Sistema Operativo Linux

Gilberto Díaz

27 de enero de 2014

Licencia de Uso

Centro de Teleinformación
Corporación Parque Tecnológico de Mérida
Mérida Edo. Mérida - Venezuela

Índice General

Índice de Figuras

Índice de Tablas

Acerca de este manual

Audiencia

Este manual, al igual que el curso, está dirigido a personas que han tenido muy poca, o ninguna, experiecia con sistemas operativos compatibles con Unix/Linux. Asi mismo, los introduce en los detalles de la administración de este tipo de sistemas operativos. Intenta igualmente ser un tutorial organizado por tipo de tarea.

Objetivos

Este material trata sobre el uso, manejo y administración del sistema operativo Unix Incluye ejemplos prácticos basados en GNU/Linux

Al finalizar este manual usted debe estar en capacidad de:

- Acceder a su ambiente de trabajo en una máquina GNU Linux.
- Entender y utilizar los distintos elementos del sistema operativo.
- Entender y utilizar los conceptos de directorios y archivos.
- Ejecutar aplicaciones.
- Utilizar aplicaciones de red para comunicarse con otros sistemas
- Instalar sistemas Unix basados en Linux.
- Conocer los detalles de arranque y detención de Unix.
- Manejar usuarios.
- Administrar sistemas de archivos.
- Administrar procesos y aplicaciones.
- Administrar interfaces de red.
- Manejar sistemas de impresión.

- Conocer los diferentes detalles de seguridad.
- Conocer herramientas para supervisión de redes.

Organización

Este manual está organizado en 5 capitulos como sigue:

- **Introducción al Sistema Operativo Linux** Aquí se hace un recuento de la historia de este sistema operativo y se introducen algunos conceptos básicos y útiles para entender la relación de los sistemas operativos y las computadoras.
- **El ambiente del Usuario** Es una introducción a los ambientes de trabajo de los usuarios. Aquí se describen los interpretadores de comandos o *conchas* y los ambientes de ventanas de GNU Linux, en particular el ambiente **X**
- **Manejo de Archivos** Muestra el funcionamiento y comandos básicos del editor de archivos de Linux `vi`. Se hace, también, una introducción a algunos otros editores de archivos.
- **Manejo de Directorios** En esta seccion se demuestra la forma en la que el usuario debe ver el contenido y propiedades de los directorios ademas de crear nuevos directorios asi como tambien eliminarlos, en el mismo orden de ideas el usuario podra a su vez ser capaz de buscar archivos y manejar la permisologia de los mismos.
- **Manejo de medios de almacenamiento secundarios** Se muestra el uso correcto de los distintos medios de alamacenamiento secundarios como Pen Drives y Discos Compactos, ademas de compartir los Directorios en red .

Información relacionada en el Web

En el web existen los siguientes documentos:

- A Basic UNIX Tutorial:
  http://www.isu.edu/departments/comcom/unix/workshop/unixindex.html
- Linux Online Courses
  http://www.linux.org/lessons/
- Introduction To UNIX:
  http://www.ceas.rochester.edu:8080/CNG/docs/IntroUnix.html
- Unix
  http://metalab.unc.edu/echernof/unix/what.html

Convenciones

En este manual se utilizan las siguientes convenciones:

`%`	Un signo de porcentaje al iniciar una linea en los ejemplos representa el *prompt* (mensaje de espera) de una concha tipo **C shell**
`$`	Un signo de dollar representa el *prompt* de una concha tipo **Bourne Shell**.
`#`	Un signo de número representa el *prompt* de superusuario.
`ctrl-x`	La secuencia de caracteres `ctrl` delante de una letra indica que se debe mantener presionada la tecla **Ctrl** al mismo tiempo que se presiona la letra indicada.
`alt-x`	La secuencia de caracteres `alt` delante de una letra indica que se debe mantener presionada la tecla **alt** al mismo tiempo que se presiona la letra indicada.

[10]

Capítulo 1

Programación de Módulos

Una distribución tradicional de Linux viene con diversos módulos para manejo de dispositivos y procesos. Estos módulos se encuentran en el directorio `/lib/modules/version`, y se están organizados en subdirectorios, por el tipo de dispositivo o función que cumplen.

La programación de módulos para el núcleo de Linux tiene algunas similitudes a la programación tradicional de una aplicación en lenguaje C, como por ejemplo la posibilidad de utilizar funciones externas. Sin embargo, sólo se pueden utilizar un conjunto reducido de funciones publicas definidas por el kernel. Además, un módulo no es un ejecutable independiente, sino un código objeto que se cargará al núcleo en tiempo de ejecución.

Un módulo debe tener como mínimo dos funciones:

- `init_module()`: es la función que se ejecuta cuando se carga el módulo.
- `cleanup_module()`: es la función que se ejecuta cuando se descarga el módulo.

A continuación se presentarán los pasos necesarios para gestionar módulos, y luego se presentarán un conjunto de ejemplos, donde se detallará el uso de las funciones que se acaban de describir, así como de otras funciones y conceptos sobre la progamación de módulos para el núcleo de Linux.

1.1. Gestión de Módulos

1.1.1. Compilación

Por facilidad, para compilar módulos generalmente se utiliza un Makefile. Un ejemplo de un Makefile simple es el siguiente:

```
obj-m += helloWorld.o

all:
```

```
make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules

clean:
make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

Esto generará un archivo con el mismo nombre y extensión .ko, que corresponde al código objeto del módulo, listo para ser cargado en el kernel. Nótese que se utiliza la opción -C para especificar que se genere código objeto.

1.1.2. Carga

Para insertar el módulo en el kernel, luego de ser compilado, se utiliza el comando

```
insmod ./nombre_modulo.ko
```

Todos los módulos cargados en el kernel, se listan en `/proc/modules`. El comando `lsmod` también puede ser utilizado para mostrar por cónsola el contenido de ese archivo.

1.1.3. Descarga

Para descargar módulos del kernel se utiliza el comando `rmmod`:

```
rmmod nombre_modulo
```

1.2. Ejemplos

1.2.1. Ejemplo 1: Hola Mundo

Como primer ejemplo se presenta un módulo sencillo, que imprime un mensaje al cargarse el módulo y un mensaje al descargarse.

```
#include <linux/module.h> /* Necesario para todos los modulos*/
#include <linux/kernel.h> /* Necesario para utilizar KERN_INFO */

int init_module(void)
{
  printk(KERN_INFO "Hola Mundo\n");

  /*
   * Un valor distinto a cero significa
   * que el módulo falló
  */
```

```
  return 0;
}

void cleanup_module(void)
{
  printk(KERN_INFO "Hasta luego\n");
}
```

Todo módulo del kernel debe incluir la librería `linux/module.h`, y adicionalmente para utilizar la macro `KERN_INFO` se debe incluir la librería `linux/kernel.h`.

En este ejemplo se utiliza la funcion `printk()`, la cual es un mecanismo para dejar registro de lo que ocurre en el kernel. `printk()` no fue concebida para comunicar información al usuario, sin embargo funciona de manera similar a la función `printf()`, sólo que la salida no es arrojada por cónsola, sino que se escribe al archivo `/var/log/messages`.

Para compilar el código, utilizando el Makefile mostrado anteriormente, se utiliza el comando `make`, el cual genera la siguiente salida por cónsola:

```
hostname:~/lkmpg-examples/02-HelloWorld# make
make -C /lib/modules/2.6.11/build M=/root/lkmpg-examples/02-HelloWorld modules
make[1]: Entering directory ‘/usr/src/linux-2.6.11’
  CC [M] /root/lkmpg-examples/02-HelloWorld/hello-1.o
 Building modules, stage 2.
  MODPOST
```

Una vez generado el archivo hello-1.ko, se instala en el kernel, con el comando insmod ./hello-1.ko

Para verificar que el módulo se instaló correctamente, se puede visualizar el contenido del archivo `/var/log/messages`, en el cual debe aparecer el mensaje "Hola mundo"que se imprimió mediante `printk()` en la función de arranque del módulo.

Luego de descargar el módulo (utilizando el comando `rmmod hello-1`), también en `/var/log/messages`, debe aparecer el mensaje "Hasta luego", que se imprimió en la función `cleanup_module`, al descargar el módulo.

1.2.2. Ejemplo 2: Agregando macros

A partir de la version 2.4 del kernel de Linux, se pueden dar nombres a las funciones de carga/descarga de los módulos, distintos a `init_module()` y `cleanup_module()`. Esto se hace mediante las macros `module_init()` y `module_exit()`, las cuales se encuentran definidas en `linux/init.h`. La única restricción es que estas funciones deben ser definidas antes de llamar a los macros.

A continuación se muestra un ejemplo muy similar al anterior, en el que se ejemplifica el uso de `module_init()` y `module_exit()`.

```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h> /* Necesario para utilizar module_init y module_exit */

static int __init hello_2_init(void)
{
  printk(KERN_INFO "Hola mundo 2\n");
  return 0;
}

static void __exit hello_2_exit(void)
{
  printk(KERN_INFO "Hasta luego\n");
}

module_init(hello_2_init);
module_exit(hello_2_exit);
```

1.2.3. Ejemplo 3: Manejando licencias

La macro `MODULE_LICENSE` se utiliza como un mecanismo para declarar la licencia bajo la cual se distribuye el módulo. Las licencias GPL (y similares) evitan que se imprima una advertencia de que el código no es abierto.

Similar a `MODULE_LICENSE`, funciona la macro `MODULE_DESCRIPTION`, la cual permite agregar una descripción del objetivo y funcionamiento del módulo. La macro `MODULE_AUTHOR` declara el autor del módulo y `MODULE_SUPPORTED_DEVICES` declara el tipo de dispositivos soportados por el módulo.

Todas estas macros se definen en `linux/module.h`. En el siguiente ejemplo se muestran en funcionamiento cada una de ellas.

```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h>

#define DRIVER_AUTHOR "Gilberto Diaz <gilberto@ula.ve>"
#define DRIVER_DESC "Descripcion del Modulo"

static int __init init_hello(void)
{
  printk(KERN_INFO "Hola, mundo \n");
  return 0;
```

```
}

static void __exit cleanup_hello(void)
{
  printk(KERN_INFO "Hasta luego \n");
}

module_init(init_hello);
module_exit(cleanup_hello);
MODULE_LICENSE("GPL"); /* Declarar que el módulo tiene licencia GPL */
MODULE_AUTHOR(DRIVER_AUTHOR); /* Definición del autor del módulo */
MODULE_DESCRIPTION(DRIVER_DESC); /* Descripción del módulo */
MODULE_SUPPORTED_DEVICE("testdevice"); /* dispositivo soportado (/dev/testdevice) */
```

1.2.4. Ejemplo 4: Pase de parámetros

Para pasar argumentos por línea de comando a un módulo se utiliza la macro `module_param`, que se define en `linux/moduleparam.h`. La declaración de las variables y los macros se debe realizar al comienzo del módulo y además, las variables deben ser declaradas como globales.

La macro `module_param` toma 3 argumentos: el nombre de la variable, su tipo y los permisos correspondientes para el archivo en el sistema de archivos.

A continuación se presenta un ejemplo:

```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h>

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Gilberto Diaz");

int Parametro1 = 0;
int Parametro2 = 0;

module_param(Parametro1, int, 0);
MODULE_PARM_DESC(Parametro1, "Primer parametro: entero con signo");

module_param(Parametro2, int, 0);
MODULE_PARM_DESC(Parametro2, "Segundo parametro: entero con signo");

static int __init iniciar(void)
{
```

```
  printk(KERN_INFO "Ejemplo de Parametros\n");
  printk(KERN_INFO "Parametro1 = %i\n", Parametro1);
  printk(KERN_INFO "Parametro2 = %i\n", Parametro2);
  return 0;
}

static void __exit finalizar(void)
{
  printk(KERN_INFO "Hasta Luego\n");
}

module_init(iniciar);
module_exit(finalizar);
```

En tiempo de ejecución, el comando `insmod` llenará las variables con cualquier argumento dado en la línea de comandos. Por ejemplo: `insmod ./nombre_modulo.ko variable = valor`

Por último, la macro `MODULE_PARM_DESC` se utiliza para documentar los argumentos que el módulo podría tomar.

1.2.5. Ejemplo 5: Utilizando varios códigos fuentes

A veces es necesario o conveniente dividir el código fuente del módulo en varios archivos. A continuación se muestra un ejemplo donde se ha dividido el código fuente en dos archivos file1.c y file2.c.

El contenido de file1.c es el siguiente:

```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h>

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Gilberto Diaz");

int Parametro1 = 0;
extern int Parametro2;

module_param(Parametro1, int, 0);
MODULE_PARM_DESC(Parametro1, "Primer parametro: entero con signo");

int init_module(void)
{
```

```
  printk(KERN_INFO "Ejemplo de Modulo codificado en varios archivos\n");
  printk(KERN_INFO "Parametro1 = %i\n", Parametro1);
  printk(KERN_INFO "Parametro2 = %i\n", Parametro2);
  return 0;
}
```

El archivo file2.c contiene el siguiente código:

```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h>

extern int Parametro1;
int Parametro2 = 0;

module_param(Parametro2, int, 0);
MODULE_PARM_DESC(Parametro2, "Segundo parametro: entero con signo");

void cleanup_module(void)
{
  printk(KERN_INFO "Hasta Luego\n");
}
```

Para compilar este módulo utilizamos un Makefile en el que especificamos los nombres de archivos fuente, así como también un nombre de objeto para el módulo combinado. Este ejemplo puede ser compilado utilizando el siguiente Makefile.

```
obj-m += variosArchivos.o
variosArchivos-objs :=  file1.o file2.o

all:
  make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules

clean:
  make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

1.2.6. Ejemplo 6: Comunicacion dinámica con el kernel

El sistema de archivos /proc es un mecanismo que permite al kernel y a los módulos enviar información a los procesos. A continuación se extenderá el código del ejemplo anterior, para mostrar como se utiliza /proc.

Este código crear un archivo /proc/holaMundo (en la función init_module), retorna un valor cada vez que es leido este archivo (en la función procfile_read) y finalmente se borra (en la función cleanup_module).

El código del archivo fuente file1.c es el siguiente:

```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h>
#include <linux/proc_fs.h> /* Necesario cuando se utiliza /proc */

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Gilberto Diaz");

#define procfs_name "EjemploDeUsoDeProcFS"


int Parametro1 = 0;
extern int Parametro2;

struct proc_dir_entry *Proc_File;


module_param(Parametro1, int, 0);
MODULE_PARM_DESC(Parametro1, "Primer parametro: entero con signo");


/* Esta función se invoca cada vez que se lee el archivo en /proc
 *
 * Argumentos
 * =========
 * 1. Buffer donde los datos se insertan, si se decide utilizarlo.
 * 2. Apuntador a apuntador de caracteres. Útil si se decide no
 *    utilizar el buffer apartado por el kernel.
 * 3. Posición actual en el archivo.
 * 4. El tamaño del buffer del primer argumento.
 * 5. Escribe "1" aqui para indicar fin de archivo (EOF).
 * 6. Apuntador a los datos. Útil en caso de una lectura común para
 *    múltiples entradas en /proc)
 *
 */
int procfile_read(char *buffer, char **buffer_location, off_t offset,
```

```
                  int buffer_length, int *eof, void *data)
{
  int ret;
  printk(KERN_INFO "El archivo en /proc ha sido leido (/proc/%s) called\n", procfs_name);
  if (offset > 0)
  {
    /* Retornar 0 si se termino de leer */
    ret = 0;
  }
  else
  {
    /* Llenar elbuffer, retornar el tamano del buffer */
    ret = sprintf(buffer, "Prueba de Proc FS!\n");
  }
  return ret;
}

int init_module(void)
{
  Proc_File = create_proc_entry(procfs_name, 0644, NULL);

  printk(KERN_INFO "Ejemplo de Modulo con ProcFS\n");
  printk(KERN_INFO "Parametro1 = %i\n", Parametro1);
  printk(KERN_INFO "Parametro2 = %i\n", Parametro2);

  if (Proc_File == NULL)
  {
    remove_proc_entry(procfs_name, NULL);
    printk(KERN_ALERT "Error: No se pudo inicializar /proc/%s\n", procfs_name);
    return -ENOMEM;
  }
  Proc_File->read_proc = procfile_read;
  Proc_File->mode = S_IFREG | S_IRUGO;
  Proc_File->uid = 0;
  Proc_File->gid = 0;
  Proc_File->size = 37;
  printk(KERN_INFO "/proc/%s created\n", procfs_name);

  return 0;
}
```

El contenido de file2.c es el siguiente:

```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h>
#include <linux/proc_fs.h>

#define procfs_name "EjemploDeUsoDeProcFS"

extern int Parametro1;
int Parametro2 = 0;

module_param(Parametro2, int, 0);
MODULE_PARM_DESC(Parametro2, "Segundo parametro: entero con signo");

void cleanup_module(void)
{
  printk(KERN_INFO "Hasta Luego\n");
  remove_proc_entry(procfs_name, NULL);
}
```

Bibliografía

[1] Kernel.org.

[2] Linux.die.net.

[3] Ivan Bowman. Conceptual architecture of the linux kernel.

[4] Marco Cesati Daniel P. Bovet. *Understanding the Linux Kernel*. O'Reilly & Associates Inc., 2003.

[5] Michael K. Johnson. Diff, Patch, and Friends. *Linux J.*, 1996(28es), 1996.

[6] M. Tim Jones. Anatomy of the linux kernel.

[7] M. Tim Jones. Inside the linux boot process.

[8] Robert Love. *Linux Kernel Development (3rd Edition)*. Addison-Wesley Professional, 3 edition, 7 2010.

[9] Gary Lawrence Murphy. *The Linux Kernel: Blueprints for World Domination*. MacMillan Computer Publishing, 2000.

[10] Peter Jay Salzman, Michael Burian, and Ori Pomerantz. *The Linux Kernel Module Programming Guide*. CreateSpace, 1 2009.