ETS de Ingenierías Informática y de Telecomunicación

LKM

Trabajo SO – Grupo C

2016/17

Integrantes:

Guillermo Gómez Trenado,
Sonsoles Jiménez Gómez
Juan Salvador Molina Martín,
Miguel Ángel Rispal Martínez,
Juan Carlos Ruiz García

LKM

Contenido

¿Qué es un LKM?

1-.Los módulos del núcleo son fragmentos de código objeto (.ko) que pueden ser cargados y eliminados del núcleo. Extienden la funcionalidad del núcleo en tiempo de ejecución, sin necesidad de reiniciar el sistema.[1]

Dichos ficheros .ko se obtienen como resultado de la compilación con una sintaxis especial creado por GNU con el fin de facilitar la tarea de compilar los módulos. El código del módulo insertado pasa a ser una parte del núcleo del sistema y por lo tanto se ejecuta con el máximo nivel de privilegio.[2]

La única diferencia con el código estático (el que se carga al arrancar el sistema) es que un módulo se puede descargar desvinculándose del núcleo, liberando todos los recursos utilizados, la memoria consumida por el propio módulo e invalidando las referencias que hubiera podido exportar al sistema.[3]

¿Qué estructura tienen?[4]

modulo.c

```
#include[s] ...

static int __init entrada(void)
{
printk(KERN_INFO "Mensaje"); //Versión kernel de printf
//Código
return 0; // Salida exitosa
}

static void __exit salida(void)
{
//Código
}

module_init(entrada); // función de entrada
module_exit(salida); // función de salida

MODULE_AUTHOR("AUTOR");
MODULE_LICENSE("LICENCIA");
MODULE_DESCRIPTION("DESCRIPCIÓN");
```

Cómo construir un módulo kernel

Los módulos kernel deben tener al menos dos funciones, una de inicio – llamada al ejecutar insmod - y otra de salida – al ejecutar rmod -. Tradicionalmente eran llamadas init_module() y cleanup_module() respectivamente, pero desde el kernel 2.3.13 puedes usar cualquier otro nombre – así se recomienda de hecho – pues se inicializan llamando a las macro module_init(nombre_funcion_entrada) y module_exit(nombre_funcion_salida), estas funciones deben estar definidas antes de llamar a las macros – incluidas en linux/init.h -. Por norma general, la función de inicio registra un manejador para algún aspecto del kernel o sustituye una de las funciones kernel con las suyas propias.[5]

1. https://wiki.archlinux.org/index.php/Kernel_modules_(Espa%C3%B1ol)
2. http://mermaja.act.uji.es/docencia/is37/data/p1.pdf
3. http://web-sisop.disca.upv.es/gii-dso/es/pl2-modulos/pl2-modulos.html
4. http://www.tldp.org/LDP/lkmpg/2.6/html/x279.html
5. http://www.tldp.org/LDP/lkmpg/2.6/html/x121.html

Kbuild (Makefile)

Los módulos kernel han de ser compilados de forma ligeramente distinta a los programas regulares, históricamente se compilaban haciendo uso de makefiles independientes, pero en los kernel actuales se usan kbuild, que integran los objetos dentro del propio mecanismo de construcción del kernel.

El makefile(kbuild) básico sería:

```
obj-m += hello-1.o
```

Donde el objeto hello-1.o – nuestro objeto – se añade a la cadena obj-m, gestionada por el kernel.

A partil del kernel 2.6 se sustituye la convención de nomemclatura de los objetos de .o a .ko, archivos objetos regulares con información adicional sobre el módulo[6]

Uso de argumentos

Los módulos pueden permitir argumentos por comandos, pero al contrario que los programas en C a los que estamos acostumbrados – con argc y argv – en un LKM hay que definir las variables como globales y llamar a la macro module_param(), module_param_array() o module_param_string() incluidas en linux/moduleparam.h dependiendo del tipo de argumento, y al usar insmod los argumentos serán pasados al módulo.[7]

Módulos y variables globales

En un programa normal, cuando acumulas demasiadas variables globales que no son suficientemente distinguibles se produce lo que se conoce como contaminación del espacio de nombres – namespace pollution –, en grandes proyectos hay que seguir estrategias de nombrado eficaces y distinguibles. Cuando escribimos código para el kernel, esto es aún más importante, pues todas las variables globales son accesibles desde todo el kernel, pudiendo oscurecer variables necesarias por otros módulos o el propio núcleo, con consecuencias del todo indeseables. La mejor forma de evitar esto es definiendo las variables como estáticas y usando prefijos bien definidos y diferenciados. Otra forma es el uso de tablas de símbolos registradas en el kernel.[3]

Llamadas al sistema

Dentro de un módulo kernel puedes modificar la tabla de llamadas al sistema, normalmente para añadir algunas instrucciones y llamar posteriormente a la función original. Es importante restaurar aquellas entradas de la tabla modificadas a su estado original al salir del módulo.[8]

Gestor de interrupciones

Las interrupciones hardware en Linux son llamadas IRQs y hay dos tipos, cortas y largas, dependiendo de si pueden ser ligeramente postergadas en el tiempo o han de ser atendidas rápidamente para no perder la información, es mejor práctica declarar – a ser posible- los manejadores de interrupciones como largas. Las interrupciones se pueden producir en cualquier momento, por eso hay ciertas cosas que no se permiten hacer dentro del manejador, por estar en un estado desconocido de ejecución, por esto lo normal es salvar la información entregada por el dispositivo que ha generado la interrupción y programar la gestión de la información para luego, comprometiéndose el kernel a llamar a esta segunda parte del manejador tan pronto como sea posible.

6. http://www.tldp.org/LDP/lkmpg/2.6/html/x181.html
7. http://www.tldp.org/LDP/lkmpg/2.6/html/x323.html
8. http://www.tldp.org/LDP/lkmpg/2.6/html/x978.html

La forma de establecer es un manejador es mediante request_irq(), mandándole el nombre de la función y la información relativa a la interrupción que quieres manejar y una vez recibida una interrupción y salvada la información queue_work() para programar la posterior gestión de esa información.9

Errores frecuentes

· Usar la librería estándar de C. Sólo pueden utilizarse las funciones declaradas por el kernel, aquellas que puedes ver en /proc/kallsyms

· Deshabilitar interrupciones. Puedes necesitar hacer esto por breves instantes, pero si no las habilitas otra vez al finalizar el sistema se quedará colgado hasta que fuerces el reinicio.

· Meter la cabeza dentro de la boca de grandes carnívoros – tenía que conservar el chiste de Peter J. Salzman del texto original -.

¿Qué órdenes suministra el sistema para manejarlos?

Instalar/Desinstalar:

A la hora de instalar e desinstalar programas utilizamos simplemente los comandos insmod y rmmod. Para ello tenemos que encontrarnos en modo superusuario. Aunque parezca muy fácil llevar a cabo la instalación puede ser bastante irritante debido al mal uso de los comandos o a instalar una versión incompatible del kernel o del módulo mostrando entonces un error por pantalla.

Un ejemplo de esto sería:

```
insmod msdos.o
  msdos.o: unresolved symbol fat_date_unix2dos
  msdos.o: unresolved symbol fat_add_cluster1
  msdos.o: unresolved symbol fat_put_super
  ...
```

Esto se debe a que ciertos símbolos en esa orden no se encuentran en el kernel. Para ver esta tabla de símbolos usamos el comando:

```
cat /proc/ksyms
```

En muchos casos necesitamos pasarle ciertos parámetros, por ejemplo, al instalar un controlador de dispositivos se necesita saber las direcciones para que no se produzca errores. Un ejemplo sería:

```
insmod ne.o io=0x300 irq=11
```

En este caso provocaría una interrupción IRQ(es una señal de un dispositivo hardware indicando que el dispositivo necesita que la CPU haga algo).

Una de las características más destacables y por lo tantos más problemáticas es que no hay parámetros entandar en los LKM por lo que cada autor decide los parámetros que llevará.

A la hora de desinstalar los LKM del núcleo utilizaremos el comando;

```
rmmod ne
```

9. *http://www.tldp.org/LDP/lkmpg/2.6/html/x1256.html*

Para ello suele ser útil utilizar el comando lsmod para muestra que LKMs tenemos cargados y desinstalar aquellos que no queramos utilizar.

Otra forma seria utilizar el comando rmmod --all que nos sirve para borrar aquellos que no han sido usados durante mucho tiempo.

Otros errores comunes es querer insertar un archivo objeto que no es un LKM, un ejemplo sería:

```
$ insmod usbcore.o
```

Nos mostraría un mensaje de error donde nos dice que no encuentra la versión del kernel.[10]

Módulos Disponibles

Se puede usar lsmod para mostrar un listado de los módulos cargados. Además, para mostrar información sobre tales módulos, se usa modinfo.

Para ver los módulos cargados podemos utilizar el comando;

```
cat /proc/modules

serial     24484   0
```

En la izquierda aparece el nombre de la LKM que usualmente aparece con el nombre del archivo objeto con el sufijo .o . En la columna del centro aparece lo que ocupa en bytes el LKM en memoria. En la columna de la derecha aparece el número de dispositivos que dependen de él. Esto sería importante ya que a menos que este "contador" sea cero no podemos quitarlo. Una excepción seria que se mostrara un -1 lo que esto significa es que este LKM no utiliza el recuento de uso sino si es aceptable la descarga del LKM y para estos casos normalmente el LKM debe dar una información.[11]

Dependencias entre ellos

El comando depmod genera el archivo modules.dep que registra las dependencias de los módulos entre sí. Cada línea de ese archivo tiene el nombre de un módulo seguido del carácter ':' y los módulos de los cuales depende, separados por espacios. Como un módulo puede requerir otros, hay dependencias que deben respetarse al cargar y descargar módulos. depmod permite calcular tales dependencias entre varios módulos o entre todos los disponibles con la opción –a. Durante el arranque las dependencias entre módulos son generadas automáticamente y los módulos especificados (junto con sus opciones) en el archivo /etc/modules son cargados. El archivo con las dependencias de los módulos se genera automáticamente cuando Linux se inicia (salvo que el archivo ya exista).[12]

Muchos de los LKM dependen de otros (en los símbolos que exporta) y que el caso que se nos muestra lp y parport_pc son dependientes de parport.

Unos de las características principales de las dependencias es que no se pueden desinstalar los LKM que dependen de otros hasta que no se hayan estos últimos.

Propiedades de un módulo

Se pueden examinar las propiedades de un módulo ya cargado con la orden tree. Por ejemplo:

tree /sys/module/"nombre del módulo en concreto"[13]

10. http://tldp.org/HOWTO/Module-HOWTO/x197.html#PROCMODULES
11. http://tldp.org/HOWTO/Module-HOWTO/x197.html#PROCMODULES
12. http://structio.sourceforge.net/guias/AA_Linux_colegio/kernel-ymodulos.html
13. http://www.crashcourse.ca/introduction-linux-kernel-programming/lesson-9-all-about-module-parameters

Ventajas e inconvenientes

Ventajas

- Cargar y descargar un módulo es mucho más flexible y rápido que recompilar un kernel y reiniciar.
- Puedes probar diferentes opciones cada vez que cargas un módulo. La mayoría de los drivers que se encargan del hardware tendrán opciones para E/S de direcciones, IRQ (Interrupt request) o DMA (Direct Memory Access), además de opciones reservadas como "full o half duplex"[14]. Por ejemplo, cuando tienes problemas para conseguir que una tarjeta funcione correctamente, la habilidad de probar diferentes opciones puede ahorrarte tiempo.
- Hace más fácil mantener múltiples máquinas en un único kernel base.

Inconvenientes

- Previamente tienes que planear y decidir que módulos compilar.
- Los módulos tienen que estar cargados. Esto puede ser manual o automático, pero aun así tiene que estar hecho.
- La tabla de símbolos para módulos es dinámica lo cual complica la depuración.
- Algunas personas consideran los módulos como un riesgo de seguridad porque un *cracker* puede cargar un módulo el cual lo esconda a él y a su código del administrador real. Personalmente considero esto engañoso. Solo *root* puede cargar módulos y si un *cracker* ya tiene *root* entonces tu máquina está muerta de todas formas.

Conclusión

En definitiva, los LKMs son herramientas muy potentes para los desarrolladores que permiten conservar las ventajas de los núcleos monolíticos – especialmente la eficiencia – e introducir a la vez un alto grado de versatilidad y facilidad de uso de sistemas LINUX para los usuarios, evitando tener que compilar el kernel cada vez que quieran usar un nuevo periférico o extender una funcionalidad del sistema.

Dicho esto no hay que olvidar que de la misma forma que un usuario o administrador de sistemas debe extremar las precauciones a la hora de instalar nuevos programas esto se presenta de una forma aún más crítica con los módulos de núcleo, pues normalmente el kernel viene incluido en una distribución Linux de confianza o es descargado de una fuente segura, sin embargo, al hacer uso de módulos kernel de terceros estamos abriendo las puertas de nuestro sistema a quien quiera hacerse con el control de éste.

De igual forma, como desarrolladores, al escribir código para LKM debemos andar con especial cautela, pues mientras que un error en un programa puede colgar la ejecución de ese proceso, un error – como puede ser un seg fault – en el kernel implica el cuelgue del sistema – en el mejor de los casos – y la potencial pérdida de archivos o interrupciones de servicios críticos. Por esto, el uso de LKM propios en producción exige un control mucho más estricto que al que se ven sometidos los programas que se ejecutan en modo usuario.

14. *Una comunicación full-duplex entre dos componentes significa que ambos pueden transmitir y recibir información entre sí de forma simultánea. En sistemas half-duplex, la transmisión y recepción de información tiene que ocurrir alternativamente. Mientras un punto esta transmitiendo, el otro solo recibe.*

Bibliografía

· (2012, 08). Kernel modules (Español). archWiki. Obtenido 10, 2016, de https://wiki.archlinux.org/index.php/Kernel_modules_(Espa%C3%B1ol)

· Fabregat Llueca, G. (2013, 09). Práctica 1. Módulos del núcleo de Linux.. Obtenido 10, 2016, de http://mermaja.act.uji.es/docencia/is37/data/p1.pdf

· Pérez Cortés , J. (2008, 09). Práctica 2. Módulos del núcleo. Dpto. de Informática de Sistemas y Computadores. Obtenido 10, 2016, de http://web-sisop.disca.upv.es/gii-dso/es/pl2-modulos/pl2-modulos.html

· Salzman, P. (2007, 05). The Linux Kernel Module Programming Guide. The Linux Documentation Project. Obtenido 10, 2016, de http://www.tldp.org/LDP/lkmpg/2.6/html/index.html

· Henderson, B. (2006, 09). Linux Loadable Kernel Module HOWTO. The Linux Documentation Project. Obtenido 10, 2016, de http://tldp.org/HOWTO/Module-HOWTO/index.html

· Day, R. (2010, 06). An introduction to Linux kernel programming. CrashCourse. Obtenido 10, 2016, de http://www.crashcourse.ca/introduction-linux-kernel-programming/lesson-9-all-about-module-parameters