Presentación

Esta práctica plantea una serie de actividades con el objetivo que el estudiante aplique sobre un sistema Unix algunos de los conceptos introducidos en los primeros módulos de la asignatura.

El estudiante tendrá que realizar una serie de ejercicios y responder justificadamente las preguntas planteadas. También tendrá que escribir unos pequeños programas en lenguaje C.

La práctica se puede desarrollar sobre cualquier sistema Unix (en el campus se facilita la distribución de Ubuntu 14.04). Se aconseja que mientras realizáis los ejercicios no haya otros usuarios trabajando en el sistema porque el resultado de la ejecución de algunos programas puede depender de la carga del sistema.

Cada ejercicio tiene indicado en el enunciado su peso en porcentaje respecto a la puntuación total de la práctica.

Competencias

Transversales:

- Capacidad para adaptarse a las tecnologías y a futuros entornos actualizando las competencias profesionales.

Específicas:

- Capacidad para analizar un problema en el nivel de abstracción adecuado a cada situación, y aplicar las habilidades y conocimientos adquiridos para abordarlo y resolverlo.

- Capacidad para diseñar y construir aplicaciones informáticas mediante técnicas de desarrollo, integración y reutilización.

Enunciado

Para realizar esta práctica os facilitamos el fichero pr1so.zip que contiene los ficheros necesarios para la realización de los ejercicios. Descomprimid el fichero con el comando **unzip**.

Para compilar un programa, por ejemplo prog.c, debéis ejecutar el comando **gcc** con la opción **-o**. Ejemplo: **$ gcc prog.c -o prog**. Como prueba, podéis abrir un editor de texto, copiar las líneas del recuadro en el editor y guardar el fichero con el nombre hello_world.c. A continuación, compilad el programa con el comando **gcc** y lo ejecutáis como sigue: **$ ./hello_world**

```
#include <stdio.h>
int main() {
  printf("Hello World!\n");
  return 0;
}
```

Editar y ejecutar hello_world.c:
Compilamos: **$ gcc hello_world.c -o hello_world**
Ejecutamos: **$ ./hello_world**

El resultado de la ejecución del programa es el siguiente:

```
montse@montse-VirtualBox:~/pr1so$ gcc hello_world.c -o hello_world
montse@montse-VirtualBox:~/pr1so$ ./hello_world
Hello World!
montse@montse-VirtualBox:~/pr1so$
```

Ejercicio 1 - Módulos 1 y 2 (Puntuación: 30% = 10% + 10% + 10%)

El objetivo de este primer ejercicio es familiarizarnos con el entorno de trabajo de la máquina virtual y trabajar con algunos de los comandos internos que tiene el sistema.

Apartado 1.1

Vamos a hacer un ejercicio de procesos. Para ello utilizaremos el comando **top**. El comando top muestra información del sistema en tiempo real. También muestra la lista de procesos que está gestionando el sistema y cuáles son los que se están ejecutando.

Paso 1: Ejecutad el comando top y poned una captura de pantalla del resultado de la ejecución.

Solución:
Resultado de la ejecución del comando $ top:

```
top - 23:17:29 up 3 min,  2 users,  load average: 3,50, 2,30, 0,96
Tareas: 180 total,   1 ejecutar,  179 hibernar,    0 detener,    0 zombie
%Cpu(s):  5,1 usuario, 10,1 sist,  0,0 adecuado, 84,8 inact,  0,0 en espera,  0,
KiB Mem:   2064212 total,   972768 used,  1091444 free,    62316 buffers
KiB Swap:  2095100 total,        0 used,  2095100 free.   437240 cached Mem

  PID USUARIO   PR  NI    VIRT    RES    SHR S  %CPU %MEM     HORA+ ORDEN
 1990 montse    20   0  433432 150708  36532 S  36,7  7,3   0:51.63 compiz
 1098 root      20   0  136896  36084   9632 S   8,6  1,7   0:10.40 Xorg
 2069 montse    20   0  215352  27496  19112 S   1,3  1,3   0:02.59 nautilus
 2668 montse    20   0  140964  19104  12736 S   1,0  0,9   0:00.93 gnome-term+
   61 root      20   0       0      0      0 S   0,3  0,0   0:00.23 kworker/2:1
  436 message+  20   0    5100   2240   1068 S   0,3  0,1   0:00.75 dbus-daemon
 1326 root      20   0   37796   4112   3404 S   0,3  0,2   0:00.30 upowerd
 1657 montse    20   0   20720   1208    840 S   0,3  0,1   0:00.33 VBoxClient
 1742 montse    20   0  159996  15520  11288 S   0,3  0,8   0:00.97 unity-sett+
    1 root      20   0    4440   2536   1448 S   0,0  0,1   0:03.59 init
    2 root      20   0       0      0      0 S   0,0  0,0   0:00.00 kthreadd
    3 root      20   0       0      0      0 S   0,0  0,0   0:00.09 ksoftirqd/0
    4 root      20   0       0      0      0 S   0,0  0,0   0:00.00 kworker/0:0
    5 root       0 -20       0      0      0 S   0,0  0,0   0:00.00 kworker/0:+
    6 root      20   0       0      0      0 S   0,0  0,0   0:00.35 kworker/u6+
    7 root      20   0       0      0      0 S   0,0  0,0   0:00.54 rcu_sched
    8 root      20   0       0      0      0 S   0,0  0,0   0:00.00 rcu_bh
```

Paso 2: El comando top acepta varios parámetros, uno de ellos es el de ordenación de los procesos. Ordenad el resultado de la ejecución de top por el identificador de proceso, el PID, de mayor a menor. Para ello puede ayudaros la

información del manual, accesible tecleando man top.
Explicad cómo cambiar la ordenación y adjuntad una captura de pantalla con la nueva ordenación. Recordad que para salir de top se pulsa la tecla q (quit).

Solución:
Para ordenar por un campo concreto, pondremos el parámetro -o seguido del campo por el que queremos ordenar. El signo + delante del campo ordenará de mayor a menor, y el signo - ordenará de menor a mayor.

$ top -o +PID

```
top - 20:45:11 up 6 min,  2 users,  load average: 3,22, 3,36, 1,64
Tareas: 173 total,   1 ejecutar,  172 hibernar,    0 detener,    0 zombie
%Cpu(s): 15,1 usuario,  5,4 sist,  0,0 adecuado, 79,6 inact,  0,0 en espera,  0,
KiB Mem:   2064212 total,   961512 used,  1102700 free,    49944 buffers
KiB Swap:  2095100 total,        0 used,  2095100 free.   426856 cached Mem

  PID USUARIO   PR  NI    VIRT    RES    SHR S  %CPU %MEM     HORA+ ORDEN
 2685 montse    20   0    6840   1500   1096 R   0,7  0,1   0:00.15 top
 2653 montse    20   0   68536   4116   3476 S   0,0  0,2   0:00.45 deja-dup-m+
 2639 montse    30  10  205412  69956  44696 S   0,0  3,4   0:47.54 update-man+
 2582 montse    20   0    8280   3268   1692 S   0,0  0,2   0:00.62 bash
 2581 montse    20   0    2420    704    584 S   0,0  0,0   0:00.08 gnome-pty-+
 2573 montse    20   0  140924  17040  12708 S   3,0  0,8   0:02.53 gnome-term+
 2395 montse    20   0  128648  14948  11708 S   0,0  0,7   0:01.56 update-not+
 2392 montse    20   0   70012  10060   6580 S   0,0  0,5   0:00.43 unity-musi+
 2389 montse    20   0  121620   9532   6272 S   0,0  0,5   0:00.57 unity-file+
 2384 montse    20   0  109500  20092  10704 S   0,0  1,0   0:02.33 unity-scop+
 2344 root      20   0    4648    856    744 S   0,0  0,0   0:00.10 getty
 2289 root      20   0   28492   1248    936 S   0,0  0,1   0:00.19 VBoxService
 2280 montse    20   0  105032  13884   8576 S   0,0  0,7   0:01.98 unity-scop+
 2270 kernoops  20   0    6392   1136    872 S   0,0  0,1   0:00.02 kerneloops
 2160 montse    20   0    5468    276    224 S   0,0  0,0   0:00.09 cat
 2132 montse    20   0   58556   8848   7080 S   0,0  0,4   0:00.96 zeitgeist-+
 2108 montse    20   0   46232   5152   4228 S   0,0  0,2   0:00.54 zeitgeist-+
```

Apartado 1.2

Ahora nos centraremos en la información que muestra top en la segunda línea, que es la información sobre los procesos o tareas que está gestionando el sistema. Lo que haremos es crear un nuevo proceso y después lo detendremos, provocando un cambio de estado del proceso. Para realizar este ejercicio os proporcionamos el programa contador.c, que es un bucle infinito que escribe por pantalla un texto y un contador. En este ejercicio debéis poner capturas de pantalla de todos los pasos, para comprobar la evolución de la información que muestra top.

Paso 1: En primer lugar ejecutad el comando top y mostrad el número total de procesos que está gestionando el sistema.

Paso 2: A continuación, abrid una segunda ventana del Terminal y ejecutad el programa contador.c, que al ser un bucle infinito nos permite ir a la primera ventana de Terminal y ver la información de los procesos que muestra top. ¿Ha habido algún cambio? Razonar la respuesta.

Paso 3: Ahora iremos a la ventana de Terminal donde se ejecuta el contador y pulsaremos Ctrl+z para detener el proceso y de nuevo miraremos la información

de top. ¿Vemos algún cambio en la información que muestra top? ¿Qué ha sucedido? Razonar la respuesta.

Paso 4: Para finalizar esta parte del ejercicio, restableceremos el estado de los procesos del sistema. El proceso del programa contador está detenido. Lo que haremos es finalizarlo. Para finalizarlo usaremos el comando **ps** que nos dirá el PID del proceso del contador. Con ese PID finalizaremos el proceso con la siguiente instrucción: **$ kill -9 Id_proceso**

Solución:
Paso 1: Ejecutamos top y tenemos este resultado:

```
top - 20:26:33 up  3:37,  3 users,  load average: 3,39, 2,93, 2,55
Tareas: 176 total,   1 ejecutar,  175 hibernar,    0 detener,    0 zombie
%Cpu(s):  8,8 usuario,  6,7 sist,  0,0 adecuado, 84,5 inact,  0,0 en espera,  0,
KiB Mem:   2064212 total,  1463732 used,   600480 free,    85316 buffers
KiB Swap:  2095100 total,        0 used,  2095100 free.   782592 cached Mem
```

Paso 2: Ejecutamos contador.c en una segunda ventana del Terminal, y en la información de top de la primera ventana aparece un nuevo proceso, que es el contador de la segunda ventana.

```
top - 20:27:06 up  3:38,  3 users,  load average: 3,02, 2,87, 2,54
Tareas: 177 total,   3 ejecutar,  174 hibernar,    0 detener,    0 zombie
%Cpu(s): 15,5 usuario, 22,4 sist,  0,0 adecuado, 61,8 inact,  0,0 en espera,  0,
KiB Mem:   2064212 total,  1466128 used,   598084 free,    85316 buffers
KiB Swap:  2095100 total,        0 used,  2095100 free.   782580 cached Mem
```

Paso 3: Pulsamos Ctrl+z para detener el proceso del contador, y miramos la información de top. Vemos que el número de procesos sigue siendo el mismo pero aparece un proceso en estado detenido o blocked.

```
top - 20:32:38 up  3:44,  3 users,  load average: 3,23, 3,26, 2,82
Tareas: 177 total,   1 ejecutar,  175 hibernar,    1 detener,    0 zombie
%Cpu(s):  6,5 usuario,  4,7 sist,  0,0 adecuado, 88,8 inact,  0,0 en espera,  0,
KiB Mem:   2064212 total,  1466724 used,   597488 free,    85664 buffers
KiB Swap:  2095100 total,        0 used,  2095100 free.   782592 cached Mem
```

Lo que ha sucedido es que al pulsar Ctrl+z en la segunda pantalla del Terminal hemos provocado una **excepción** (transferencia de control al SO provocada por la ejecución de una instrucción de usuario no prevista), deteniendo la ejecución del contador. El proceso ha cambiado de estado, pasando de estar en ejecución (Run) a Detenido (Blocked). Si miramos la información de top de la primera ventana, como top muestra la información en tiempo real, aparecerá el nuevo estado del proceso contador, antes estaba en ejecución y ahora está detenido (blocked), el proceso ya no está utilizando CPU.

Paso 4: Usamos ps para ver el número de proceso del programa contador. En este caso es el pid 3396:

```
montse@montse-VirtualBox:~/pr1so$ ps
  PID TTY          TIME CMD
 3299 pts/11   00:00:00 bash
 3396 pts/11   00:00:03 contador
 3485 pts/11   00:00:00 ps
```

Finalizamos el proceso con el comando kill: **$ kill -9 3396**. A continuación, ejecutamos ps y comprobamos que ya no aparece el proceso.

```
montse@montse-VirtualBox:~/pr1so$ kill -9 3396
[1]+  Terminado (killed)      ./contador
montse@montse-VirtualBox:~/pr1so$ ps
  PID TTY          TIME CMD
 3299 pts/11   00:00:00 bash
 3486 pts/11   00:00:00 ps
montse@montse-VirtualBox:~/pr1so$
```

Consultamos top y vemos que el proceso detenido está finalizado, con lo que el sistema ya no lo gestiona.

```
top - 00:08:59 up  7:20,  3 users,  load average: 2,90, 2,43, 2,29
Tareas: 175 total,   1 ejecutar,  174 hibernar,    0 detener,    0 zombie
%Cpu(s):  2,0 usuario,  3,0 sist,  0,0 adecuado, 95,0 inact,  0,0 en espera,  0
KiB Mem:   2064212 total,  1466388 used,   597824 free,    86384 buffers
KiB Swap:  2095100 total,        0 used,  2095100 free.   782612 cached Mem
```

Apartado 1.3

En este apartado veremos la relación entre el número de procesadores disponibles y la información del sistema sobre el porcentaje de uso de la CPU.

Para saber cuántos procesadores tenemos, ejecutaremos el siguiente comando:
$ grep processor /proc/cpuinfo

Ahora, ejecutad el comando top. En una segunda ventana de Terminal, ejecutad el programa contador.c (bucle infinito). A continuación, fijaos en el comando top, que se está ejecutando en la primera ventana, en la información que muestra en la tercera línea. Es la información relativa al porcentaje de uso de la CPU.

Poned una captura de pantalla de los procesadores disponibles y otra captura de top. Explicad los valores de la tercera línea de información que muestra top, relacionándolo con el número de procesadores que tengáis.

Solución:
En este caso, disponemos de 3 procesadores:

```
montse@montse-VirtualBox:~/pr1so$ grep processor /proc/cpuinfo
processor       : 0
processor       : 1
processor       : 2
montse@montse-VirtualBox:~/pr1so$
```

La tercera línea del comando top muestra el porcentaje de uso de CPU en modo usuario, en modo sistema, e idle inactivo. La suma de los tres porcentajes será prácticamente del 100% de procesadores disponibles.

Si tenemos un solo procesador, la CPU estará siendo usada y veremos que el % de uso en modo usuario + el % de uso en modo sistema suman aprox el 100%. Si el % de uso de CPU en modo usuario + el % de uso en modo sistema sumados es bastante inferior al 100% debe ser porque tenemos más de un procesador.

En este ejemplo, tenemos 3 procesadores y el % de uso de la CPU en modo usuario (16,2) + el % de uso de la CPU en modo sistema (21,4) suman aprox un 34%, lo que significa que el proceso que se está ejecutando está usando uno de los 3 procesadores disponibles, y los otros 2 procesadores están libres, lo vemos en el % de inactividad, 62%.

```
top - 00:15:25 up  7:26,  3 users,  load average: 3,41, 2,56, 2,36
Tareas: 177 total,   3 ejecutar,  174 hibernar,    0 detener,    0 zombie
%Cpu(s): 16,2 usuario, 21,4 sist,  0,0 adecuado, 61,7 inact,  0,6 en espera,  0,
KiB Mem:   2064212 total,  1466824 used,   597388 free,    86544 buffers
KiB Swap:  2095100 total,        0 used,  2095100 free.   782640 cached Mem
```

Ejercicio 2 - Módulo 3 (Puntuación: 40% = 20% + 20%)

En este ejercicio practicaremos la asignación dinámica de memoria. El ejercicio consiste en realizar un programa en lenguaje C cuyo código deberéis entregar junto con el pdf con las respuestas de la práctica.

Apartado 2.1

En este apartado veréis el código de un programa que suma valores. El programa requiere para su ejecución dos parámetros de entrada, y muestra como resultado la suma de los números que hay entre el primer parámetro y el segundo parámetro, ambos incluidos.

A continuación podéis ver un par de ejemplos del resultado de la ejecución del programa:

```
montse@montse-VirtualBox:~/pr1so$ ./sumar 14 32
La suma de los números desde el 14 hasta el 32 es de 437
montse@montse-VirtualBox:~/pr1so$
montse@montse-VirtualBox:~/pr1so$ ./sumar 50 64
La suma de los números desde el 50 hasta el 64 es de 855
montse@montse-VirtualBox:~/pr1so$
```

El código que véis a continuación es un esqueleto del programa, y no está del todo correcto, veréis que contiene algunos errores de programación. Analizad el código del programa e identificad los errores.

Las líneas están numeradas para facilitar la respuesta al ejercicio. De cada uno de los errores que encontréis, indicad en qué línea está y explicad por qué es un error y cómo arreglarlo.

El código del programa es el siguiente:

```
1   #include <stdlib.h>
2   #include <stdio.h>
3   #include <error.h>
4   #include <string.h>
5
6   void panic (char *message)
7   {
8    fprintf (stderr, "%s\n", message);
9    exit (-1);
10  }
11
12  int main (int argc, char *argv[])
13  {
14   int valor1, valor2, total, i;
15   int *result;
16
17   valor1 = atoi (argv[1]);
18   valor2 = atoi (argv[2]);
19
20   if (argc != 2) panic ();
21
22   if (valor1 < 0 || valor2 < 0) panic ();
23
24  result[0] = valor1;
25
26   // Asignar memoria dinámica
27   int nums = valor2 - valor1 +1;
28   result = malloc(nums * sizeof(int));
29
30   for (i = 0; i < nums; i++) {
31        result[i] = result[i-1] + (valor1 + i);
32        total = result[i];
33   }
34
35   printf("La suma de los números desde el %d hasta el %d es de %d \n",
valor1, total);
36
37   exit (0);
38  }
```

Solución:
Los errores que hay en el código anterior son los siguientes:

Línea 17-18: no se pueden asignar los parámetros de entrada a las variables del programa antes de comprobar que los parámetros sean correctos. Si alguno de los parámetros no se hubiera entrado o no fuera correcto, podría provocar que la asignación de memoria posterior fuera demasiado pequeña, generando una excepción porque sería insuficiente para realizar los cálculos, o demasiado grande, con lo que se podría retornar NULL por no haber memoria suficiente para asignarle, generando también una excepción.

Línea 20: el programa requiere dos parámetros, pero debemos tener en cuenta que el nombre del programa que ejecutamos también cuenta como parámetro. Por lo tanto, la comparación debería ser de 3 parámetros en lugar de 2.

Líneas 20 y 22: la función panic requiere un mensaje a mostrar.

Línea 24: Falta la comprobación de que el segundo parámetro sea mayor o igual que el primero, mostrando un mensaje de error en caso contrario.

Línea 25: no se le puede asignar un valor a la variable result antes de asignarle memoria (malloc). Recordar que tampoco podríamos usar la variable después de liberar la memoria (free).

Línea 30: cuando hacemos una asignación dinámica de memoria, siempre hay que comprobar si la asignación ha sido correcta o si se ha producido algún error. En este caso, faltaría la siguiente comprobación del resultado de la asignación de memoria:

```
// Comprobar si hay error en la reserva de memoria
if (result == NULL) panic("malloc: error en la asignación de memoria");
```

Línea 31: si el bucle for lo empezamos en i = 0 generaremos un error ya que se intentará acceder a una posición negativa del array, result[-1].

Línea 30: la variable total no está inicializada. Si no le asignamos un valor inicial, en el caso que el segundo parámetro sea igual al primero, el programa no entra en el bucle, con lo que el resultado de la ejecución será incorrecto porque no se asigna ningún valor a la variable total. Lo que habría que hacer es asignarle el valor del primer parámetro (total = valor1;) por si los dos parámetros son iguales. En el resto de casos, el programa siempre entrará en el bucle y se le asignará un valor.

Línea 36: el printf final muestra 3 valores pero solamente tiene dos variables. Falta añadir el valor2.

Línea 39: falta liberar la memoria asignada dinámicamente.

```
// Free memory
free(result);
```

Apartado 2.2

En el fichero zip encontraréis el programa sumar.c, que es el esqueleto de programa mostrado en el apartado anterior. Modificad el programa para corregir los errores que hayáis identificado. El programa requiere que se indiquen dos parámetros de entrada. El resultado de la ejecución del programa sumar.c será la suma de los números que hay entre el primer parámetro y el segundo parámetro, ambos incluidos.

El resultado esperado de la ejecución del programa es el que se muestra en la siguiente captura, donde tenéis varios ejemplos:

```
montse@montse-VirtualBox:~/pr1so$ ./sumar 14 32
La suma de los números desde el 14 hasta el 32 es de 437
montse@montse-VirtualBox:~/pr1so$
montse@montse-VirtualBox:~/pr1so$ ./sumar 50 64
La suma de los números desde el 50 hasta el 64 es de 855
montse@montse-VirtualBox:~/pr1so$
montse@montse-VirtualBox:~/pr1so$ ./sumar 83 116
La suma de los números desde el 83 hasta el 116 es de 3383
montse@montse-VirtualBox:~/pr1so$
montse@montse-VirtualBox:~/pr1so$ ./sumar 247 378
La suma de los números desde el 247 hasta el 378 es de 41250
montse@montse-VirtualBox:~/pr1so$
```

El programa recibe como parámetros dos números con valores positivos o cero. Debéis realizar un control de los parámetros de entrada, tanto del número de parámetros como de que sus valores sean correctos.

Añadid en el documento de respuestas una captura de pantalla con el resultado de la ejecución de vuestro programa.

Solución:
Una posible solución sería la siguiente.

```
#include <stdlib.h>
#include <stdio.h>
#include <error.h>
#include <string.h>

void panic (char *message)
{
 fprintf (stderr, "%s\n", message);
 exit (-1);
}

int main (int argc, char *argv[])
{
 int valor1, valor2, total, i;
 int *result;

 if (argc != 3) panic ("Error: el programa requiere dos parámetros");

 valor1 = atoi (argv[1]);
 valor2 = atoi (argv[2]);

 if (valor1 < 0 || valor2 < 0) panic ("Los parámetros no pueden ser negativos");
  if (valor2 < valor1) panic ("El segundo parámetro debe ser mayor o igual que el primero");

 // Asignar memoria dinámica
 int nums = valor2 - valor1 +1;
 result = malloc(nums * sizeof(int));

 // Comprobar si hay error en la reserva de memoria
 if (result == NULL) panic("malloc: error en la asignación de memoria");

 result[0] = valor1;
 total = valor1;

 for (i = 1; i < nums; i++) {
            result[i] = result[i-1] + (valor1 + i);
            total = result[i];
 }

  printf("La suma de los números desde el %d hasta el %d es de %d \n", valor1, valor2, total);

 // Free memory
 free(result);

 exit (0);
}
```

Observaciones:

- Recordad verificar si la memoria se ha asignado correctamente
- Antes de finalizar el programa, es necesario liberar explícitamente la memoria que haya sido solicitada.
- En la entrega debéis adjuntar el código fuente de vuestro programa y añadir en el documento de las respuestas una captura de pantalla que muestre su funcionamiento.

Ejercicio 3 - Módulo 4 (Puntuación: 30% = 10% + 10% + 10%)

En este ejercicio trabajaremos el concepto de dispositivos de entrada / salida y el uso de pipes para encadenar la ejecución de comandos.

Apartado 3.1

Abrid dos ventanas del Terminal. Desde la primera ejecutad el comando tty. Este comando retorna como resultado un nombre de fichero que representa la ventana. Por ejemplo /dev/tty1.

Desde la segunda ventana del Terminal ejecutad lo siguiente:

- $ ls -l /dev
- $ ls -l /dev > output
- $ ls -l > nombre_del_fichero_retornado_por_el_comando_tty

Contestad las siguientes preguntas:

Paso 1: desde el punto de vista del usuario, qué diferencia hay entre las 3 ejecuciones anteriores.

Solución:
En la primera ejecución, el resultado se muestra por pantalla (salida estándar).
En la segunda ejecución, la salida estándar se ha redireccionado a un fichero, con lo que el resultado se escribirá en el fichero.
En la tercera ejecución, el resultado se mostrará en la otra ventana, donde hemos ejecutado el tty.

Paso 2: desde el punto de vista del programador del comando ls, qué diferencia hay entre las 3 ejecuciones anteriores.

Solución:
Desde el punto de vista del programador, no hay ninguna diferencia. El comando ls siempre escribe el resultado por la salida estándar. Es el intérprete de comandos el encargado de gestionar y tratar los metacaracteres, y asociar la salida estándar a la pantalla, a un fichero o a un dispositivo. Pero esto es transparente al código del ls.

Apartado 3.2

Compilad y ejecutad el programa numeros.c que os proporcionamos para ver el resultado de su ejecución. Este programa muestra por pantalla el contenido de un array. El array contiene valores numéricos.

```
// programa numeros.c

#include <stdio.h>

int main ()
{
 int array[10] = {88, 35, 15, 1, 7, 52, 80, 9, 47, 60};
 int i;

  for(i = 0; i < 10; i++)
    printf("%d\n", array[i]);

  return 0;
}
```

No tenéis que modificar el programa, solamente ejecutarlo. Entonces, ejecutando el programa numeros y usando pipes, formulad los comandos necesarios de manera que muestren el resultado indicado en los siguientes enunciados.

Paso 1: ordenad el resultado de la ejecución del programa numeros.c en orden descendente

Solución:
$ ./numeros | sort -nr

Nota: añadir la n para que ordene numéricamente, sin la n ordena alfabéticamente.

```
montse@montse-VirtualBox:~/pr1so$ ./numeros | sort -nr
88
80
60
52
47
35
15
9
7
1
montse@montse-VirtualBox:~/pr1so$
```

Paso 2: calculad cuántos valores muestra el programa numeros.c al ejecutarse

Solución:
$ ./numeros | wc -l

```
montse@montse-VirtualBox:~/pr1so$ ./numeros | wc -l
10
montse@montse-VirtualBox:~/pr1so$
```

Paso 3: calculad cuántos de los valores que muestra el programa numeros.c contienen el número 5

Solución:
$ ./numeros | grep 5 | wc -l

```
montse@montse-VirtualBox:~/pr1so$ ./numeros | grep 5
35
15
52
montse@montse-VirtualBox:~/pr1so$ ./numeros | grep 5 | wc -l
3
montse@montse-VirtualBox:~/pr1so$
```

Apartado 3.3

Formulad los siguientes comandos, usando pipes, de manera que muestren el resultado indicado en los enunciados.

Paso 1: Lee la fecha del sistema y escribe la hora (hora, minutos y segundos) en un fichero llamado hora.txt

Solución:
Dos de las posibles soluciones serían las siguientes.
$ date | cut -c12-19 > hora.txt
$ date +%X > hora.txt

```
montse@montse-VirtualBox:~/pr1so$ date | cut -c12-19 > hora.txt
montse@montse-VirtualBox:~/pr1so$ cat hora.txt
15:43:04
montse@montse-VirtualBox:~/pr1so$
```

Paso 2: Muestra por pantalla el contenido del fichero hora.txt y del fichero hello_world.c, utilizado en el primer ejercicio de la práctica, y cópialo todo en dos ficheros distintos, hello1.txt y hello2.txt. Como resultado, los ficheros hello1.txt y hello2.txt serán iguales y contendrán la hora y el contenido de hello_world.c. Utiliza pipes para realizar todas las instrucciones en una sola línea.

Solución:
$ cat hora.txt hello_world.c | tee hello1.txt hello2.txt

```
montse@montse-VirtualBox:~/pr1so$ cat hora.txt hello_world.c | tee hello1.txt he
llo2.txt
15:43:04

#include <stdio.h>

int main() {
   printf("Hello World!\n");
   return 0;
}
```

```
montse@montse-VirtualBox:~/pr1so$ cat hello1.txt
15:43:04

#include <stdio.h>

int main() {
   printf("Hello World!\n");
   return 0;
}

montse@montse-VirtualBox:~/pr1so$ cat hello2.txt
15:43:04

#include <stdio.h>

int main() {
   printf("Hello World!\n");
   return 0;
}

montse@montse-VirtualBox:~/pr1so$
```

Recursos

- Módulos 1, 2, 3 y 4 de la asignatura.
- Documento "Introducción a la programación en Unix", disponible en el campus virtual.
- Documento "Intérprete de comandos", disponible en el aula, o bien cualquier otro manual similar.
- El propio manual incluido en las máquinas Unix, manual accesible con el comando man.
- El aula "Laboratorio de Sistemas Operativos", donde podéis plantear dudas relativas al entorno UNIX, programación, C, etc.

Criterios de valoración

En la corrección se tendrán en cuenta los siguientes aspectos:

- Las respuestas deberán estar razonadas siguiendo los conceptos estudiados en la asignatura.
- Se valorará la correcta justificación de las respuestas, aunque las explicaciones deberán ser breves y concisas.
- Se valorará la capacidad de análisis de los resultados de los ejercicios y la metodología seguida para su obtención.
- El código entregado debe poder ejecutarse correctamente en cualquier máquina Linux. Que el código se ejecute correctamente en vuestro ordenador no implica necesariamente que sea correcto. Revisad vuestra solución sin ignorar los warnings que os pueda mostrar el compilador.

- El peso de cada ejercicio está indicado en el enunciado.

Formato y fecha de entrega

Se entregará un fichero zip que contenga un fichero pdf con las respuestas a los ejercicios y también los ficheros .c con los códigos fuente de los ejercicios.

El nombre del fichero zip a entregar tendrá el siguiente formato: "Apellido1Apellido2_PRA1.zip". Los Apellidos se escribirán sin acentos. Por ejemplo, Marta García Vallès entregaría un fichero de nombre GarciaValles_PRA1.zip.

La fecha límite de entrega de la práctica será el miércoles día **9 de noviembre** a las 24:00 h.

Nota: Propiedad intelectual

Al contestar una PEC o una práctica, a menudo es inevitable hacer uso de recursos creados por terceras personas. Para que esto no suponga un plagio, siempre que una PEC o práctica haga uso de recursos ajenos, se citará la fuente de manera correcta.