

INFORMÁTICA I

CUADERNO DE LABORATORIO



Universidad de Deusto – Facultad de Ingeniería
2º Ingeniería Técnica Industrial

María José Gil Larrea
Jorge González Barturen
Manuel de la Herrán Gascón
© Octubre 2000



INFORMÁTICA I: Cuaderno de Laboratorio 2ª Edición

Departamento de Ingeniería del Software – Sistemas Concurrentes. Bilbao, octubre 2000.

Índice

INTRODUCCIÓN.....	1
PRESENTACIÓN.....	1
PRÁCTICA 1: TOMA DE CONTACTO CON LINUX.....	3
Objetivos	3
Requisitos	3
Desarrollo de la práctica.....	3
Tareas Pendientes.....	6
PRÁCTICA 2: SHELLS Y EDITORES.....	9
Objetivos	9
Requisitos	9
Desarrollo de la práctica.....	9
Tareas Pendientes	12
LECTURA: EXTRACCIÓN DE ARCHIVOS PRODUCIDOS DE UNA MÁQUINA LINUX	12
Copia de Ficheros a Disquete.....	12
Acceso vía FTP	12
Impresión en Linux	13
PRÁCTICA 3: EL COMPILADOR “CC”	15
Objetivos	15
Requisitos	15
Desarrollo de la práctica.....	15
Tareas pendientes.....	16
PRÁCTICA 4: MANEJO DE SEÑALES.....	17
Objetivos	17
Requisitos	17
Desarrollo de la práctica.....	17
Tareas pendientes.....	19

PRÁCTICA 5: MANEJO DE PROCESOS.....21

Objetivos	21
Requisitos	21
Desarrollo de la práctica:	21

**PRÁCTICA 6: MANIPULACIÓN DE PROCESOS A NIVEL DE USUARIO.
(kill, ps).....25**

Objetivos	25
Requisitos	25
Desarrollo de la práctica	26
Tareas pendientes.....	27

**PRÁCTICA 7: LLAMADAS RELACIONADAS CON LA E/S ESTÁNDARES.
.....29**

Objetivos	29
Requisitos	29
Desarrollo de la Práctica.....	29

**PRÁCTICA 8: MANEJO DE “PIPES” Y GENERACIÓN DE MÚLTIPLES
HIJOS.....33**

Objetivos	33
Requisitos	33
Desarrollo de la práctica	33
Tareas pendientes.....	35

PRÁCTICA 9: APLICACIÓN DE LO APRENDIDO A UN CASO REAL37

Objetivos	37
Requisitos	37
Desarrollo de la práctica	37
Tareas pendientes.....	39

Introducción

Tradicionalmente, se han venido utilizando, y se siguen utilizando, sistemas operativos *Unix-like* tanto en sistemas de control de producción como en sistemas de soporte de gestión. No en vano estos sistemas siguen gozando de una cuota de mercado considerablemente elevada, especialmente en entornos industriales. Más recientemente, Linux es el sistema operativo elegido para un porcentaje elevado de computadores que dan soporte a los servicios de comunicación.

Por todo ello, se persigue mediante la asignatura *Conceptos de Sistemas Operativos* adquirir un conocimiento de los sistemas operativos en general, y de los *Unix-like* en particular, comprendiendo su definición, evolución histórica, funciones y manera de acceder a sus servicios, logrando la capacidad necesaria para manejar los componentes del sistema operativo Linux, especialmente en cuanto a procesos se refiere.

Para facilitar su asimilación por parte del alumnado, los conceptos y principios detallados en el desarrollo teórico de la asignatura se implementen en las sesiones de laboratorio. El presente texto pretende ser una guía del trabajo a realizar antes, durante y después de cada una de ellas.

Presentación

Cada una de las prácticas recogidas en este *Cuaderno de Laboratorio* presenta una serie de objetivos que el alumno debe alcanzar con su realización.

Antes de cada una de las prácticas, se deben cumplir los requisitos especificados, necesarios para completar correctamente, el trabajo previsto. El desarrollo de las mismas en el laboratorio comprende una serie de cuestiones y ejercicios que el alumno resolverá siguiendo las indicaciones y explicaciones de los profesores de la asignatura. Además, en algunas de ellas se plantea una serie de cuestiones complementarias que el alumno tendrá que realizar después de la sesión de laboratorio como trabajo personal.

La lista de distribución de la asignatura pretende ser el punto de encuentro de alumnos y profesores de Conceptos de Sistemas Operativos. Es posible enviar a la lista de distribución mensajes con preguntas, dudas, reflexiones o sugerencias sobre cualquier aspecto de la asignatura. Estos mensajes serán compartidos por todos los miembros de la lista.

Para cualquier duda sobre el funcionamiento de la lista de distribución, realizar preguntas a los profesores responsables de la asignatura, descarga de documentación complementaria, etc., se puede consultar la página web de la asignatura en la dirección: <http://www.eside.deusto.es/asignaturas/inf1/> .



Práctica 1: Toma de contacto con Linux.

Objetivos

- Tomar contacto real – desde el punto de vista de un usuario – con el sistema operativo Linux.
- Familiarizarse y hacer uso de los comandos del sistema operativo Linux más comunes, y ejecutables en la gran mayoría de sistemas *Unix-like*, entre los que cabe citar Solaris, AIX, HP-UX o Minix.

Requisitos

- Conocer la cuenta y la clave para poder acceder a la máquina Linux en la que se desarrollarán las prácticas.

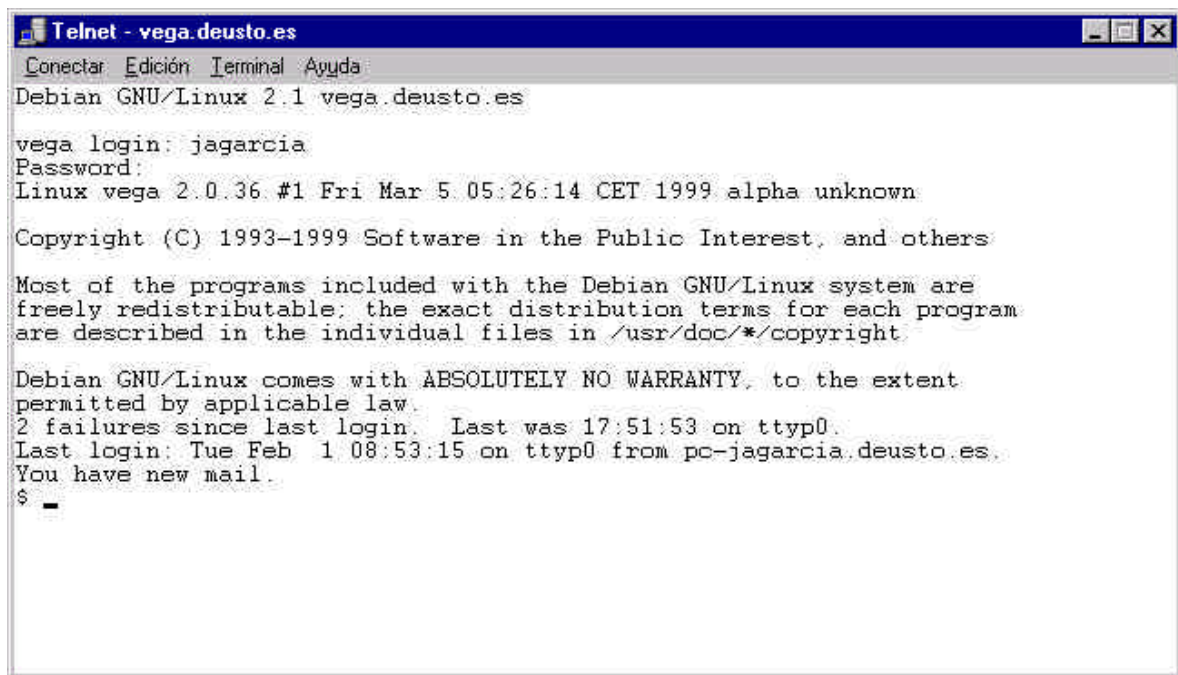


- Lectura del tema 3: “*Interfaz de Usuario Proporcionada por Linux*” de los apuntes de la asignatura, prestando especial atención a los apartados 3.1, 3.2 y 3.3.
- Lectura de los temas 3 (“*En el principio*”) y 4 (“*El shell de Unix*”) de la “Guía del Usuario Linux” tomando nota de aquellos comandos más relevantes.

Desarrollo de la práctica

Desde los ordenadores conectados a la red del Centro de Cálculo de la Facultad de Ingeniería – ESIDE, se puede ejecutar el programa `telnet.exe` para establecer una sesión en la máquina Linux remota (`vega.deusto.es`) que tiene sistema operativo Linux, con la que se trabajará. También se puede establecer una sesión en `vega.deusto.es` es con la utilidad Reflection, instalada en los ordenadores del Centro de Cálculo de ESIDE.

1. **Establecimiento de una sesión.** El primer paso será acceder a un ordenador del Centro de Cálculo en el dominio CDK. Una vez establecida la sesión en él, es necesario ejecutar un programa que permita establecer sesiones remotas (`telnet.exe` o Reflection) y conectarse a la máquina `vega.deusto.es`. Cuando aparezca el prompt del sistema Linux se introducirá el nombre de usuario (cuenta) y la clave (Ver Figuras 1 y 2).



Facultad de Ingeniería

Figura 1. Sesión en vega.deusto.es utilizando telnet.exe.

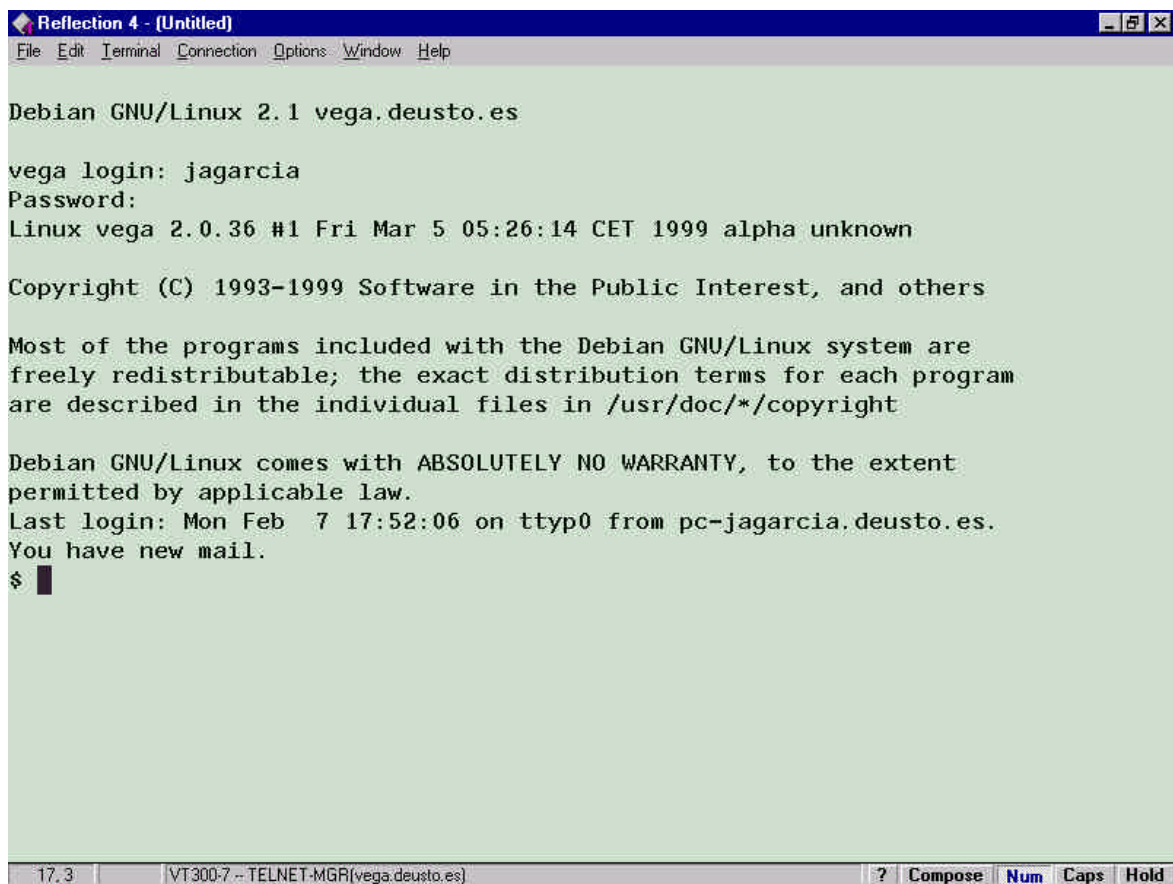


Figura 2. Sesión en vega.deusto.es utilizando Reflection.

2. **Comando man.** Este comando permite el acceso a la ayuda de Linux. Para obtener información sobre el propio comando de ayuda será necesario teclear `man man`.

3. Probar el funcionamiento y opciones de algunos de los comandos más útiles, como son los siguientes:

`ls`

`rm`

`mv`

`cp`

`cd`

`cat`

`sort`

exit

logout

who

wc



Tareas Pendientes.

1. Cambiar la clave de acceso a la máquina Linux. Consultar la ayuda del comando `passwd` y utilizarlo.

2. Consultar la ayuda de los comandos siguientes y aprender a manejarlos:

`mkfs` (dar formato). Anotar los parámetros necesarios para dar distintos formatos a los sistemas de ficheros (DOS, ext2, ...)

`mount` (montar sistemas de ficheros).

umount (desmontar sistemas de ficheros).





Práctica 2: Shells y Editores.

Objetivos

- Familiarizarse con el uso de la shell de Linux, accediendo a los valores de las variables del sistema. Tomar contacto con algunas de las posibilidades de redireccionamiento de la entrada y salida de los comandos.
- Aprender a desenvolverse y a utilizar los editores de texto más comunes en Linux prestando especial atención a los shells *vi* y *joe*.

Requisitos

- Leer el apartado 3.6 de los apuntes de la asignatura que hacen referencia a los distintos editores que existen en sistemas Linux, tomando nota de los nombres de los principales editores, así como sus ventajas e inconvenientes.

- Leer el capítulo 4: “*Shells*” de los apuntes de la asignatura, poniendo especial atención en los puntos 4.1, 4.2 y 4.3, en los que se explican diversos conceptos de las Shells en Linux.
- Leer el capítulo 6 (“*Trabajando con Unix*”) y el apartado 9.1 (“*Personalización del bash*”) de La Guía Linux para el Usuario.
- Recordar la sintaxis del lenguaje C para escribir programas sencillos. Leer, para ello, el *Anexo 8. Programación en C* de los apuntes de la asignatura.

Desarrollo de la práctica

1. Averiguar cuál es la shell con la que se está trabajando en Linux. Esta información está en la variable del sistema SHELL. El contenido de las variables del sistema se puede consultar con el comando `set`. Teclear `man set` para conseguir ayuda sobre el uso de este comando.

2. Practicar la orden `grep`, `sort` y el manejo de pipes o tuberías. Probar en la máquina y escribir los comandos necesarios para:

a- Listar todos los archivos cuyo propietario o grupo sea *root*.

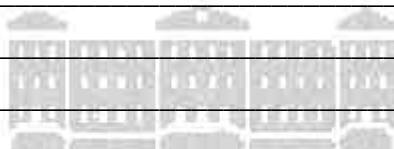
b- Crear un fichero de nombre *orden.dat* que contenga los nombres de todos los ficheros de un directorio, en orden alfabético.

c- Listar todos los ficheros de un directorio presentándolos en orden alfabético.



3. Utilizar el editor `joe` para escribir el código fuente de un programa de nombre *suma.c* que lea del teclado dos números enteros y muestre su suma por la pantalla. Recordar que, dentro de este editor, la combinación de teclas `^KZ` permite acceder al prompt del sistema, si bien el editor queda abierto.

4. Leer el epígrafe *Introducción a vi* del capítulo 3 de los apuntes de la asignatura tomando nota de los distintos modos de trabajo que presenta. Extrayendo también los comandos más habituales del mismo, como por ejemplo salvar un fichero, abrirlo, moverse por las líneas, salir del editor, ...



Facultad de Ingeniería

5. Repetir el ejercicio del apartado 3. de la presente práctica haciendo uso del editor *vi*. Llamar al fichero resultante *sumavi.c*.

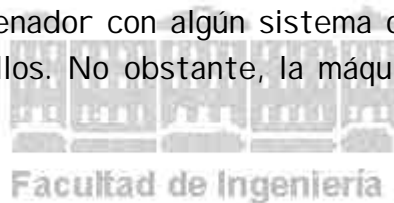
Tareas Pendientes

1. Lectura del documento siguiente, que permite hacerse con los ficheros elaborados durante las sesiones de laboratorio de Linux.

Lectura: Extracción de archivos producidos de una máquina Linux

Copia de Ficheros a Disquete

Exige tener acceso físico a la máquina Linux en que residen, y por tanto a su disquetera. Los ficheros elaborados bajo Linux se pueden copiar a un disquete y ser trasladados a un ordenador con algún sistema operativo de la familia MS Windows para acceder a ellos. No obstante, la máquina Linux debe cumplir los siguientes requisitos:



- Linux debe estar preparado para dar soporte al sistema de ficheros de *DOS*. Esta opción deberá haber sido seleccionada en el momento de llevar a cabo la instalación de Linux.
- La disquetera del ordenador debe estar configurada para soportar el montaje de sistemas de ficheros *dos*.. Como las opciones de montaje relativas a sistemas de ficheros se almacenan en el fichero `/etc/fstab`, en la línea correspondiente a la disquetera (`/dev/fd0`) deberá indicar que ésta soporta el sistema de ficheros *dos* en vez del sistema de ficheros *ext2* (Linux Native).

Si el sistema cumple estos requisitos, sólo se podrán copiar ficheros a la disquetera si se tienen privilegios para montar sistemas de ficheros en `/dev/fd0`. Si es así, se introduce un disquete previamente formateado bajo MS-DOS o MS Windows, o bien un disquete sin formatear (es posible darle formato mediante el comando `mkfs`). Una vez que se dispone de un disquete con formato en la disquetera, se monta mediante el comando `mount` y se copian archivos a la misma con el comando `cp`. Cuando la copia ha finalizado, y antes de extraer el disquete es necesario desmontar el sistema de ficheros de `/dev/fd0`, usando el comando `umount`.

Acceso vía FTP

Si la máquina que dispone de Linux no es físicamente accesible, es decir, se trabaja con ella desde otro ordenador, por ejemplo por medio de *telnet*, no se

podrá utilizar la disquetera, y por tanto no será posible llevar copiar directamente los archivos a la misma.

En este caso, es posible utilizar el protocolo FTP¹ para transferir ficheros desde la máquina remota. Las transferencias de ficheros vía FTP se hacen entre dos programas denominados "cliente FTP" y "servidor FTP". Estos dos programas deben estar instalados cada uno en una de las máquinas que van a intervenir en la transferencia de ficheros (suponiendo que ambas máquinas pueden conectarse la una con la otra a través de alguna red).

En el ordenador en el que se encuentra instalado el sistema operativo Linux, el administrador del sistema será el encargado de instalar y configurar el servidor FTP de manera que atienda las solicitudes de transferencia de archivos. El otro ordenador precisará de un navegador web (MS Internet Explorer, Netscape Communicator, etc.) que incorpora un cliente de FTP, o bien de un programa específico que desempeñe este cometido, como por ejemplo el que se instala junto con Microsoft Windows NT, `c:\winnt\system32\ftp.exe`. No obstante, en este último caso puede ser necesario el conocimiento de los comandos necesarios para la transferencia de ficheros y el recorrido de la estructura de directorios de la máquina remota, propios del protocolo FTP.

Por otro lado, además de disponer del cliente y servidor FTP, es necesaria una cuenta de acceso a la máquina remota vía FTP, que generalmente coincidirá con la cuenta que se utiliza para el establecimiento de la sesión en Linux.

Una vez llevada a cabo la transferencia, bastará generalmente con "cerrar" el cliente FTP, si bien debe tenerse cuidado de finalizar la conexión con la máquina remota de manera correcta, aunque esta tarea suele llevarse a cabo de manera automática. En cualquier caso, cerrado o no el cliente FTP, los ficheros ya serán accesibles dentro de la máquina local.

Impresión en Linux

Otra posibilidad de cara a la obtención de copias de los archivos elaborados bajo Linux, radica en la obtención de copias impresas. Para ello, basta con tener instalada y configurada una impresora en el sistema Linux.

¹ File Transfer Protocol

Para poder instalar una impresora en un ordenador con Linux, ésta tiene que ser admitida por el sistema, es decir, deben localizarse los *drivers* necesarios para hacer funcionar la citada impresora en Linux. Es importante destacar el hecho de que no todos los dispositivos compatibles con sistemas MS-DOS o MS-Windows son compatibles con Linux.

También es necesario instalar el programa `lpd`, que se encarga de la gestión de la impresora. La instalación y configuración de esta aplicación es responsabilidad del administrador del sistema Linux.



Práctica 3: El Compilador “cc”.

Objetivos

- Tomar contacto y familiarizarse con el manejo del compilador `cc`, el compilador estándar de Linux.
- Comprender y poner en práctica el proceso de elaboración de un programa en lenguaje C, haciendo uso de un editor para la generación del código fuente, y del compilador `cc` para la obtención de un fichero ejecutable, de nombre definido por el usuario en el momento de la compilación.

Requisitos

- Leer el *Anexo II: Programación en C* de los apuntes de la asignatura en el que se recuerdan todos los fundamentos de la programación estructurada y se repasa la sintaxis del lenguaje C.
- Leer el apartado 5.2 de los apuntes de la asignatura en el que se explica lo que es un compilador.

Desarrollo de la práctica

1. Acceder a la ayuda el línea del compilador `cc`. `man cc`. Tomar nota de los aspectos más destacables.

2. Escribir los parámetros que hay que introducir al compilador para obtener, a partir del programa fuente, el programa ejecutable con una sola ejecución de `cc` (compilar y montar).

3. ¿Cuál es el nombre del fichero ejecutable que resulta de la ejecución del compilador?
 ¿Es posible variar este nombre?

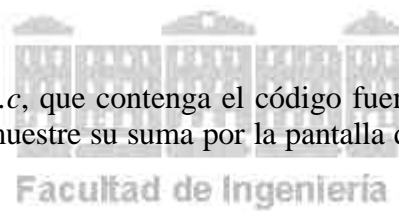
4. Haciendo uso del editor `vi`, generar un programa que pida por teclado tres números enteros y muestre por la pantalla el mayor de los tres. Compilarlo con el compilador `cc`.

5. Una vez obtenido el ejecutable, ejecutarlo para probarlo. ¿Por qué no se puede ejecutar el programa tecleando simplemente el nombre del fichero ejecutable?

6. ¿Qué solución hay para no tener este problema a la hora de ejecutar ficheros desde una cuenta de usuario de un alumno?

Tareas pendientes

1. Crear un fichero, *suma3.c*, que contenga el código fuente de un programa que lea del teclado tres números y muestre su suma por la pantalla del ordenador.



2. Obtener en un disquete todos los ficheros de los programas fuente realizados en esta práctica y de los ficheros *suma.c* y *sumavi.c* de la práctica anterior. Anotar los pasos realizados.

Práctica 4: Manejo de señales.

Objetivos

- Conocer y saber usar el mecanismo de comunicación entre procesos consistente en el envío de señales.
- Comprender la diferencia fundamental entre este mecanismo de comunicación y los mecanismos de transmisión de información.

Requisitos

- Leer y comprender el apartado 5.4 *Señales* de los apuntes de la asignatura.
- Comprender por qué para el intercambio de señales entre procesos de usuario es imprescindible, que éste se lleve a cabo entre procesos de igual identificativo de usuario.

Facultad de Ingeniería

- Leer y comprender el apartado 4. *Llamadas al Sistema* de la Guía Linux de Programación.

Desarrollo de la práctica

1. ¿Qué es una llamada al sistema?

2. Comprender los fragmentos de código y los ejemplos que aparecen en el apartado 5.4 *Señales* de los apuntes de la asignatura.

3. Plantear una solución que permita garantizar que cierta señal estará siempre capturada, se de cuando se dé.

4. Realizar un organigrama que muestre el funcionamiento de la función *signal()*: su uso para la captura de una señal.



5. Representar mediante un organigrama del funcionamiento del sistema operativo cuando éste recibe una llamada *kill*, que implica el envío de una señal a otro proceso del mismo usuario, y que no siempre conlleva la eliminación del proceso receptor.

Tareas pendientes

1. Programar en C una función que ejecute un comando del sistema como proceso hijo, de manera que el nombre de dicho comando será un parámetro de entrada a la función. El proceso padre, no deberá ser interrumpido, es decir, deberá ignorar las señales SIGINT o SIGQUIT mientras se está ejecutando concurrentemente con el proceso hijo; si bien, el proceso hijo deberá conservar el tratamiento que el proceso padre tenía para estas llamadas antes de la creación del proceso hijo. Probar esta función mediante un programa que la invoque una única vez.





Práctica 5: Manejo de Procesos.

Objetivos

- Comprender y observar la ejecución simultánea de diversos procesos en un mismo sistema..
- Adquirir la capacidad de utilizar los servicios de Linux para lograr la ejecución simultánea de varios procesos, generados a partir de un mismo fichero fuente..

Requisitos

- Leer el capítulo 6 (“Procesos”) de los apuntes de la asignatura, hasta el apartado 6.4 en los que se introduce el concepto de proceso.
- Tomar nota de los estados por los que puede pasar un proceso en Linux.

- Entender cómo Linux asigna identificadores a los procesos.

Desarrollo de la práctica:

1. Realizar un programa en el que se cree un proceso con la función *fork()*. El programa debe mostrar en todo momento, sacando mensajes por la pantalla, cuál es el proceso que está en ejecución (el padre o el hijo). Lanzar a ejecución el programa y anotar los mensajes generados por pantalla.

2. Desarrollar y ejecutar un programa que realice las siguientes operaciones:

- Inicializar una variable de tipo entero con un valor.
- Visualizar por pantalla el valor de esa variable.
- Crear un proceso hijo con la función *fork()*.

- Comprobar cómo el proceso hijo también tiene la variable definida en el padre. Para ello, cambiar el valor de la variable, en el código del proceso hijo y mostrar por pantalla el valor.
- Al acabar el proceso hijo, mostrar por la pantalla el valor de la variable en el proceso padre.







Práctica 6: Manipulación de procesos a nivel de usuario. (kill, ps).

Objetivos

- Observar los procesos existentes en el sistema, adquiriendo la capacidad de manipularlos directamente.
- Saber elaborar programas que permitan contemplar la existencia de procesos en varios estados, así como su eliminación.

Requisitos

- Repasar el modelo de procesos de Linux. Para ello leer y comprender el capítulo 6 de los apuntes de la asignatura.
- Recordar la utilización del comando `man` que permite acceder a la ayuda relativa a los comandos Linux. (ver la práctica 1, “Toma de contacto con Linux”).
- Hacer uso de este comando para obtener ayuda sobre los comandos `ps` y `kill`.

Comando `ps`

Comando `kill`

- Repasar los conceptos relativos al paso de señales. (Ver práctica 4, “Manejo de señales”).

- Repasar el apartado “Jerarquía de Procesos” descrito en el capítulo 6 de los apuntes de la asignatura y elaborar un gráfico en el que se observen los procesos involucrados tanto en el arranque del sistema como en el mantenimiento de una interacción con el mismo (es decir, dibujar la relación entre los procesos *init*, *login* y *shell (bash)*). Consultar, para ello, el apartado 3.2 *Linux Despierta* de la Guía del usuario Linux.



- Leer las páginas 52 a 56 de la Guía del Usuario Linux en las que se hace referencia al uso del control de los trabajos.

Desarrollo de la práctica

1. Realizar, compilar y ejecutar en segundo plano un programa que tenga un bucle infinito. Una vez lanzado, visualizar todos los procesos existentes en el sistema mediante el comando `ps`; eliminarlos mediante el comando `kill`. Anotar el comando necesario para eliminar un proceso.
2. Ejecutar un proceso padre finito que cree un proceso hijo infinito, de manera que el padre quede esperando indefinidamente el fin del hijo. Anotar el resultado de eliminar mediante el comando `kill` el proceso hijo.

3. Lanzar los mismos procesos, eliminando en esta ocasión el proceso padre, también mediante el comando `kill`. Tomar nota de los resultados obtenidos.

4. Ejecutar el código correspondiente al padre infinito que crea un proceso hijo finito, de manera que éste último quedará en estado *zombie*. Observar si realmente se produce esta circunstancia.

5. Tratar de eliminar el proceso hijo – proceso en estado *zombie* – anotando los resultados percibidos.

6. Con el mismo ejemplo, eliminar el proceso padre, tomando nota de los resultados que se obtengan.

Tareas pendientes

1. Capturar señales haciendo uso de la llamada al sistema *signal*, accesible a través de la función *signal(señal, puntero a función)*, probando además distintas ubicaciones de la misma. La función que se “active” en el momento de la recepción de la señal por parte del proceso, deberá mostrar un mensaje por pantalla que deje constancia de que la señal se ha recibido.

Tomar para ello, cualquiera de los dos ficheros de código anteriormente referenciados, e incluir una función *signal()* antes de la función *fork()*, referida por ejemplo a la señal SIGTERM.

2. A continuación, matar primeramente al proceso padre y posteriormente al proceso hijo, tomando nota de los resultados obtenidos.

3. Intentar eliminar el proceso hijo infinito haciéndole llegar una señal distinta de la capturada. Comprobar si el proceso ha sido eliminado.

4. Someter a ejecución el mismo código y tratar de hacer que el proceso padre pase a ser hijo del proceso *init*. ¿Cómo sería factible este tipo de operación? ¿Qué proceso debería ser eliminado del sistema?



Práctica 7: Llamadas relacionadas con la E/S estándares.

Objetivos

- Comprender la forma que tiene el sistema operativo Linux de representar y hacer uso de los dispositivos de E/S estándar.
- Adquirir la capacidad de trabajar desde línea de comandos y mediante programas escritos en lenguaje C con la E/S estándar.

Requisitos

- Leer y comprender la parte del capítulo 6 de los apuntes de la asignatura en la que se hace referencia a las llamadas del sistema operativo relacionadas con el manejo de ficheros.
- Leer y comprender el apartado 6.3 de la Guía del Usuario Linux en el que se habla de la entrada estándar y la salida estándar.
- Comprender cómo es posible el paso de información entre comandos de Linux haciendo uso de *pipes*.
- Recordar el manejo de los ficheros y sus descriptores mediante programas escritos en el lenguaje de programación C.

Desarrollo de la Práctica

1. Redireccionamiento de la salida de los comandos de Linux.

- a- Escribir y probar la secuencia de comandos necesaria para redireccionar al fichero list.ord el listado del directorio /bin ordenado por orden alfabético.

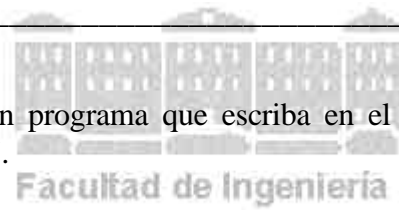
- b- Escribir y probar la secuencia de comandos necesaria saber el número de usuarios que hay conectados a la máquina en el momento en el que se ejecuta esta secuencia de comandos. Consultar en el manual el uso de los comandos `who` y `wc`.

- c- Escribir y probar la secuencia de comandos necesaria para mostrar, únicamente, el contenido de la variable `PATH` del sistema.

- d- Escribir y probar la secuencia de comandos necesaria para almacenar en el fichero *dev.lst* el listado del directorio */dev*.

- e- Escribir y probar la secuencia de comandos necesaria para almacenar el contenido ordenado del fichero *dev.lst* en el fichero *dev.lst.ord*.

2. Desarrollar y ejecutar un programa que escriba en el fichero *tabla7.dat* la tabla de multiplicar del número 7.



3. Desarrollar y ejecutar el mismo programa, pero utilizando la instrucción `printf` para escribir en el fichero. Para ello, es necesario hacer que el fichero de salida, tenga el descriptor de `stdout` (salida estándar). Explicar cómo se ha conseguido asignar el descriptor de fichero que inicialmente tenía `stdout` a `tabla7.dat`.



4. Desarrollar y ejecutar otro programa que lea el contenido de cualquier fichero y los muestre por pantalla. Si no se introduce el nombre de ningún fichero en línea de comandos, se mostrará por pantalla el contenido del fichero `tabla7.dat`.

5. Desarrollar y ejecutar un programa que lea el contenido del fichero `tabla7.dat`, utilizando la instrucción `scanf` y lo muestre por la pantalla. Explicar, cómo se ha conseguido asignar el descriptor del fichero que inicialmente tenía `stdin` a `tabla7.dat`.



Práctica 8: Manejo de “pipes” y generación de múltiples hijos.

Objetivos

- Poner en práctica los conceptos relativos a la gestión de procesos vistos en teoría y prácticas anteriores
- Familiarizarse y saber utilizar los *pipes* - mecanismo de intercambio de información disponible en la mayoría de sistemas *Unix-like*.

Requisitos

- Asimilar el funcionamiento de las llamadas *fork*, *exit* y *wait*, manipuladas durante la práctica anterior.
- Leer con aprovechamiento las páginas del capítulo 6 de los apuntes de la asignatura relativas a la creación e intercambio de información a través de *pipes*; éstos no son sino elementos semejantes a ficheros temporales, cuya manipulación puede llevarse a cabo por tanto como si de ficheros se tratara.
- Lectura de las páginas 17 a 29 de la Guía Linux de Programación en las que se hace referencia al mecanismo de comunicación entre procesos en Linux mediante *pipes*.

Desarrollo de la práctica

1. Explicar mediante un gráfico las situaciones inicial y final del sistema, haciendo especial énfasis en los descriptores de fichero de los dos procesos siguientes:

```

pipe (a);
if (fork()!=0)          /* padre- escribe en el 'pipe' */
{
    close (a[0]);      /* cierra descriptor de lectura del 'pipe' */
    close (1);         /* cierra la salida estándar para reasignarla al 'pipe' */
    dup (a[1]);        /* crea un nuevo descriptor para la escritura en el
                        'pipe', le será reasignado el número 1, ya que es el
                        más bajo disponible, por esto lo cerramos antes */
    close (a[1]);      /* cierra el descriptor original de escritura del
                        'pipe' */
    execv (prog1, argv, NIL_PTR);
}
else
{
    /* hijo – ejecuta el proceso que lee del 'pipe' */
    /* realiza las mismas operaciones que el padre
    para conseguir reasignar la entrada estándar al
    escritor de lectura del 'pipe' */

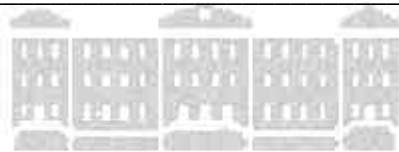
    close (a[1]);
    close (0);
    dup (a[0]);
    close (a[0]);
    exec (prog2, argv, NIL_PTR);
}
    
```



2. ¿Es imprescindible la creación del *pipe* antes de que el proceso padre ejecute la *fork()* con la que se crea el proceso hijo?

3. Creación de dos procesos. ¿Dónde hay que escribir el código de cada uno? Dibujar mediante un organigrama en el que se detalle la ubicación de las sentencias *fork* dónde escribir el código de cada proceso, de manera que se tenga un proceso padre que cree dos hijos, y no un proceso padre que crea un proceso hijo, que a su vez crea otro.

4. ¿Cómo se las arregla el proceso padre para esperar a que terminen sus procesos hijo?
¿Puede ejecutar dos *wait* seguidas – una sobre cada *pid* de los hijos – o podría causar esto una situación de postergación indefinida? ¿Cuál es el código correcto para llevar a cabo esta comprobación?



Tareas pendientes

1. Escribir un programa completo en el que un proceso padre que cree dos procesos hijo. Codificar los códigos de los hijos de manera que uno escriba mensajes en el *pipe* y el otro los vaya leyendo y mostrando por pantalla. Hacer uso de las llamadas *read* y *write*, tanto redireccionando la salida / entrada estándar de cada hijo, como sin hacerlo. Usar, una vez redireccionadas, sentencias de lectura y escritura en la E/S estándar, como *printf*, *scanf*, *puts* o *gets*.



Práctica 9: Aplicación de lo aprendido a un caso real

Objetivos

- Implementar procesos parte de un caso real; es decir, utilizar todo lo aprendido a un entorno industrial, en el cual la informática constituya el medio para dar solución a problemas.

Requisitos

- Recordar los conceptos involucrados en los sistemas de control

Un sistema de control es un conjunto de elementos que interactúan entre sí para mantener a cierto sistema dentro un comportamiento marcado por una serie de parámetros. El elemento “sensor” es el encargado de tomar mediciones periódicas que permitan representar el estado o alguna característica de un sistema. Por ejemplo un termómetro en un climatizador de un vehículo.

El sistema procesará las informaciones recibidas, de manera que en el momento en que detecte una variación, tratará de subsanarla por medio de un “actuador”. Este elemento podría ser, por ejemplo, un ventilador que se activaría automáticamente para bajar la temperatura al valor de referencia cada vez que se detecten desviaciones al alza.
- Tener claros los aspectos teóricos relacionados con la manipulación de jerarquía de procesos (creación, eliminación de procesos, etc.), los relacionados con la manipulación de señales y el manejo de *pipes*.
- Realizar un modelo de comportamiento de un software de monitorización de una variable registrada con un sensor. El dato leído servirá para realizar un control proporcional y enviar una señal de corrección a un elemento actuador. Utilizar la notación Yourdon DFD con extensión para sistemas en Tiempo Real.

Desarrollo de la práctica

Simular mediante un generador de números aleatorios, la existencia de un sensor en el sistema. Este “sensor” será un proceso dependiente del proceso principal. El “actuador” se implementará en base a un fichero de texto que irá recogiendo los valores a enviar al actuador.

El sistema constará de dos procesos, el “sensor” y el “actuador”. El primero creará al segundo, de manera que ambos estén comunicados a través de un *pipe*.

El proceso sensor invocará cada 10 segundos a una función denominada “leer_sensor()” que será la encargada de generar un número aleatorio entero que hará las veces del dato medido. Este valor será enviado al proceso actuador a través de un *pipe*.

1. Mostrar mediante un organigrama el funcionamiento del sistema.



2. Mostrar en un esquema el funcionamiento del proceso “actuador”, que recibe datos enviados por el proceso “sensor”. Para cada lectura, llama a la rutina “regula()”, que devolverá un “1” si el valor de la lectura es mayor que cierta constante, o un “-1” si el valor es menor que dicha constante. A continuación este proceso enviará, mediante una única operación de escritura, a un fichero denominado “registro.dat” el valor recibido a través del *pipe* y el devuelto por la función “regula()”.

Tareas pendientes

1. Codificar estos dos procesos, siguiendo los organigramas descritos anteriormente. Tener en cuenta que el fichero *registro.dat* será un histórico en el que se vaya registrando a lo largo del tiempo, la lectura obtenida por el sensor, y el resultado generado por el actuador (en vez de que cada escritura en el citado fichero *machaque* el contenido de éste).

