UNIVERSIDAD AUTÓNOMA DE BAJA CALIFORNIA
FACULTAD DE INGENIERÍA (UNIDAD MEXICALI)
DOCUMENTO DEL SISTEMA DE CALIDAD

Formatos para prácticas de laboratorio

CARRERA	PLAN DE ESTUDIO	CLAVE ASIGNATURA	NOMBRE DE LA ASIGNATURA
IC	2003-1	5314	Sistemas Operativos

PRÁCTICA No.	LABORATORIO DE		DURACIÓN (HORA)
2	NOMBRE DE LA PRÁCTICA	Introducción al Sistema Operativo Linux	2

1. INTRODUCCIÓN

En esta práctica se recordará el procedimiento de enlace al servidor tiburón, para posteriormente realizar un acceso a tu cuenta y practicar con algunos mandos básicos del shell de Linux, finalmente terminarás con tu sesión de trabajo correctamente.

2. OBJETIVO (COMPETENCIA)

El alumno comprenderá y practicará con algunos mandos básicos del shell de Linux.

3. FUNDAMENTO

Qué es un sistema operativo?

El sistema operativo es el software que se extiende al hardware cuya función principal es la buena administración de los recursos computacionales a la vez de proporcionarle una interna la usuario.

Conceptos Básicos

Shell. Es el intérprete de comandos y es también la interfaz primaria entre un usuario sentado ante su terminal y el sistema operativo. Cuando un usuario ingresa en el sistema, se inicia un shell. El shell tiene la terminal como entrada estándar y salida estándar, y lo primero que hace es exhibir el prompt, un carácter de signo de dólar, que le indica al usuario que el shell esta esperando para aceptar un mando.

Kernel. Es el programa que tiene acceso directo a hardware (solo una parte en ensamblador), también se encarga de coordinar los tiempos para cada usuario. Entre las funciones principales del KERNEL tenemos:

Manejar la memoria de la computadora
Controlar el acceso a la computadora
Mantener el sistema de archivos
Manejar interrupciones (señal para terminar ejecución)
Manejar errores
Realizar servicios de entrada y salida
Asignar los recursos de la computadora

Diferencias entre LINUX, DOS/WINDOWS y WINDOWS NT

Dos es mono-usuario.
Dos no es multitareas.
Dos no tiene la gran cantidad de herramientas que tiene LINUX.
Dos no está orientado a redes ni a comunicaciones.

WINDOWS NT vs LINUX

Son sistemas operativos de 32 bits.
Ambos son multitareas. NT solo soporta un usuario a la vez.
WINDOWS NT solo se apega parcialmente a poxis.
Ambos corren en diferentes plataformas.
Existen más aplicaciones para LINUX que para NT.
NT está orientado a eventos; LINUX esta orientado a procesos.
NT no es un sistema operativo abierto.

Además podemos decir que LINUX

Es gratis.
Proporciona ambiente multitareas.
Contiene ambiente gráfico (windows).
Apoyo integrado para red.
Compatibilidad con el estándar IEEE de posix.1.
Código fuente no propietario (no le pertenece a nadie).
Soporte de software GNU.
Ocupa 250 Mb, de 12 a 16 en RAM, y corre de 386 para arriba.

Mandos para el manejo de directorios

En el manejo de directorios es de gran utilidad lo siguiente:

. Directoria actual
.. Directorio padre
/ Directorio raíz
~ la tilde es un caracter especial que representa el nombre de ruta del directorio propio (home).
Resulta ser muy útil cuando necesita usar el nombre ruta absoluta de un archivo o de un directorio.

NOTA: el shell de LINUX posee los siguientes comodines: * , ? , [ ]

* Equivale a cualquier conjunto de caracteres de nombre de archivo.
? Coincide con un solo carácter cualquiera de nombre de archivo
[ ] Coincide con una clase de posibles caracteres de nombre de archivo.

Ya familiarizados un poco con lo que es LINUX, veamos ahora algunos mandos importantes que tiene LINUX para el manejo de directorios. Estos mandos los podemos encontrar en la tabla 1.

Comando en LINUX	Comando en DOS	FUNCION
ls	Dir	Despliega el contenido del directorio
mkdir	Md	Crear un directorio nuevo
rmdir	Rd	Borrar un directorio existente
pwd		Directorio actual de trabajo
mv	Ren	Renombra un directorio
cd	Cd	Cambio de directorio

Tabla 1

Modalidades de las ordenes para el manejo de directorios

cd

cd ..↵ Regresa al directorio padre.
cd <directorio> ↵ Cambia el directorio de trabajo al directorio especificado.

ls

ls -F ↵ Antepone un símbolo al nombre cada archivo, para indicar que tipo de archivo es.
ls -R ↵ Lista el directorio de trabajo así como también todos los subdirectorios.
ls -l ↵ Listado alfabético (a-z).
ls -a ↵ Muestra los archivos ocultos.
ls -r ↵ Muestra el contenido de un directorio en sentido inverso.

mkdir

mkdir <nuevo directorio> ↵ Crea un nuevo directorio en el directorio actual.

mv

mv <nombre anterior> <nombre nuevo> ↵ Renombra el archivo al con el nombre nuevo.

rmdir

rmdir <directorio> ↵ Borra el directorio especificado, el cual debera estar vacío.

pwd

pwd ↵ Muestra el directorio de trabajo actual.

Mandos para la administración de archivos

Un archivo es la unidad fundamental de un sistema de archivos en Linux. Existen tres diferentes tipos de archivos en Linux: Archivos ordinarios, archivos de dispositivo y archivos de directorio.

Archivos ordinarios: Un archivo ordinario es una colección de bytes. Generalmente son documentos, códigos fuente de programas, o datos de programas. Los archivos binarios ejecutables, programas se consideran también archivos ordinarios. Los bytes de un archivo ordinario se interpretan como caracteres texto, instrucciones binarias, o cláusulas de programas, por los programas que los examinan.

Cada archivo ordinario contiene los siguientes atributos:

- Un nombre de archivo (no necesariamente único).
- Un número de filesystem único, conocido como i-node.
- Un tamaño en bytes.
- La hora de última modificación.
- Un juego de permisos de acceso.
- Un dueño y un grupo.

Archivos de Dispositivos: Cada dispositivo físico en el sistema, tales como un disco duro, disco flexible, impresores, terminales y el sistema de memoria tienen asignados un “archivo especial.” Estos archivos son llamados “archivos de dispositivos”.

Archivos de Directorios: Los archivos de directorios son los lugares donde los archivos son almacenados (conceptualmente, no físicamente). Un archivo de directorio es referido como un “directorio” y contiene los nombres y la localización de los archivos “que están en el”.

Protección de archivos: En los sistemas multiusuarios, es necesario proteger ciertos archivos, negando y permitiendo a la vez el acceso a ellos de algunos usuarios. Los archivos son protegidos asignándoles “permisos de acceso” (access permissions). Linux provee 3 niveles de permisos de acceso:

- **read (lectura)** Los permisos de lectura en un archivo, permiten al usuario ver el contenido del archivo. Un usuario con permiso de solo lectura (read-only), no puede editar un archivo.
- **write (escritura)** Con este permiso, un usuario puede editar un archivo.
- **execute (ejecución)** Si el archivo es un programa, teniendo los permisos de ejecución, permite al usuario correrlo. Los permisos de acceso son asignados por el dueño del archivo. Cualquier combinación de los 3 niveles es permitida. Esto posibilita al dueño del archivo a determinar que usuarios pueden leer, escribir y/o ejecutar el archivo. Linux permite separar los permisos de acceso para el dueño de un archivo, para el grupo del archivo y para los demás usuarios. Generalmente, el dueño del archivo debe tener los permisos de lectura y escritura, el grupo solo los permisos de lectura y los demás usuarios con ningún permiso.

Nombres de archivos: Un nombre de archivo es una secuencia de 1 a 14 caracteres consistente de letras, dígitos y caracteres especiales. Los nombres de archivo deben indicar el contenido de los mismos. Estos nombres deben ser únicos en el directorio y pueden repetirse en todo el sistema. Directorios diferentes pueden contener diferentes archivos con el mismo nombre. Cuando un nombre de archivo contiene un punto al inicio (.), es un archivo "oculto." Los archivos de configuración del sistema por lo general son archivos ocultos. Los caracteres como el signo de interrogación (?), el asterisco (*), corchetes ([,]), guión (-) nunca deben usarse para nombrar archivos porque tienen un significado especial para el shell.

Nombres de ruta: Un nombre de ruta es una secuencia de nombres de directorios seguidos por un nombre de archivo, separados del nombre previo por una diagonal (slash /). Si una ruta comienza con una diagonal, se especifica que el archivo puede encontrarse comenzando una búsqueda desde la raíz de la estructura de árbol. De otra manera, los archivos serán encontrados a partir del directorio actual donde se encuentre el usuario. Una ruta que empieza con una diagonal es llamada *nombre de ruta completa o absoluta*. las rutas absolutas son un mapa de localización de archivos y son únicas en el sistema. Cuando una ruta no empieza con una diagonal, se le llama *ruta relativa,* ya que es relativa al directorio actual donde se encuentre el usuario.

Ejemplo de ruta relativa
./datos/fuente.txt =>busca en el directorio actual

Ejemplo de ruta absoluta
/datos/fuente.txt => busca desde la raíz

En la tabla 2 se te presenta un resumen de las principales órdenes o mandos en Linux para la administración de archivos y su equivalente para el sistema operativo MS-DOS.

Comando en LINUX	Comando en DOS	FUNCION
cat	Type	Despliega el contenido de un archivo, Crea un archivo.
more	Type-More	Muestra el contenido de un archivo haciendo pausas.
touch		Cambia la fecha y hora de creacion, modificación o última lectura de un archivo.
mv	Move	Renombra un directorio.
cp	Copy	Copiar archivos.
grep		Encuentra las líneas de un archivo que contienen un patrón determinado.
cut		Seleccionar campos o columnas de archivos.
sort		Ordena las líneas de un archivo de texto.
wc		Muestra la cantidad de palabras, caracteres y líneas que tiene un archivo.
tail		Muestra las últimas n líneas de un archivo.
head		Muestra las primeras n líneas de un archivo.
chmod	Attrib	Permite cambiar los permisos de un archivo.

Tabla 2

Cada uno de los mandos anteriores puede funcionar bajo diferentes modalidades, por lo tanto se te presenta a continuación algunas de ellas.

cat

cat <archivo> ↵ Lista el contenido de un archivo, no hace pausa.
cat ↵ Sin argumento permite escribir como si fuera un editor, pero no graba al salir.
cat -n <archivo>↵ Muestra el contenido del archivo, numerando cada línea.
cat -A <archivo>↵ Muestra los caracteres de control como caracteres visibles.

sort

sort -f <archivo>↵ Ordena considerando de igual valor mayúsculas y minúsculas.
sort -M <archivo>↵Busca las cadenas "Jan", "Feb", … "Dec", y ordena según el calendario.
sort -n <archivo> ↵Ordena las líneas por número en forma ascendente.
sort +1 <archivo>↵Ordena por la segunda columna.

more

more <archivo1, archivo2,...,archivon> Despliega el contenido de uno o más archivos texto.

Donde puede utilizar:

- La Barra espaciadora para cambiar a la siguiente pantalla.
- <Enter> para avanzar línea por línea.
- n Para avanzar al siguiente archivo
- q para salir de more

more -l<archivo> ↵Ignora el carácter formfeed.
more [+inicio] [-líneas] <archivo>↵ Muestra n líneas de un archivo a partir de la línea de inicio.

Donde:

inicio: Número de línea de inicio.
líneas: Número de líneas por pantalla.

touch

touch [-t MMDDhhmm[[CC]YY][.ss]] <archivo>↵ Cambia la fecha y hora del archivo.

Donde:

MM=mes DD=días hh=horas mm=minutos.
Parámetros opcionales: CC=siglos YY=años ss=segundos.

mv

mv -i <nombre viejo> <nombre nuevo>↵ Renombrar archivo.
mv -i <destino1 destino2 ... destinon> <destino>↵ Mueve el (los) archivo(s) hacia el directorio destino.

Donde:

-i es el argumento para verificar si ya existe el archivo y pregunta antes de sobre escribirlo.

cp

cp -i <fuente> <destino>↵ Copia el archivo fuente al archivo destino
cp -i <fuente1 fuente2 fuente3... fuente N> <directorio destino> Copia los archivos hacia un directorio.

Donde:

-i es el argumento para verificar si ya existe el archivo y pregunta antes de sobre escribirlo.

grep

grep <cadena> <archivo>↵ Muestra la línea(s) donde encuentra la cadena
grep -n <cadena> <archivo>↵ Muestra la línea y el número de línea en donde encuentra la cadena.
grep -c <cadena> <archivo>↵ Muestra cuántas líneas contienen el patrón especificado.
grep -v <cadena> <archivo>↵ Muestra las líneas que no cumplen con el patrón de búsqueda.
grep -w <cadena> <archivo>↵ Muestra las líneas que contienen la cadena como palabra completa.
grep -x <'frase'> <archivo>↵ Muestra la línea donde se encuentra la frase completa.
grep -i <cadena><archivo>↵ Evita la distinción entre mayúsculas y minúsculas.

tail

tail <archivo>↵ Muestra las últimas 10 líneas del archivo.
tail -n <archivo>↵ Muestra las n últimas líneas del archivo.

head

head <archivo>↵ Muestra las primeras 10 líneas del archivo.
head -n <archivo>↵Muestra las n primeras líneas del archivo.

cut

cut -f<numero> <archivo>↵ Selecciona sólo el campo con el número especificado.
cut -f<inicio-fin> <archivo> ↵ Selecciona el rango de columnas desde inicio a fin.
cut -f<col1,col2,coln> <archivo>↵ Selecciona sólo las columnas especificadas en la lista.
cut -f -d'caracter' -f<col> <archivo>↵Muestra la columna indicada, considerando como delimitador entre columnas el caracter especificado en caracter.

wc

wc <archivo>↵	Muestra cuantos caracteres, líneas y palabras tiene el archivo.
wc -c <archivo>↵	Muestra el total de caracteres que tiene el archivo.
wc -w <archivo>↵	Muestra el total de palabras que tiene el archivo.
wc -l <archivo>↵	Muestra el total de líneas que tiene el archivo.

Permisos de archivos y directorios

Los sistemas UNIX y Linux permiten al dueño de un archivo o directorio restringir el acceso a ellos. Los permisos en un archivo limitan la lectura, escritura y/o ejecución, mientras que para un directorio limitan a quien pudiera cambiarse a ese directorio, listar su contenido asi como crear y borrar archivos dentro del mismo. Con el comando **ls -l** se despliegan la descripción y permisos de los archivos distribuidos de la siguiente manera:

El primer caracter indica el tipo de archivo.

-	Indica un archivo ordinario.
b	Indica un dispositivo especial de bloque como un disco duro o flexible.
c	Indica un dispositivo especial de caracter como una línea de impresión o una terminal serial.
d	Indica un directorio.
m	Indica un archivo de compartimiento de datos.
n	Indica un archivo de nombre especial.
p	Indica una pipa nombrada.
s	Indica un semáforo.

De izquierda a derecha, los siguientes 9 caracteres se interpretan como los juegos de los 3 permisos. Cada juego indica los siguientes permisos:

Permisos del dueño.
Permisos del grupo.
Permisos de los otros usuarios.

Cada juego contiene los permisos de lectura, escritura y ejecución del archivo como comando. Para un directorio, el permiso de “ejecución” significa que se permite buscar en el directorio cualquier archivo o subdirectorio.

Para los archivos ordinarios, los permisos tienen el siguiente significado:

r	El archivo puede leerse.
w	Se puede escribir en el archivo .
x	El archivo puede ejecutarse.
-	El permiso no esta otorgado o nulo.

Para los directorios, los permisos tienen el siguiente significado:

r	Los archivos pueden listarse, el directorio debe tener el permiso “x”.
w	Pueden crearse o borrarse archivos en el directorio.
x	Puede buscarse en el directorio.

Cambio de los permisos de archivos

El comando **chmod** cambia los permisos de lectura, escritura y ejecución y busca permisos en un archivo o directorio. La sintaxis es la siguiente:

chmod <modo><archivo>↵

Hay dos métodos para usar el comando chmod, uno de ellos se vale de números y se llama método absoluto; el otro utiliza símbolos y recibe el nombre de método simbólico.

Método Absoluto: Este método, también conocido como numérico, usa un número octal de tres dígitos para almacenar los permisos.

Mascara de bits octal	Significado/permiso
1	Ejecución
2	Escritura
4	Lectura

Para encontrar los dígitos octales que necesita para especificar los permisos para una cierta categoría (usuario, grupo u otros), basta con sumar los números que estén asociados con los permisos que desee activar. Por ejemplo: Si se desea activar todos los permisos para el usuario, solo lectura y ejecución para el grupo y ninguno para otros usuarios, la asignación sería:

Primer juego de permisos (dueño): 4 + 2 +1 =7
Segundo juego de permisos (grupo): 4 + 1 =5
Tercer juego de permisos(otros): 0

El comando chmod para otorgar estos permisos se escribiría: **chmod** 750 <archivo>↵

Método simbólico. Utiliza símbolos para establecer categorías y permisos

Los usuarios se especifican como sigue:

- **u** Usuario, el dueño de un archivo o directorio.
- **g** Grupo, el grupo de usuarios al cual el dueño del archivo pertenece.
- **o** Otros, todos los usuarios del sistema que no están en u o g.
- **a** Todos los usuarios del sistema.

Los permisos se especifican como sigue:

- **r** Lectura, el cual permite a los usuarios visualizar pero no modificar o borrar el archivo.
- **w** Escritura, el cual permite a los usuarios modificar o borrar el archivo.
- **x** Ejecución, que permite a los usuarios ejecutar el archivo como comando.

Ejemplos de cambio de permisos:

chmod u+x <archivo>↵	Otorga el permiso de ejecución al dueño del archivo.
chmod go+x <archivo>↵	Otorga al grupo y a otros usuarios el permiso de ejecución sobre el archivo.
chmod o-w <archivo>↵	Prohíbe a otros usuarios la escritura en el archivo.
chmod o+r-wx <archivo>↵	Otorga permiso de lectura para otros usuarios. Suprimen los permisos de escritura y de ejecución.

Se utiliza la misma forma para cambiar permisos en directorios.

Direccionamiento de entrada y salida de datos

Linux tiene la capacidad de direccionar la información que le permite mover datos fácilmente hacia dentro y hacia afuera de archivos. Puede direccionar la salida estándar de manera que, en lugar de verla por pantalla, quede guardada en un archivo. También puede direccionar la entrada estándar, quitándola del teclado y asignándola a un archivo de manera que la entrada se lea desde el archivo en lugar desde el teclado.

Cuando se ejecuta una orden de Linux que genera una salida, ésta se pone en el flujo de salida de datos estándar. El destino por default del flujo de salida es un dispositivo, en este caso, la pantalla. Los dispositivos tales como el teclado y la pantalla son tratados como archivos. La pantalla es un dispositivo que visualiza un flujo continuo de bytes. Por default, la salida estándar enviará sus datos al dispositivo pantalla, que entonces visualizará los datos.

Direccionamiento de la salida estándar

Si quisiera desviar la salida estándar a un archivo en lugar de a la pantalla. Se pone el operador de redirección de salida (signo mayor que) y el nombre de un archivo en la línea de órdenes. La operación de redirección crea el nuevo archivo de destino. Si el archivo ya existe, su contenido será reemplazado por los datos de la salida estándar.

Aunque el operador de dirección y el nombre de archivo se colocan detrás de la orden, la operación de redirección no se ejecuta después de la orden, sino que se ejecuta antes que la orden. La operación de redirección crea el archivo y se establece la redirección antes de recibir ningún dato de la salida estándar. Si el archivo ya existe, será destruido y reemplazado por un archivo con el mismo nombre. En efecto, la orden que genera la salida es ejecutada sólo después de que el archivo de redirección haya sido creado.

Adición de la salida estándar. También puede agregar la salida estándar a un archivo existente usando el operador de redirección >>. En lugar de sobre escribir el archivo, los datos de la salida estándar se añaden al final del mismo.

Redirección		Ejecución
Orden > archivo	nombre	Direccionar la salida estándar a un archivo o dispositivo, creando el archivo si no existe y sobre escribiéndolo si ya existe
Orden >> archivo	nombre	Direccionar la salida estándar a un archivo o dispositivo, añadiendo la salida al final del archivo.

Ejemplos:

Ls -l > lista	Ejecuta el comando ls y envía el resultado al archivo lista.
finger >> usuarios	Ejecuta el comando finger y agrega el resultado al final del archivo usuarios.

Redirección de la entrada estándar

Muchos comandos necesitan recibir datos de la entrada de datos estándar. La entrada estándar recibe los datos de un dispositivo o de un archivo, por default la entrada de datos estándar es el teclado, los caracteres que son introducidos desde el teclado son colocados en la entrada estándar, que es entonces dirigida al comando de Linux. De manera similar a lo que sucede con la salida estándar, también se puede redirigir la entrada estándar, para recibirse desde un archivo en lugar desde el teclado. El operador para redirigir la entrada estándar es el signo menor que (<).

Redirección		Ejecución
Orden < archivo	nombre	Direccionar la entrada estándar desde un archivo o dispositivo hacia un comando.

Ejemplos:

```
cat < lista
cut -f1 < telnos
```

tr

Uno de los comandos que requiere de direccionamiento de entrada es **tr**, este es un comando que da salida a un "traducción" de la entrada, en la cual los caracteres pueden ser eliminados o sustituidos por otros.

Modalidades de tr

- tr "a" "A"< archivo — Sustituye el caracter a minúscula por A mayúscula.
- tr "[a-z]" "[A-Z]" < archivo — Especifica un rango de caracteres (a-z) que son sustituidos por otro rango de caracteres (A-Z), en este caso sustituye minúsculas por mayúsculas.
- tr -d "abc" — Suprime cualquier caracter dentro de la lista.
- tr -c "abc" "xyz" <archivo — Reemplaza aquellos caracteres que no están en la lista de caracteres.
- tr -s "." "\012" <archivo — Elimina caracteres repetidos. En este caso uno o más puntos serán sustituídos por un enter.

Puede combinar las operaciones de redirección tanto para la entrada como para la salida estándar. Observe el siguiente ejemplo, normalmente **cut** recibe entrada desde la entrada estándar y envía la salida a la salida estándar; pero en el ejemplo recibe la entrada desde un archivo y dirige la salida hacia otro archivo.

Ejemplo:

```
cut -d':' -f2 < passwd >salida
```

Entubamientos o canalizaciones

A menudo se encontrará con situaciones en las que tendrá la necesidad de enviar datos de una orden a otra y no a un archivo destino; es decir, necesitará enviar la salida estándar de un comando hacia la entrada estándar de otro. Para formar una conexión como ésta en Linux se usa algo que se conoce como canalización o entubamiento. El operador de canalización es el carácter de barra vertical (|), conocido también como pipe. Este se coloca entre dos

órdenes para formar una conexión entre ellas, la salida estándar de la primera se convierte en la entrada estándar para la siguiente. La operación de entubamiento recibe la salida de la orden que va delante de ella y envía estos datos como en entrada a la orden detrás del tubo.

Es necesario tener presente la diferencia entre un archivo y una orden: un archivo es un medio de almacenamiento que contiene datos. Puede guardar datos en él o leer los datos que hay en él. Una orden es un programa que ejecuta instrucciones; puede leer o guardar datos en un archivo, pero no es en sí misma un archivo. Por esta razón, una operación de redirección opera sobre archivos y no sobre órdenes. La redirección puede enviar datos desde un programa a un archivo, pero no puede enviar datos de un programa a otro programa. Sólo los archivos pueden ser destino de una operación de redirección, pero no los programas.

Ejemplos:

home$ cat -n carta | more	El comando cat -n numera las líneas del archivo carta, la salida generada se muestra por pantallas con el comando more.
home$ ls -l | more ls -l	Lista el contenido del directorio actual, pero hace una pausa al llenarse cada pantalla.

Tee

Copia la entrada estándar en un archivo mientras la envía a la salida estandar. Generalmente se usa como un filtro y permite guardar la salida en un archivo al mismo tiempo que se envía a otro filtro.

Ejemplos:

sort archivo | tee archivo_nuevo	El archivo se ordena y la salida ordenada es canalizada a la Orden tee, la cual guarda la salida ordenada en el nuevo archivo
sort archivo | cat –n tee archivo_nuevo | lpr	Guarda la salida de un archivo y lo imprima
sort archivo | tee arcivo_nuevo cat –n | lpr	Guarda una versión ordenada de la lista y se imprime

Mandos de Comunicaciones

finger

Despliega información sobre los usuarios del sistema.

finger –s <user name> Despliega el login name del usuario, su nombre real, nombre de la terminal name y el status de escritura (``*'' después del nombre de terminal si el permiso de escritura está denegado), tiempo inactivo, hora de entrada, ubicación y teléfono de la oficina.

finger -l <user name> Despliega toda la información de la opción -s en forma multilínea, además muestra el home directory del usuarios, su número telefónico, login shell, el status del correo y los contenidos de los archivos .plan, .project y . fordward.

finger -l <user name> Evita que la opción -l despliegue el contenido de los archivos .plan y . project.

finger -m <user name> Si el login name de un usuario coincide con el nombre real de otro, esta opción obliga a que sólo se muestre la información del usuario que tenga el login name especificado.

Nota: Si no especifica el user name, se mostrará la información de los usuarios que estén en el sistema en ese momento.

w

Muestra quienes están en el sistema y que están haciendo.

who

Muestra la información de los usuarios que estan conectados al sistema

mail

Esta ordena permite mandar un correo, busque mediante la orden man la forma en que se puede utilizar.

4. PROCEDIMIENTO (DESCRIPCIÓN)

A) EQUIPO NECESARIO	MATERIAL DE APOYO
Computadora con sistema operativo Linux	Práctica 2 del manual de sistemas operativos Para el plan 2003-1

Antes de comenzar la práctica escucha bien las indicaciones de tu maestra

1. Accesar a linux.
2. Introduce tu login y palabra clave.
3. Despliegue el nombre del directorio de trabajo actual.
4. Lista en orden alfabético todos los archivos (incluyendo los ocultos) de tu directorio.
5. Lista en orden alfabético el contenido de tu **home directory** mostrando sus permisos, dueño, fecha de última modificación etc.
6. Desarrollar la estructura de directorios que se indique en el pizarrón.
7. Verifique que la estructura haya sido creada correctamente.
8. Borra el último nivel del árbol de directorios.
9. Dentro de un directorio llamado apodos, crea un directorio para cada alumno del salón, asígnale como
10. nombre el apodo de la persona (puedes verificar la lista de usuarios mediante la orden **who**).
11. Investigar mediante la orden **man** (ayuda) como mostrar el listado en forma descendente (z-a).
12. Renombra todos los directorios del directorio apodos con los nombre de pila de tus compañeros.
13. Crea un directorio llamado refranes.
14. Copia el archivo refran1 (pregunta a tu maestra(o) donde se encuentra localizado dicho archivo) .
15. De la misma forma, copia los archivos refran2 y luego refran3.
16. Borrar los tres archivos que copiaste en los pasos anteriores.
17. Ahora copia los tres archivos utilizando una sola instrucción.
18. Muestra en el monitor el contenido de refran1.
19. De la misma forma muestra en pantalla el contenido de refran2 y refran3.
20. Muestra el contenido de refran3 en el monitor, numerando cada línea.
21. Muestra el contenido de los tres archivos de refranes al mismo tiempo.

22. Copia el archivo datos que se encuentra en el directorio So200 a tu directorio.
23. Muestra en contenido del archivo datos en el monitor.
24. Muestra ahora el contenido del archivo datos en el monitor página por página.
25. Muestra las últimas diez líneas del archivo datos.
26. Efectúa una ordenación del archivo datos por apellido paterno de la A a la Z.
27. Ordena el archivo ahora por apellido materno de la A a la Z.
28. Ordena el archivo por apellido materno de la Z a la A (investigar con la orden man la ordenación descendente).
29. El archivo datos ahora se llamara alumnos.
30. Muestra las últimas quince líneas del archivo alumnos.
31. Muestre el número de línea y la línea donde se encuentra Lopez en el archivo datos.
32. Muestre en pantalla sólo la primer columna del archivo alumnos.
33. Muestre ahora la segunda columna del archivo alumnos.
34. Muestre la última y penúltima columna del archivo alumnos.
35. Muestre todas las líneas del archivo datos, excepto las personas cuyo apellido comienzan con la letra **L.**
36. Liste todas las personas cuyo apellido comienzan con la letra **L**.
37. ¿Cuántas palabras en total contiene el archivo alumnos?.
38. ¿Cuántos caracteres en total contiene el archivo alumnos?.
39. Verifica los permisos de todos los directorios que están en el directorio operativo. Observa como están los permisos para el grupo y otros usuarios.
40. Le darás permiso a todo el grupo para que puedan leer y escribir en tu directorio.
41. Selecciona a uno de tus compañeros y escribirás un archivo en su directorio, dicho archivo se llamará **Historia** (utiliza la orden Cat > Nombre_del_archivo , esta orden se explicará en la siguiente práctica). En el archivo escribirás un texto que contenga en grandes rasgos la historia de tu vida desde tu niñez a la fecha actual.
42. Le darás permiso de lectura al grupo para este archivo.
43. Todos analizaran la historia correspondiente y la platicaran en forma sintetizada al grupo.
44. Copia la historia de tres de tus compañeros a un directorio llamado **HIS.**
45. Restringe los permisos de lectura y escritura en tu directorio y a todo el grupo.
46. Copiar el archivo **info_alumnos** (pregunta a tu maestra(o) donde se encuentra localizado).
47. Muestre las líneas y el número de líneas del archivo **info_alumnos** que contienen a **Lopez Aguirre Alicia**
48. Muestre en el monitor los datos del alumno que contiene el comentario **Conflictivo.**
49. Liste las primeras 10 líneas del archivo.
50. Muestre ahora en el monitor solo las tres últimas líneas del archivo.
51. Muestre en monitor de las 10 primeras líneas del archivo **info_alumnos** solo los alumnos que están reprobados.
52. Repita el paso anterior pero direcciona la salida a un archivo llamado **reprobados.**
53. Copie solo los apellidos del archivo **info_alumnos** a un archivo llamado **apellidos.**
54. Ahora copie solo los nombres y matrículas a un archivo llamado **alumnos.**
55. Copie ahora solo el apellido paterno y el nombre de cada alumno a un archivo llamado **paterno_nom**
56. Añada a el archivo **alumnos**, el archivo **paterno_nom.**
57. Copiar dentro de un archivo llamado Lista, la lista de archivos que contiene tu cuenta.
58. Copiar dentro del archivo **usuarios** la información de los usuarios actuales.
59. Copie en un archivo llamado **info_usuarios** el nombre completo de cada usuario del grupo y su login.

60. Copie todos los login del archivo **info_usuarios** a un archivo llamado **log.**
61. Observa quienes están conectados a tu grupo y que están haciendo.
62. Selecciona tres personas a los cuales les mandaras un **mail.**
63. Pide a tres de tus compañeros que te manden un correo.
64. Verifica que tengas los tres mail en tu lista de correos.
65. Mediante la orden **man** investigar como borrar un correo de la lista de mail.
66. Borre el correo número 1.
67. Ahora borra el correo 2.

C) CÁLCULOS Y REPORTE

5. RESULTADOS Y CONCLUSIONES

Al finalizar la práctica el alumno demostrará su habilidad para manejar los mandos ejemplificados en la sección del desarrollo de la práctica.

6. ANEXOS

7. REFERENCIAS

http://yaqui.mxl.uabc.mx/~so

UNIVERSIDAD AUTÓNOMA DE BAJA CALIFORNIA
FACULTAD DE INGENIERÍA (UNIDAD MEXICALI)
DOCUMENTO DEL SISTEMA DE CALIDAD

Formatos para prácticas de laboratorio

CARRERA	PLAN DE ESTUDIO	CLAVE ASIGNATURA	NOMBRE DE LA ASIGNATURA
IC	2003-1	5314	Sistemas Operativos

PRÁCTICA No.	LABORATORIO DE		DURACIÓN (HORA)
3	NOMBRE DE LA PRÁCTICA	Llamas al sistema exec() y system()	2

1. INTRODUCCIÓN

Las llamadas al sistema proporcionan la interfaz entre el sistema operativo y un programa en ejecución. Linux permite efectuar llamadas al sistema directamente desde un programa realizado en un lenguaje de alto nivel, en particular en lenguaje C, en cuyo caso las llamadas se asemejan a llamadas a funciones, tal y como si estuvieran definidas en una biblioteca estándar. En esta práctica trabajaremos con las funciones exec() y system() las cuales permiten realizar llamadas al sistema.

2. OBJETIVO (COMPETENCIA)

En esta práctica el alumno se familiarizará con las llamadas al sistema exec( ) y system( ) pudiendo deducir de los ejemplos la diferencia en el funcionamiento de cada una de ellas.

3. FUNDAMENTO

Cuando se ejecuta un programa se copia el ejecutable en una imagen de programa (imagen de carga) en la memoria principal y se le asigna un espacio de direcciones. Cada programa en ejecución es un proceso que es manejado por el sistema mediante una estructura de datos a la que se accede a través de un entero denominado identificador de proceso (*pid*). La familia de llamadas **exec ( )**, reemplaza la imagen del proceso actual con una nueva imagen de carga referente al archivo nombrado en el primer parámetro. En el caso más general, **exec** tiene tres parámetros: el nombre del archivo por ejecutar, un apuntador al arreglo de argumentos y un apuntador al arreglo del entorno. Típicamente, después de una bifurcación (*fork*), uno de los dos procesos emplea la llamada **exec**, para reemplazar su espacio de memoria virtual con un nuevo programa. Esta llamada carga en memoria un archivo binario (destruyendo el contenido en memoria del programa que contiene la llamada, y comienza su ejecución. Se dispone de varios procedimientos de biblioteca, *execl, execlp, execle, exect, execv, execve,* y *execp,* para que los parámetros sean omitidos o especificados de varias formas.

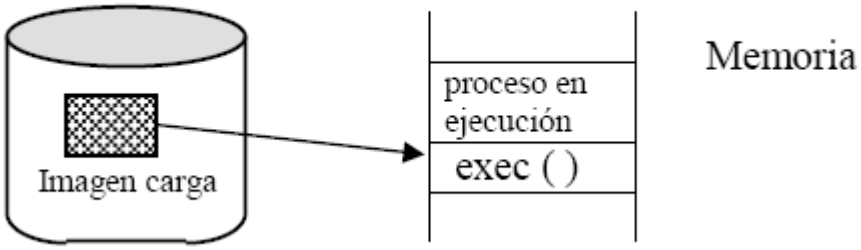


Figura 1

En la figura 1 se observa que el proceso que hace la llamada **exec**, pierde el control del procesador para ejecutar el código correspondiente al archivo cargado. Las líneas de código escritas en un programa que vienen justamente detrás de **exec** no se ejecutan, salvo si hay algún error en la carga.

Sintaxis de la función excel()

int = execl(nombre_absoluto, arg0,arg1,...,0);

Donde:
nombre_absoluto: es el *pathname* absoluto donde reside la imagen de carga del fichero a ejecutar.
arg0: es el nombre del fichero que se quiere ejecutar.

Los parámetros para el archivo que se quiere ejecutar vienen especificados desde *arg1* a *argN*, siendo *argN* el último argumento que se pasa, terminando la llamada siempre con un "0, para indicar que el argumento anterior es el último.

Programa ejemplo que muestra el uso de execl

```
/* exec.c - Listar los procesos del usuario usando exec. */
#include <stdio.h>
#include <unistd.h>

int main ()
   {
   int  salida;/* Salida del comando */

   printf ("Ejemplo de exec \n");
   execl ("/bin/ps", "ps", "-fu", getenv ("USER"), 0);
   printf ("Salida del comando: %d\n", salida);
exit (0); }
```

La salida del programa será la siguiente:

Ejemplo de exec
UID PID PPID C STIME TTY TIME CMD
lopez 29395 29393 0 Apr29 ? 00:00:00 [sshd]
lopez 29396 29395 0 Apr29 pts/1 00:00:00 -bash
lopez 5266 29396 0 12:39 pts/1 00:00:00 ps -fu lopez

system( )

La subrutina system( ) llamada a un intérprete para ejecutar una orden El proceso espera a que finalice la ejecución de la subrutina y devuelve la salida del programa ejecutado.

Sintaxis de la subrutina system

```
int = system(cadena);
```

Programa ejemplo que muestra el uso de system( )

```
/* system.c - Listar los procesos del usuario usando system. */
#include <stdio.h>
#include <stdlib.h>
int main ()
{
   int  salida;     /* Salida del comando */
   char comando[100];       /* Comando a ejecutar */
   printf ("Ejemplo de system.\n");
   sprintf (comando, "/bin/ps -fu %s", getenv ("USER"));
   salida = system (comando);
   printf ("Salida del comando: %d\n", salida);
   exit (0);
}
```

La salida del programa será la siguiente:

Ejemplo de system.
UID PID PPID C STIME TTY TIME CMD
lopez 29395 29393 0 Apr29 ? 00:00:00 [sshd]
lopez 29396 29395 0 Apr29 pts/1 00:00:00 -bash
lopez 5285 29396 0 12:42 pts/1 00:00:00 ./procesos1
lopez 5286 5285 2 12:42 pts/1 00:00:00 /bin/ps -fu lopez
Salida del comando: 0

4. PROCEDIMIENTO (DESCRIPCIÓN)

Formatos para prácticas de laboratorio

A) EQUIPO NECESARIO	MATERIAL DE APOYO
Computadora con sistema operativo Linux.	Práctica 3 del Manuel manual de Sistemas Operativos1.- Elabora un programa en C bajo Linux

utilizando la función exec() que funciones como un MINISO. Tu MINISO deberá admitir las siguientes ordenes:

Listar (ls)
Fecha (date)
Renombrar (mv)
Borrar (rm)
Limpiar(ls)

Cuando el programa inicie deberá mostrar al usuario el siguiente prompt:

MINISO]$

Cuando el usuario escriba alguna de las ordenes permitidas deberá mostrar el resultado de la ejecución de la orden y volver a mostrar el prompt hasta que el usuario escriba la palabra Salida.

Ejemplo 1:

MINISO]$ Borrar MiArchivo
MINISO]$ Salida

Ejemplo 2:

MINISO]$ Renombrar MiArchivo TuArchivo
MINISO]$ Salida

2.- Elabore el mismo MINISO planteado en el paso 1, pero ahora utilizando la función system().
3.- ¿Cuál es la diferencia entre las funciones exec() y system()?.

C) CÁLCULOS Y REPORTE

5. RESULTADOS Y CONCLUSIONES

Al finalizar la práctica el alumno conocerá la diferencia entre las llamadas a las funciones system() y exec().

6. ANEXOS

7. REFERENCIAS

http://yaqui.mxl.uabc.mx/~so

UNIVERSIDAD AUTÓNOMA DE BAJA CALIFORNIA
FACULTAD DE INGENIERÍA (UNIDAD MEXICALI)
DOCUMENTO DEL SISTEMA DE CALIDAD

Formatos para prácticas de laboratorio

CARRERA	PLAN DE ESTUDIO	CLAVE ASIGNATURA	NOMBRE DE LA ASIGNATURA
IC	2003-1	5314	Sistemas Operativos

PRÁCTICA No.	LABORATORIO DE		DURACIÓN (HORA)
4	NOMBRE DE LA PRÁCTICA	Procesos	2

1. INTRODUCCIÓN

En C bajo Linux en proceso es creado mediante la llamada al sistema fork( ). El proceso que realiza la llamada se denomina proceso padre y el proceso creado a partir de la llamada se denomina proceso hijo. En esta práctica veremos algunos ejemplos que ilustran el funcionamiento de la función fork().

2. OBJETIVO (COMPETENCIA)

En esta práctica el alumno se familiarizará con la llamada al sistema fork( ) y será capaz de comprender la diferencia en ejecución entre el los procesos padres e hijos.

3. FUNDAMENTO

En UNIX, cada vez que se ejecuta un programa se crea un proceso, es decir, un proceso es un programa en ejecución. Cada vez que el programa sea invocado se creará un proceso diferente. Los procesos a su vez pueden crear subprocesos, llamados procesos hijo (child process), un proceso puede tener varios procesos hijo, pero un proceso hijo sólo tendrá un padre. Cada proceso tiene un identificador único llamado PID. El identificador de un proceso padre se conoce como PPID.

El mando en el sistema operativo Linux que se utiliza para desplegar la información de los procesos existentes en el sistema **ps.** La forma de utilizarlo es:

ps <opciones>

Entre las opciones más comunes están (para la lista completa consultar man ps):

-A Selecciona todos los procesos.
-G Selecciona los procesos por grupo.
-U Selecciona los procesos por usuario.
-f Muestra un listado completo de información orientada al administrador.

Formuló	Revisó	Aprobó	Autorizó
MC Alicia del R. López Aguirre	MC Gloria E. Chávez Valenzuela		Dr. Maximiliano de Las Fuentes Lara
Maestro	Coordinador de Programa Educativo	Gestión de Calidad	Director de la Facultad

Procesos Foreground (en primer plano)

Cuando se teclea un mando, el shell crea un proceso hijo en el cual se ejecuta el mando. El proceso padre (el shell en este caso) se queda esperando a que se termine de ejecutar este proceso hijo. Mientras se esté ejecutando, el usuario no puede teclear otro mando, se dice entonces que el proceso hijo está ejecutándose en el foreground.

Una vez que el mando termina de ejecutarse, el proceso hijo muere, el proceso padre se reactiva, el prompt del shell aparece y entonces se puede teclear otro mando.

En resumen, un proceso que está ejecutándose en el foreground es aquél que tiene la atención total de la terminal.

Background Jobs (tareas en segundo plano)

Un proceso que se inicie desde el shell se puede ejecutar en el background como un **job** y así evitar que se bloquee la terminal. Para hacer esto solo se requiere agregar un **&** (ampersand) al final de la línea del mando.

Ejemplo:

```
[tu1316@tiburon tu1316]$ ./programa &
```

En el ejemplo anterior, *programa* fue invocado desde la línea de mando con la instrucción de que se ejecute en el background, entonces Linux inicia el proceso y retorna el PID que le asignó así como el número de job.

Para saber cuáles procesos están ejecutándose en el background utilizamos el mando *jobs*.

Ejemplo:

```
[tu1316@tiburon tu1316]$ jobs

[1]+ Running ./programa &
[tu1316@tiburon tu1316]$ fg 1
./programa
```

En el ejemplo anterior el proceso ahora es dueño de la terminal.
Para poner un proceso foreground en background es necesario realizar los siguientes pasos:

1. Suspender el proceso presionando la tecla Ctrl y la tecla z (Ctrl/z)
2. Teclear el mando bg para enviar el proceso al background.

Ejemplo:

```
[tu1316@tiburon tu1316]$ ./programa
Aquí se presiona CTRL-Z
[1]+ Stopped ./programa


[tu1316@tiburon tu1316]$ bg
[1]+ ./programa &
```

```
[tu1316@tiburon tu1316]$ jobs
[1]+ Running ./programa &
[tu1316@tiburon tu1316]$
```

kill

Este mando sirve para terminar procesos, la forma más común de utilizarlo es:

kill -9 [PID].

Ejemplo:

```
[tu1316@tiburon tu1316]$ ps
PID TTY TIME CMD
14327 pts/1 00:00:00 bash
30550 pts/1 00:21:18 programa
1469 pts/1 00:00:00 ps
[tu1316@tiburon tu1316]$ kill -9 30550
[1]+ Killed ./programa
```

La función fork()

La llamada fork() crea procesos nuevos haciendo una copia de la imagen en la memoria del padre. El nuevo proceso que se crea recibe una copia del espacio de direcciones del padre. Los dos procesos continua su ejecución en la instrucción siguiente al fork:

Sintaxis de fork()

```
#include <sys/types.h>
#include <unistd.h>
pid_t fork(void);
```

La creación de dos procesos totalmente idénticos no suele ser útil. Así el valor devuelto por fork() permite diferenciar el proceso padre del hijo, ya que *fork devuelve 0 al hijo y el ID del hijo al padre*. En caso de error devuelve -1.

En el ejemplo 1 se muestra las acciones internas de la llamada fork( )

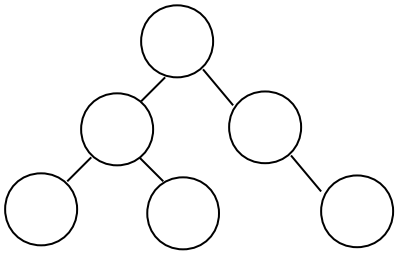
```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>

int main()
{
    pid_t rf;

    rf = fork();
    switch (rf)
    {
        case -1:
            printf ("No he podido crear el proceso hijo \n");
            break;
        case 0:
            printf ("Soy el hijo, mi PID es %d y mi PPID es
                     %d \n", getpid(), getppid());
            sleep (20);
            break;
        default:
            printf ("Soy el padre, mi PID es %d y el PID de
                    mi hijo es %d \n", getpid(), rf);
            sleep (30);
    }
    printf ("Final de ejecución de %d \n", getpid());
    exit (0);
}
```

Ejemplo 1

4. PROCEDIMIENTO (DESCRIPCIÓN)	
A) EQUIPO NECESARIO	**MATERIAL DE APOYO**

Computadora con sistema operativo Linux.

1.- Copie y compile el programa del Ejemplo 1.
2.- Ejecute el programa del Ejemplo 1 en segundo plano
3.- Verifique su ejecución con la orden ps -l
4.-Observe los valores de PID y PPID e identifique que atributos son heredados entre padre e hijo y cuales no.
5.-Anote el valor mostrado por el shell inmediatamente después de lanzar el proceso en el segundo plano e investigue que significa dicho valor.
6.-¿Cuáles son los PID del proceso padre e hijo?
7.-¿Qué tamaño de memoria ocupan los proceso padre e hijo?
8.-¿Qué proceso concluye antes su ejecución?
9.-¿Cuál es la primer instrucción que ejecuta el proceso hijo?
10.- Modifique el código del programa para que el proceso padre imprima en su mensaje de presentación (“Soy el porceso….”) antes que el proceso hijo imprima el suyo.
11.-Modifique el código del Ejemplo 1 declarando una variable entera llamada **varfork .** Coloque como valor inicial a esta variable el valor de 10. Dicha variable deberá incrementarse 10 veces en el padre y de 10 en 10, mientras que el hijo la incrementará 10 veces de uno en uno.
Anote cada una de las impresiones que realiza el programa, así como el valor final de la variable **varfork** para el padre y para el hijo y explique por que se tiene como salida dicha infomación.
12-. Escriba un programa cuyo nombre será arbolfork que permita generar el siguiente árbol de procesos



Cada proceso deberá imprimir un mensaje diferente

C) CÁLCULOS Y REPORTE

5. RESULTADOS Y CONCLUSIONES

Al finalizar la práctica el alumno demostrará su habilidad para bifurcar procesos.

6. ANEXOS

7. REFERENCIAS

http://148.231.82.20/~so

	UNIVERSIDAD AUTÓNOMA DE BAJA CALIFORNIA **FACULTAD DE INGENIERÍA (UNIDAD MEXICALI)** **DOCUMENTO DEL SISTEMA DE CALIDAD**

Formatos para prácticas de laboratorio

CARRERA	PLAN DE ESTUDIO	CLAVE ASIGNATURA	NOMBRE DE LA ASIGNATURA
IC	2003-1	5314	Sistemas Operativos

PRÁCTICA No.	LABORATORIO DE		DURACIÓN (HORA)
5	**NOMBRE DE LA PRÁCTICA**	**Comunicación entre procesos "Envío de señales"**	2

1. INTRODUCCIÓN

La comunicación entre procesos, en inglés **IPC** (*Inter-process Communication*) es una función básica de los sistemas operativos. En esta práctica se trabajará con un tipo especial de comunicación entre procesos llamada Envío de señales.

2. OBJETIVO (COMPETENCIA)

En esta práctica el alumno se familiarizará con la comunicación entre procesos, para cual enviará señales de un proceso a otro utilizando la función kill().

3. FUNDAMENTO

Aunque muchas tareas pueden realizarse en procesos aislados, la gran mayoría requieren la intervención de más de un proceso. Para que dichos procesos cooperantes lleven a buen término una tarea común, es necesario algún tipo de comunicación entre ellos. Los mecanismos de comunicación entre procesos (IPC's) habilitan mecanismo*s* para que los procesos puedan intercambiar datos y sincronizarse. A la hora de comunicar dos procesos, se consideran dos situaciones diferentes:

1. Que los procesos se estén ejecutando en una misma máquina, o bien
2. Que los procesos se ejecuten en máquinas diferentes.

En el caso de comunicación local entre procesos, aunque existen diferentes mecanismos que se engloban bajo esta denominación común, cada uno de ellos tiene su propósito específico.

Así, las señales son interrupciones software y, como tales, no deberían considerarse parte de la forma habitual de comunicar procesos y su uso debería restringirse a la comunicación de eventos o situaciones excepcionales. Para

dar solución a necesidades de comunicación más genéricas entre procesos, Unix y su adaptación para PC (Linux) utilizan diferentes soluciones que emplean como canal la memoria principal.

Una de las primeras formas de este tipo de comunicación son los *pipes* o tuberías. Los *pipes* permiten un mecanismo de comunicación unidireccional sencillo, pero también limitado. Como soluciones más elaboradas a la comunicación entre procesos se incorporan otros tres tipos de mecanismos: semáforos, memoria compartida y colas de mensajes.

Tipos de señales

Todas las señales tienen una rutina de servicio, o una acción, por defecto. Dicha acción por defecto puede ser de varios tipos:

- **A:** terminación del proceso.
- **B:** ignorar la señal.
- **C:** volcar un farchivo *core*.
- **D:** parar el proceso.
- **E:** la rutina de servicio no puede modificarse.
- **F:** la señal no puede ser ignorada.

Cada una de las señales se identifica mediante un número entero mayor que cero. En la tabla 1 se describen las señales más importantes.

Señal	Identificador	Tipo	Descripción
SIGKILL	9	AEF	Aborta la ejecución del proceso.
SIGSTOP	19	DEF	Para el proceso.
SIGCONT	18		Continúa si el proceso está parado.
SIGCHLD	17	B	Un proceso hijo ha terminado o ha parado.
SIGUSR1	10	A	Señal de usuario.
SIGUSR2	12	A	Señal de usuario.

Tabla 1 Resumen de las principales señales

Nota: Para obtener más información, se recomienda ejecutar en un terminal**: man 7 signal**

C bajo Linux proporciona una serie de funciones para llamadas al sistema que nos permiten el envío y recepción de señales, entre ellas tenemos la función kill( ). A continuación se explica como es que esta función trabaja y se muestra un ejemplo de ella.

Envío de una señal utilizando la función kill()

Cabecera:

```
#include <sys/types.h>
#include <signal.h>

int kill(int pid, int sig);
```

Comportamiento de la llamada:

kill() se utiliza para enviar una señal concreta, *sig*, a un proceso determinado que se caracteriza por su identificador, *pid*. Si se produjo cualquier error devolverá un valor **-1** y, en caso contrario, devuelve el valor **0**.

Ejemplo:

En el siguiente código se muestra un ejemplo en el que un proceso envía la señal SIGSTOP a otro proceso cuyo **pid** recibe como parámetro.

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <signal.h>

int main(int argc, char *argv[]) {

  int pid = 0;

  if (argc != 2) {
    printf(``usage: ejemplo_kill pid\n");
    exit(1);
  }
  pid = atoi(argv[1]);
  if (kill(pid, SIGSTOP) == -1) {
    printf(``\nError al enviar la señal SIGSTOP al proceso %d\n",pid);
    exit(1);
  }
  exit(0);
}
```

Formatos para prácticas de laboratorio

4. PROCEDIMIENTO (DESCRIPCIÓN)

A) EQUIPO NECESARIO	MATERIAL DE APOYO
Computadora con sistema operativo Linux.	Práctica 5 del Manuel de sistemas operativos Para el plan de estudios 2003-1

El alumno elaborará un programa que permita leer el PID de un proceso, posterior a esta lectura imprimirá el siguiente menú de opciones:

Parar proceso……. 1
Continuar proceso……. 2
Terminar proceso…… 3
Matar proceso……. 4
Salir…….. 5

Si el usuario selecciona la opción uno, dos, tres o cuatro deberá enviar la señal adecuada utilizando la función **kill()** al proceso indicado. Si el usuario selecciona la opción cinco el programa deberá terminar de la siguiente manera:

- Obtener el PID del proceso con la función **getpid( )**
- Enviarle una señal al proceso mediante la función **kill( )** para matar dicho proceso, es decir SUICIDAR AL PROCESO.

C) CÁLCULOS Y REPORTE

5. RESULTADOS Y CONCLUSIONES

Al finalizar la práctica el alumno demostrará su habilidad para comunicar procesos mediante la función kill()

6. ANEXOS

7. REFERENCIAS

http://148.231.82.20/~so

	UNIVERSIDAD AUTÓNOMA DE BAJA CALIFORNIA **FACULTAD DE INGENIERÍA (UNIDAD MEXICALI)** **DOCUMENTO DEL SISTEMA DE CALIDAD**

Formatos para prácticas de laboratorio

CARRERA	PLAN DE ESTUDIO	CLAVE ASIGNATURA	NOMBRE DE LA ASIGNATURA
IC	2003-1	5314	Sistemas Operativos

PRÁCTICA No.	LABORATORIO DE		DURACIÓN (HORA)
6	NOMBRE DE LA PRÁCTICA	**Comunicación entre procesos “Tuberías”**	2

1. INTRODUCCIÓN

Aunque muchas tareas pueden realizarse en procesos aislados, la gran mayoría requieren la intervención de más de un proceso. Para que dichos procesos cooperantes lleven a buen término una tarea común es necesario algún tipo de comunicación entre ellos. Linux ofrece la función ***pipe( )*** para abrir un canal de comunicación entre dos procesos. A lo largo de esta práctica se estudiará la forma en que la función ***pipe()*** trabaja.

2. OBJETIVO (COMPETENCIA)

El alumno aprenderá y practicará la forma de intercambiar información entre procesos diferentes, empleando los mecanismos de tuberías que proporciona el sistema operativo Linux.

3. FUNDAMENTO

El sistema operativo Linux permite que procesos diferentes intercambien información entre ellos. Para ello proporciona diferentes mecanismos que pueden utilizarse en función de las características de la comunicación:

1. Para procesos que se están ejecutando bajo el control de una misma máquina: *tuberías*, *semáforos*, *memoria compartida* y *colas de mensajes*.
2. Para procesos que se ejecutan en máquinas separadas: *sockets*.

Tuberías

Una tubería **(*pipe)*** se puede considerar como un canal de comunicación entre dos procesos. Los mecanismos que se utilizan para manipular tuberías son los mismos que para archivos, con la única diferencia de que la información de la tubería no se almacena en el disco duro, sino en la memoria principal del sistema.

Así pues, este mecanismo de comunicación consiste en la introducción de información en una tubería por parte de un proceso (similar a la escritura en un archivo de disco). Posteriormente otro proceso extrae la información de la tubería (similar a la lectura de información almacenada en un archivo de disco) de forma que los primeros datos que se introdujeron en ella son los primeros en salir. Este modo de funcionamiento se conoce como **FIFO** (**F**irst **I**n, **F**irst **O**ut; el primero en entrar es el primero en salir).

La comunicación mediante tuberías es de tipo ***half-duplex***, es decir, en un instante dado, la comunicación solamente puede tener lugar en un sentido. Si se quiere que un proceso **A** pueda simultáneamente enviar y recibir información de otro **B**, en general se debe recurrir a crear dos tuberías, una para enviar información desde **A** hacia **B** y otra para enviar desde **B** hacia **A**.

Clasificación de tuberías

Se pueden distinguir dos tipos de tuberías dependiendo de las características de los procesos que pueden tener acceso a ellas:

1. **Sin nombre**: Solamente pueden ser utilizadas por los procesos que las crean y por los descendientes de éstos.
2. **Con nombre o *fifo***: Se utilizan para comunicar procesos entre los que no existe ningún tipo de parentesco.

Además, cualquier sistema operativo Lunix garantiza que una tubería puede contener en cualquier momento un mínimo de 4096 bytes.

Tuberías sin nombre

Al igual que un archivo en disco se crea con la instrucción *open*, las tuberías sin nombre también han de ser creadas. Para tal fin se dispone de la llamada *pipe* al sistema.

```
#include <unistd.h>
int pipe (int descriptor[2]);
```

Si la llamada funciona correctamente devuelve el valor 0 y crea una tubería sin nombre. En caso contrario, devuelve –1 y deja en *errno* el código del error. Cada elemento del array *descriptor* se utiliza como si fuera un descriptor de un archivo de disco, correspondiendo el elemento *descriptor[0]* con la salida de la tubería (para escritura) y *descriptor[1]* con la entrada (para lectura). Así pues, el acceso a la tubería para comunicar información entre dos procesos tiene lugar como sigue:

1. Introducción de datos en la tubería: escritura en el "archivo" *descriptor[1].*
2. Lectura de datos de la tubería: lectura del "archivo" *descriptor[0]*.

Una característica importante de estas tuberías es que puesto que los descriptores de archivos se heredan de padres a hijos tras una llamada a ***fork()***, un hijo de un proceso puede utilizar las tuberías sin nombre que su padre hubiera creado antes de crear al hijo. En resumen, y tal como se muestra en el ejemplo siguiente, **para que un proceso padre y su hijo se puedan comunicar mediante una tubería sin nombre, ésta debe haber sido creada antes del nacimiento del proceso hijo**. Esta idea se puede extender a toda la descendencia de un proceso y no solamente a los procesos hijo.

```
#include<string.h>
#include<stdio.h>
#include<unistd.h>
#include<sys/types.h>
#include<stdlib.h>
void main()
{
    int filedes[2]; //Descriptor de archivo doble
   int pid;
   char cadena1[50]="Informacion pasada por el pipe\n";
   char entrada[255];
   printf("Ejemplo de pipe\n");
   printf("Proceso padre pid=%d\n",getpid());
   pipe(filedes);
   pid=fork();
   switch(pid)
   {       case -1: printf("Error");
           break;
           case 0:  printf("Hijo Enviando Cadena\n");
                    write(filedes[1],cadena1,strlen(cadena1));
                    exit(0);
           break;
          default:  printf("Preparando para leer informacion del hijo\n");
          close(filedes[1]);
          read(filedes[0],entrada,sizeof(entrada));
          printf("Datos leidos de la pipa:  %s\n",entrada);
          exit(0);
    }
}
```

4. PROCEDIMIENTO (DESCRIPCIÓN)	
A) EQUIPO NECESARIO	**MATERIAL DE APOYO**
Computadora con sistema operativo Linux.	Practica 6 del manual de sistemas operativos Plan 2003-1

1.- Escriba y verifique el funcionamiento del **Ejemplo 1.**
2.- Para este ejercicio utilizaremos dos programas de tuberías con nombre. El primer programa se llama Lector y el segundo Escritor.

Programa Lector.c

```
#include <stdio.h>
#include <unistd.h>
#include <sys/stat.h>
```

```
#include <sys/types.h>
#include <fcntl.h>
#define NOMBREFIFO "mififo"
#define TAM_BUF 80
#define TRUE 1
int main(void)
{
int fp;
char buffer[TAM_BUF];
int nbytes;
mknod(NOMBREFIFO,S_IFIFO|0660,0);
while(TRUE)
{
fp=open(NOMBREFIFO,O_RDONLY);
nbytes=read(fp,buffer,TAM_BUF-1);
buffer[nbytes]='\0';
printf("Cadena recibida: %s \n",buffer);
close(fp);
}
return 0;
}
```

Nota: Una vez escrito el programa, deberás ejecutarlo y dejarlo corriendo sobre esta terminar, luego escribe el programa Escritor que se presenta a continuación.

Programa Escritor .c

```
#include <sys/types.h>
#include <fcntl.h>
#define NOMBREFIFO "mififo"
int main(int argc, char *argv[])
{
int fp;
   int result=1;
   if(argc!=2)
   printf("uso: %s cadena \n", argv[0]);
   else if ((fp=open(NOMBREFIFO,O_WRONLY))==-1)
   perror("fopen");
   else
   {
   write(fp,argv[1],strlen(argv[1]));
   close(fp);
   result=0;
   }
   return result;
   }
```

Nota 1: Para ejecutar este programa primero abre otra terminal y ejecuta el programa escribiendo lo siguiente: **./Escritor HolaCrayola**

Nota 2: Observa lo que tienes ahora en la terminar que esta ejecutando el proceso Lector. **¿Qué sucedió?**

3. Investiga como se utiliza el mando **mkfifo** desde el shell de Linux (no como una función de C bajo Linux) y deberás realizar un ejemplo de este mando **frente a tu maestra(o)**. Recuerda que para Linux una tubería no es más que un archivo por lo tanto las puedes borrar de la misma forma en que borras un archivo.

C) CÁLCULOS Y REPORTE

5. RESULTADOS Y CONCLUSIONES

Al finalizar la práctica el alumno demostrará su habilidad para enviar señales mediante el uso de tuberías.

6. ANEXOS

7. REFERENCIAS

http://148.231.82.20/~so