1.a Buscar en el manual la lista de funciones disponibles para el manejo de hilos y copiarla en la memoria junto con el comando usado para mostrarla. Las funciones de manejo de hilos comienzan por “pthread”.

man. -k pthread

pthread_attr_destroy (3) - initialize and destroy thread attributes object
pthread_attr_getaffinity_np (3) - set/get CPU affinity attribute in thread attributes object
pthread_attr_getdetachstate (3) - set/get detach state attribute in thread attributes object
pthread_attr_getguardsize (3) - set/get guard size attribute in thread attributes object
pthread_attr_getinheritsched (3) - set/get inherit-scheduler attribute in thread attributes object
pthread_attr_getschedparam (3) - set/get scheduling parameter attributes in thread attributes object
pthread_attr_getschedpolicy (3) - set/get scheduling policy attribute in thread attributes object
pthread_attr_getscope (3) - set/get contention scope attribute in thread attributes object
pthread_attr_getstack (3) - set/get stack attributes in thread attributes object
pthread_attr_getstackaddr (3) - set/get stack address attribute in thread attributes object
pthread_attr_getstacksize (3) - set/get stack size attribute in thread attributes object
pthread_attr_init (3) - initialize and destroy thread attributes object
pthread_attr_setaffinity_np (3) - set/get CPU affinity attribute in thread attributes object
pthread_attr_setdetachstate (3) - set/get detach state attribute in thread attributes object
pthread_attr_setguardsize (3) - set/get guard size attribute in thread attributes object
pthread_attr_setinheritsched (3) - set/get inherit-scheduler attribute in thread attributes object
pthread_attr_setschedparam (3) - set/get scheduling parameter attributes in thread attributes object
pthread_attr_setschedpolicy (3) - set/get scheduling policy attribute in thread attributes object
pthread_attr_setscope (3) - set/get contention scope attribute in thread attributes object
pthread_attr_setstack (3) - set/get stack attributes in thread attributes object
pthread_attr_setstackaddr (3) - set/get stack address attribute in thread attributes object
pthread_attr_setstacksize (3) - set/get stack size attribute in thread attributes object
pthread_cancel (3) - send a cancellation request to a thread
pthread_cleanup_pop (3) - push and pop thread cancellation clean-up handlers
pthread_cleanup_pop_restore_np (3) - push and pop thread cancellation clean-up handlers while saving cancelability type
pthread_cleanup_push (3) - push and pop thread cancellation clean-up handlers
pthread_cleanup_push_defer_np (3) - push and pop thread cancellation clean-up handlers while saving cancelability type
pthread_create (3) - create a new thread
pthread_detach (3) - detach a thread
pthread_equal (3) - compare thread IDs
pthread_exit (3) - terminate calling thread
pthread_getaffinity_np (3) - set/get CPU affinity of a thread
pthread_getattr_default_np (3) - get or set default thread-creation attributes
pthread_getattr_np (3) - get attributes of created thread

pthread_getconcurrency (3) - set/get the concurrency level
pthread_getcpuclockid (3) - retrieve ID of a thread's CPU time clock
pthread_getname_np (3) - set/get the name of a thread
pthread_getschedparam (3) - set/get scheduling policy and parameters of a thread
pthread_join (3) - join with a terminated thread
pthread_kill (3) - send a signal to a thread
pthread_kill_other_threads_np (3) - terminate all other threads in process
pthread_mutex_consistent (3) - make a robust mutex consistent
pthread_mutex_consistent_np (3) - make a robust mutex consistent
pthread_mutexattr_getpshared (3) - get/set process-shared mutex attribute
pthread_mutexattr_getrobust (3) - get and set the robustness attribute of a mutex attributes object
pthread_mutexattr_getrobust_np (3) - get and set the robustness attribute of a mutex attributes object
pthread_mutexattr_setpshared (3) - get/set process-shared mutex attribute
pthread_mutexattr_setrobust (3) - get and set the robustness attribute of a mutex attributes object
pthread_mutexattr_setrobust_np (3) - get and set the robustness attribute of a mutex attributes object
pthread_rwlockattr_getkind_np (3) - set/get the read-write lock kind of the thread read-write lock attribute object
pthread_rwlockattr_setkind_np (3) - set/get the read-write lock kind of the thread read-write lock attribute object
pthread_self (3) - obtain ID of the calling thread
pthread_setaffinity_np (3) - set/get CPU affinity of a thread
pthread_setattr_default_np (3) - get or set default thread-creation attributes
pthread_setcancelstate (3) - set cancelability state and type
pthread_setcanceltype (3) - set cancelability state and type
pthread_setconcurrency (3) - set/get the concurrency level
pthread_setname_np (3) - set/get the name of a thread
pthread_setschedparam (3) - set/get scheduling policy and parameters of a thread
pthread_setschedprio (3) - set scheduling priority of a thread
pthread_sigmask (3) - examine and change mask of blocked signals
pthread_sigqueue (3) - queue a signal and data to a thread
pthread_spin_destroy (3) - initialize or destroy a spin lock
pthread_spin_init (3) - initialize or destroy a spin lock
pthread_spin_lock (3) - lock and unlock a spin lock
pthread_spin_trylock (3) - lock and unlock a spin lock
pthread_spin_unlock (3) - lock and unlock a spin lock
pthread_testcancel (3) - request delivery of any pending cancellation request
pthread_timedjoin_np (3) - try to join with a terminated thread
pthread_tryjoin_np (3) - try to join with a terminated thread
pthread_yield (3) - yield the processor
pthreads (7) - POSIX threads

1.b Consultar en la ayuda en qué sección del manual se encuentran las “llamadas al sistema” y buscar información sobre la llamada al sistema write. Escribir en la memoria los comandos usados.

Las llamadas a sistema se encuentran en la sección 2 del manual.

usamos el comando “man write” para buscar información sobre la llamada al sistema write

2.a Escribir un comando que busque las líneas que contengan “molino” en el fichero “don quijote.txt” y las añade al final del fichero “aventuras.txt”. Copiar el comando en la memoria, justificando las opciones utilizadas.

añadimos la flag -w para filtrar el contenido del archivo por palabras, en este caso molino. Utilizamos >> para copiar el contenido al final del fichero, como se especifica en el enunciado.

grep -w molino Downloads/don\ quijote.txt >> aventura.txt

2.b Elaborar un pipeline que cuente el número de ficheros en el directorio actual. Copiar el pipeline en la memoria, justificando los comandos y opciones utilizados.

Primero usamos ls para identificar los ficheros del directorio actual (cada uno de estos cuenta como una línea). Conectamos la tubería con el comando wc para contar el número de palabras y le añadimos la flag -l para que el conteo sea de líneas.

ls | wc -l

2.c Elaborar un pipeline que cuente el número de líneas distintas al concatenar “lista de la compra Pepe.txt” y “lista de la compra Elena.txt” y lo escriba en “num compra.txt”. Si alguno de los ficheros no existe, hay que ignorar los mensajes de error, para lo cual se redirigirá la salida de errores a /dev/null. Copiar el pipeline en la memoria, justificando los comandos y opciones utilizados.

Usamos el comando cat para concatenar los 2 archivos requeridos. El siguiente paso de la tubería es ordenar la concatenación (usando sort) con la flag -d para que sea en orden alfabético. En la siguiente parte de la tubería usamos el comando uniq para que no haya elementos repetidos y por último redirigimos todo esto al nuevo archivo usando >.
cat ruta hasta el archivo/lista\ de\ la\ compra\ Pepe.txt ruta hasta el archivo/lista\ de\ la\ compra\ Elena.txt 2>/dev/null | sort -d | uniq -i > "num compra.txt"

3.a ¿Qué mensaje se imprime al intentar abrir un fichero inexistente? ¿A qué valor de errno corresponde?

Imprime el mensaje “No such file or directory”
Corresponde al siguiente valor: $ errno 2

ENOENT 2 No such file or directory

3.b ¿Qué mensaje se imprime al intentar abrir el fichero /etc/shadow? ¿A qué valor de errno corresponde?

Imprime el mensaje “Permission denied”
Corresponde al siguiente valor: $ errno 13

EACCES 13 Permission denied

3.c Si se desea imprimir el valor de errno antes de la llamada a perror, ¿qué modificaciones se deberían realizar para garantizar que el mensaje de perror se corresponde con el error de fopen?

La modificación que deberemos realizar es guardar errno en una variable auxiliar.

4.a Escribir un programa que realice una espera de 10 segundos usando la función clock en un bucle. Ejecutar en otra terminal el comando top. ¿Qué se observa?

Se observa que al ejecutar el programa el procesador pasa a trabajar más hasta que este termina.

4.b Reescribir el programa usando sleep y volver a ejecutar top. ¿Ha cambiado algo?

El cambio que se puede observar es que al empezar la ejecución antes de entrar a la función sleep sube el uso del cpu pero cuando empieza a esperar vuelve a un uso similar al que había cuando no se estaba ejecutando.

5.a ¿Qué hubiera pasado si el proceso no hubiera esperado a los hilos? Para probarlo basta eliminar las llamadas a pthread_join.

Cuando el proceso no espera a los hilos sólo se imprime el primer carácter de cada palabra.

5.b Con el código modificado del apartado anterior, indicar qué ocurre si se reemplaza la función exit por una llamada a pthread_exit.

Disponemos de dos casos. Si cambiamos todas las funciones por llamadas a pthread_exit volvemos al resultado inicial del archivo antes de las modificaciones del apartado a, con la única diferencia que el mensaje de finalización se imprime antes.

Por otro lado si no cambiamos la última llamada a exit pero si las demás obtendremos el mismo resultado que en el apartado a.

5.c Tras eliminar las llamadas a pthread_join en los apartados anteriores, el programa es ahora incorrecto porque no se espera a que terminen todos los hilos. Escribir en la memoria el código que sería necesario añadir para que sea correcto no esperar a los hilos creados.

El código que necesitamos añadir es “sleep(5)” antes de la última llamada a printf.

6.a Analizar el texto que imprime el programa. ¿Se puede saber a priori en qué orden se imprimirá el texto? ¿Por qué?

A priori no se puede saber porque son ejecuciones concurrentes

6.b Cambiar el código para que el proceso hijo imprima su PID y el de su padre en vez de la variable i. Copiar las modificaciones en la memoria y explicarlas.

```
intmax_t hijo;
intmax_t ppid; /*Declaramos dos variables para obtener el pid y el ppid*/
…
        else if (pid = 0) {
                hijo = (intmax_t) getpid();
                ppid = (intmax_t) getppid();
                printf(“Soy el hijo con pid: %jd\n”, hijo);
                printf(“y el pid del padre es: %jd\n”, ppid);
                exit(EXIT_SUCCESS);
        }
```

Le asignamos el valor del pid y el ppid a las variables creadas y posteriormente imprimimos esos valores usando printf.

6.c ¿A cuál de los dos árboles de procesos se corresponde el código de arriba, y por qué? ¿Qué modificaciones habría que hacer en el código para obtener el otro árbol de procesos?

EL primero porque los fork los realiza el proceso principal. Para que fuese como el segundo habría que quitar el exit de la condición de pid = 0 y añadirla después de un wait en el padre.

6.d El código original deja procesos huérfanos, ¿por qué?

Esto se debe a que solo se realiza una espera, por lo que el proceso padre solo espera al hijo que primero responde. El resto de procesos hijos se quedan huérfanos.

6.e Introducir el mínimo número de cambios en el código para que no deje procesos huérfanos. Copiar las modificaciones en la memoria y explicarlas.

Debemos añadir la línea “wait(NULL);” dentro de la condición del proceso padre para que así espere a todos los procesos hijo.

7.a En el programa anterior se reserva memoria en el proceso padre y se inicializa en el proceso hijo usando la función strcpy (que copia un string a una posición de memoria). Una vez el proceso hijo termina, el padre lo imprime por pantalla. ¿Qué ocurre cuando se ejecuta el código? ¿Es este programa correcto? ¿Por qué?

Cuando se ejecuta el código da error porque el padre y el hijo no comparten memoria, por lo que no se realiza correctamente el strcpy.

7.b El programa anterior contiene una fuga de memoria ya que el array sentence nunca se libera. Corregir el código para eliminar esta fuga y copiar las modificaciones en la memoria. ¿Dónde hay que liberar la memoria, en el proceso padre, en el hijo o en ambos? ¿Por qué?

Debemos liberar memoria en ambos porque al no compartir memoria si liberas en uno el otro todavía faltaría por eliminarse.

8.a ¿Qué sucede si se sustituye el primer elemento del array argv por la cadena "mi-ls"? ¿Por qué?

No ocurre ningún cambio porque el primer parámetro se toma como nombre con lo cual se ignora.

8.b Indicar las modificaciones que habría que hacer en el programa anterior para utilizar la función execl en lugar de execvp.

Las modificaciones serían las siguientes:

```
char *argv[2] = {"mi-ls", "./"};
execl("/usr/bin/ls", argv[0], argv[1], (char * )NULL)
```

9. Buscar para alguno de los procesos la siguiente información en el directorio /proc y escribir tanto la información como el fichero utilizado en la memoria.

9.a El nombre del ejecutable

./usr/libexec/evolution-data-server/evolution-alarm-notify

9.b El directorio actual del proceso.

proc/2189

9.c La l´ınea de comandos que se us´o para lanzarlo.

cat cmdline | tr '\0' '\n'
/usr/libexec/evolution-data-server/evolution-alarm-notify

9.d El valor de la variable de entorno LANG.

cat environ | tr '\0' '\n'| grep LANG
LANG=en_US.UTF-8

9.e La lista de hilos del proceso.

PID	SPID	TTY	TIME	CMD
2189	2189	?	00:00:00	evolution-alarm
2189	2287	?	00:00:00	gmain
2189	2288	?	00:00:00	dconf worker
2189	2289	?	00:00:00	evolution-alarm
2189	2290	?	00:00:00	gdbus
2189	2354	?	00:00:00	evolution-alarm

10.a Stop 1. Inspeccionar los descriptores de fichero del proceso. ¿Qué descriptores de fichero se encuentran abiertos? ¿A qué tipo de fichero apuntan?

lrwx------ 1 bernardo bernardo 64 mar 4 14:32 0 -> /dev/pts/0
lrwx------ 1 bernardo bernardo 64 mar 4 14:32 1 -> /dev/pts/0
lrwx------ 1 bernardo bernardo 64 mar 4 14:32 2 -> /dev/pts/0

10.b Stop 2 y Stop 3. ¿Qué cambios se han producido en la tabla de descriptores de fichero?

Ahora los descriptores de fichero apuntan al fichero creado file1.txt
lrwx------ 1 bernardo bernardo 64 mar 4 14:34 3 -> /home/bernardo/soper/p1/file1.txt

10.c Stop 4. ¿Se ha borrado de disco el fichero FILE1? ¿Por qué? ¿Se sigue pudiendo acceder al fichero a través del directorio /proc? ¿Hay, por tanto, alguna forma sencilla de recuperar los datos?

File1 no se ha borrado porque unlink quita la ruta hasta el fichero y hasta que no se borren todas las rutas y se cierren los procesos que lo tuvieron abierto no se borrara de disco. Se pueden recuperar los datos de forma sencilla reenlazando el archivo.

10.d Stop 5, Stop 6 y Stop 7. ¿Qué cambios se han producido en la tabla de descriptores de fichero? ¿Que se puede deducir sobre la numeración de un descriptor de fichero obtenido tras una llamada a open?

En Stop 5 se ha hecho el borrado definitivo de file1.txt porque se ha cerrado el proceso. En Stop 6 se ha creado file3.txt. En Stop7 se ha creado otro proceso que lee del archivo file3.txt.

11.a ¿Cuántas veces se escribe el mensaje “Yo soy tu padre” por pantalla? ¿Por qué?

2. Porque al introducirse en buffer se leen más caracteres de los que se pueden.

11.b En el programa falta el terminador de línea (\n) en los mensajes. Corregir este problema. ¿Sigue ocurriendo lo mismo? ¿Por qué?

No porque al obligar que se imprima en la terminal con el \n hacemos que se vacíe el buffer.

11.c Ejecutar el programa redirigiendo la salida a un fichero. ¿Qué ocurre ahora? ¿Por qué?

Vuelve a pasar lo del apartado a porque se vuelve a intentar minimizar las llamadas a sistema ya que no tiene que mostrar por pantalla cada una de las impresiones si no que lo hace solo en 1 llamada.

11.d Indicar en la memoria cómo se puede corregir definitivamente este problema sin dejar de usar printf.

Llamando a fflush con argumento null despues de llamar a printf para que se vacíen todos los streams de salida.

12.a Ejecutar el código. ¿Qué se imprime por pantalla?

He escrito en el pipe
He recibido el string: Hola a todos!

12.b ¿Qué ocurre si el proceso padre no cierra el extremo de escritura? ¿Por qué?

El programa no finaliza nunca porque se intenta que con una sola tubería haya comunicación bidireccional siendo esto imposible además que nunca se leería un EOF.

13.c

Se usa el execvp porque permite usar tanto el comando de forma normal como pasar la ruta al ejecutable, además que como no se sabe el número exacto de argumentos que se le pasarán. Se pasan como un array de argumentos en el que el argumento que marca que ya no quedan más es un puntero NULL.

sh -c inexistente Exited with value: 127

abort Terminated by signal: 6