard desde la shell.

2 Programa una aplicación que copie un fichero de origen en un fichero de destino, y que coja los nombres de destino y origen desde la consola de comandos.

3 Programa una aplicación que copie un fichero de origen en un fichero de destino, y que coja los nombres de destino y origen desde el teclado una vez que el programa ha arrancado. Asume para simplificar que los nombres de archivo tienen 7 caracteres y están separados por un espacio en blanco

4 Programa una aplicación que se comporte como `tee`.

5 Programa una aplicación en C con 2 procesos. El proceso padre le pasa dos sumandos al proceso hijo a través de una tubería, el hijo realiza la suma y le pasa el resultado al padre a través de otra tubería. El padre le pasará los operandos de una segunda suma al hijo y este nuevamente entregará el resultado a través de una tubería. El padre debe imprimir los resultados en consola y como siempre, recoger el valor de retorno del hijo, que ahora será +1 en caso de que no haya problema.

4. Un poco más divertidos

1 La empresa MadComputer ofrece servicios de cómputo masivo. Esta empresa recibe sus peticiones de trabajo en un fichero, llamado CmdFile, que tiene una petición por línea. En cada una de ellas se especifica un comando a ejecutar, el nombre del fichero del que dicho comando debe leer los datos y el nombre del fichero en que debe escribir el resultado.

Se pide escribir una aplicación en lenguaje C que, utilizando llamadas al sistema Linux, procese las peticiones de trabajo, de tal forma que para cada petición el comando solicitado se ejecute teniendo como entrada (estandard) y salida (estandard) los ficheros indicados.

Tras la ejecución de cada petición, la aplicación mostrará el mensaje `Ejecución correcta` si el código de finalización fue 0 y `Ejecución incorrecta` si el código de finalización indica errar (distinto de 0).

2 Una empresa que tiene un almacén de mercancías robotizado acaba de contrarnos como ingeniero. Tienen varios robots móviles que transportan los contenedores de un sitio a otro. En cada robot, un programa *navegador* genera las órdenes de movimiento adecuadas, que escribe en su salida estándar, y un programa llamado *motores* las recibe por su entrada estándar y las lleva a cabo. Para poder controlar el flujo de órdenes, *motores* las lee de una en una por su entrada estándar y cuando está listo para recibir otra, escribe la palabra "listo" por su salida estándar.

El problema que tiene la empresa es que *navegador* genera 23 órdenes por segundo, y *motores* sólo es capaz de procesar 5 por segundo. Nos piden que escribamos en lenguaje C un programa que utilizando las llamadas al sistema de `GNU/Linux` haga de intermediario entre ambos. Este programa leerá las órdenes del *navegador* y cada vez que *motores* esté listo le enviará la última orden recibida del *navegador*, descartando las anteriores. Cada orden de movimiento ocupa 1 byte y se dispone de los ejecutables de *navegador* y *motores*, pero no de su código fuente.