# Programacin de Controladores para Linux

Edgardo E. Hames, Julio A. Bianco

Revisin 2.0.1

## 1. Introduccin

Linux es un clon del sistema operativo Unix, escrito desde cero por Linus Torvalds con ayuda de un grupo lejano de *hackers* en la Red. Linux tiene todas las caractersticas de un sistema operativo moderno incluyendo multitarea, memoria virtual, bibliotecas compartidas, carga en demanda, una correcta administracin de la memoria y soporte de red para IPv4 e IPv6 entre otros protocolos. La mayor parte de Linux es independiente del hardware donde se ejecuta. Sin embargo, para cada dispositivo soportado por Linux, alguien ha escrito el correspondiente controlador que lo hace interactuar con el resto del sistema. Sin estos controladores, ningn sistema es capaz de funcionar.

Los controladores de dispositivos (*device drivers*) desempean un papel muy importante en el ncleo de Linux: son las cajas negras que hacen que cierto hardware responda a una interfaz bien definida de software y ocultan completamente los detalles de cmo funciona el dispositivo. Las actividades desde el espacio de usuario se realizan por medio de un conjunto estandarizado de llamadas al sistema que son independientes del controlador: **el rol del controlador es asociar esas llamadas a las operaciones especficas del hardware**. Esta interfaz de programacin es tal que los controladores pueden ser construidos en forma separada del resto del ncleo, y enlazados y activados en tiempo de ejecucin cuando sean necesarios. Un controlador de Linux programado para ser cargado y activado sobre un ncleo activo se denomina "mdulo".

El propsito de este material es presentar una introduccin sobre cmo desarrollar controladores para el ncleo de Linux enfatizando la construccin de mdulos.

## 2. Utilidades

Cuando trabajamos con mdulos para el ncleo de Linux, el conjunto de utilidades *modutils* nos permiten manipularlos desde su compilacin hasta que deseemos removerlos del sistema. Aqu se presenta una breve resea de cada una, pero se recomienda al lector que consulte las pginas del manual de cada una de estas aplicaciones.

- depmod: Crea una dependencia intermodular al estilo de Makefile, basado en los smbolos que encuentra en los mdulos mencionados en la lnea de comandos o en los directorios especificados en el archivo de configuracin. Este archivo es utilizado por *modprobe* para cargar la pila correcta de mdulos.

  -a Buscar los mdulos en todos los directorios especificados en el archivo de configuracin */etc/modules.conf*.

-e Muestra los smbolos no resueltos de cada mdulo.

-n Escribe el archivo de dependencia en la salida estndar en vez de en el rbol de */lib/modules*.

- modinfo: Muestra informacin sobre un mdulo.

  -a Muestra el autor del mdulo.

  -d Muestra la descripcin del mdulo.

  -l Muestra la licencia del mdulo.

  -p Muestra los parmetros del mdulo.

  -n Muestra el path completo del archivo que corresponde al mdulo.

- lsmod: Muestra la lista de mdulos cargados. Esta informacin se obtiene de */proc/modules*.

- insmod: Instala un mdulo en el ncleo en ejecucin.

  -f Carga el mdulo aunque pertenezca a una versin distinta del ncleo.

  -p Prueba si el mdulo puede ser cargado.

- rmmod: Desinstala un mdulo del ncleo en ejecucin.

- modprobe: Instala o desinstala mdulos del ncleo en ejecucin.

  -r Descarga un mdulo y todos los que lo referencian.

- dmesg: Permite examinar los mensajes del ncleo. Los dos usos ms frecuentes son:

```
[usuario@localhost]$ dmesg >boot.messages
[usuario@localhost]$ dmesg | less
```

# 3. Compilacin del kernel

Una de las ventajas de Linux es que el ncleo del sistema puede ser recompilado por el usuario para ajustarlo a sus necesidades. De esta manera, uno puede deshabilitar funcionalidades que no vayan a ser usadas y obtener as un ncleo ms liviano y rpido.

Brevemente, los pasos para recompilar el ncleo de Linux son:

1. Descomprimir el archivo con el cdigo fuente.

   ```
   [root@localhost]# tar -xjf linux-2.6.20.tar.bz2
   ```

2. Cambiar al directorio donde se descomprimi el archivo.

   ```
   [root@localhost]# cd linux-2.6.20
   ```

3. Configurar el ncleo. Esto puede realizarse de alguna de estas maneras:

   `[root@localhost]# make config` (Modo interactivo)

   `[root@localhost]# make menuconfig` (Men modo texto)

`[root@localhost]# make xconfig` (Men modo grfico)

4. Generar las dependencias de los archivos, compilar la imagen del ncleo, los mdulos.

   `[root@localhost]# make`

5. Instalar los mdulos. Esto los copiar a una carpeta en */lib/modules* que depender de la versin del ncleo que se ha compilado. Por ejemplo, */lib/modules/2.6.20*.

   `[root@localhost]# make modules_install`

6. Copiar el ncleo a */boot*.

   `[root@localhost]# cp arch/i386/boot/bzImage /boot`

7. Configurar el cargador de arranque. Dependiendo de cul use, esto consistir en agregar una nueva entrada para el ncleo que acaba de compilar.

8. Reiniciar el equipo y rezar para que todo ande.

Se puede encontrar una explicacin ms detallada de cmo recompilar el ncleo de Linux, consultando [**?**].

# 4. La construccin de mdulos

Ya es hora de comenzar a programar! En esta seccin se presentan conceptos sobre mdulos y programacin del ncleo de Linux 2.6. Mostraremos el cdigo de un mdulo completo (aunque poco til) y veremos el cdigo que comparten muchos mdulos.

## 4.1. Preparacin del sistema

La construccin de mdulos para el ncleo de Linux 2.6 requiere que tenga el rbol de fuentes de un ncleo configurado y construido en su sistema. Este requerimiento es un cambio desde versiones anteriores del ncleo en las cuales alcanzaba con tener los encabezados que correspondieran a la versin en uso. Los mdulos del ncleo 2.6 se enlazan con archivos objetos encontrados en rbol de fuentes del ncleo.

En los sistemas GNU/Linux basados en Debian, esto significa instalar el paquete `linux-headers-gen` o `linux-kernel-headers`.

## 4.2. Primer mdulo

Un mdulo para el ncleo de Linux agrega una funcionalidad al sistema, la cual puede corresponder al controlador de un componente de hardware o no. Es importante notar este punto ya que nuestro primer ejemplo es una clara muestra de que no es necesario que un mdulo haga nada con el hardware:

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

MODULE_LICENSE("GPL")
```

```
static int hello_init(void)
{
        printk(KERN_INFO "Hello, world\n");
        return 0;
}

void hello_exit(void) {
        printk(KERN_INFO "Goodbye, cruel world\n");
}

module_init(hello_init);
module_exit(hello_exit);
```

Este mdulo define dos funciones, una para ser invocada cuando se carga y activa el mdulo (*hello_init*) y otra para cuando el mdulo es removido y desactivado (*hello_exit*). Las macros *module_init* y *module_exit* sirven para indicar el rol de estas dos funciones. La macro MODULE_LICENSE es usada para indicar que este mdulo tiene una licencia libre; sin esta declaracin, el ncleo se queja cuando se carga el mdulo.

La funcin sealada por *module_init* debe contener todo el cdigo de inicializacin del mdulo. Como ya veremos ms adelante, en caso de que estemos implementando un controlador, aqu debemos realizar la registracin del dispositivo.

La funcin indicada por *module_exit* debe contener el cdigo de terminacin del mdulo. La posibilidad de descargar un mdulo con el sistema en ejecucin es una de las caractersticas ms apreciadas por los desarrolladores, ya que reduce el tiempo de desarrollo; uno puede probar distintas versiones del controlador sin necesidad de reiniciar el equipo cada vez que se realiza una nueva versin.

La funcin *printk* est definida en el ncleo de Linux y se comporta de manera similar a la funcin *printf* de la biblioteca estndar de C. La constante *KERN_INFO* indica la prioridad del mensaje.

¿Para qu necesitamos otra implementacin de una funcin que ya est en la biblioteca de C? Dado que el ncleo se ejecuta por s mismo y sin la ayuda de bibliotecas externas, entonces debe definir todas las funciones que le hagan falta. Por lo tanto, al no estar enlazado con ninguna biblioteca, el cdigo fuente de los mdulos *nunca* debe incluir los archivos de cabecera comunes.

Ahora podremos probar el mdulo utilizando las utilidades *insmod* y *rmmod*. Notar que slo el superusuario puede cargar y descargar mdulos.

```
[usuario@localhost]$ make
make -C /lib/modules/2.6.20-15-generic/build M=/path/to/src modules
make[1]: Entering directory ‘/usr/src/linux-headers-2.6.20-15-generic’
 CC [M]  /path/to/src/hello.o
 Building modules, stage 2.
 MODPOST 1 modules
 CC      /path/to/src/hello.mod.o
 LD [M]  /path/to/src/hello.ko
make[1]: Leaving directory ‘/usr/src/linux-headers-2.6.20-15-generic’

[root@localhost]# insmod ./hello.ko
Hello, World
[root@localhost]# rmmod hello
```

```
Goodbye, cruel world
```

El lector atento descubrir que hemos hecho uso de la herramienta *make* para poder construir el mdulo. En la prxima subseccin encontrar los detalles sobre cmo escribir Makefiles para el ncleo y sus mdulos. Para concluir, se presenta un resumen de las funciones usadas hasta el momento.

Cuadro 1: funcin de incializacin

| **Funcin** | `static int <nombre a eleccin>(void)` |
|---|---|
| **Sinopsis** | Punto de entrada de mdulos del ncleo. |
| **Descripcin** | Esta funcin debe estar implementada en todo mdulo y define el punto de entrada al realizarse su carga. En los controladores es la encargada de inicializar el dispositivo que maneja. |
| **Retorna** | En caso de xito: 0.<br>En caso de error: Cdigo de error apropiado. |

Cuadro 2: funcin de finalizacin

| **Funcin** | `static void <nombre a eleccin>(void)` |
|---|---|
| **Sinopsis** | Punto de salida de mdulos del ncleo. |
| **Descripcin** | Esta funcin debe estar implementada en todo mdulo y define el punto de salida al realizarse su descarga. En los controladores es la encargada de realizar el proceso de terminacin del dispositivo que maneja. |
| **Retorna** | Nada |

Cuadro 3: module_init

| **Macro** | `module_init` |
|---|---|
| **Sinopsis** | Seala el punto de entrada de mdulos del ncleo. |
| **Retorna** | Nada |

Cuadro 4: module_exit

| **Macro** | `module_exit` |
|---|---|
| **Sinopsis** | Seala el punto de salida de mdulos del ncleo. |
| **Retorna** | Nada |

## 4.3. Makefiles para mdulos

El proceso de construccin de un mdulo difiere significativamente de la construccin de un programa tradicional. El ncleo es un programa autnomo con ciertos requerimientos sobre cmo juntar todas sus piezas. El nuevo sistema de construccin es ms sencillo y simple que la versin 2.4. Mostramos a continuacin un Makefile de ejemplo, apto para mdulos con uno o ms archivos objeto.

```
KERNELDIR ?= "/lib/modules/$(shell uname -r)/build"
PWD := "$(shell pwd)"

obj-m := hello.o
hello-objs := hello.o  # Agregar otros archivos objeto

default:
$(MAKE) -C "$(KERNELDIR)" "M=$(PWD)" modules
```

Este archivo Makefile es muy flexible y fcilmente reusable ya que no tiene dependencia sobre particularidades del sistema donde se invoca.

## 4.4. Parmetros de mdulos

As como los programas de espacio usuario pueden tomar parmetros para modificar su comportamiento, el ncleo de Linux permite que los mdulos tambin sean ajustados mediante el paso de parmetros durante su carga (con *insmod* o *modprobe*):

```
[root@localhost]$ insmod mimodulo count=42
```

En el ejemplo anterior, el parmetro *count* del modulo *mimodulo* tomar el valor 42. En el cdigo, ese parmetro se declarar agregando estas lneas (adems de *<linux/stat.h>*):

```
static int count = 1; /* valor por defecto */
module_param(count, int, S_IRUGO); /* permiso de lectura para todo el mundo */
```

## 4.5. Creacin de dispositivos

Los dispositivos que el ncleo controla pueden corresponderse con artefactos fsicos o lgicos. Sin embargo, todos estn mapeados a algn nodo o archivo especial normalmente en el directorio */dev*. Por ejemplo, */dev/hda* corresponde a la unidad de disco duro IDE master primaria del equipo y */dev/urandom* es un dispositivo virtual que nos permite obtener nmeros aleatorios generados a partir del *"ruido"* del equipo.

Mediante el comando *ls* podemos ver los atributos de estos dispositivos:

```
[usuario@localhost]$ ls -l /dev/hda /dev/urandom
brw------- 1 root root 3, 0 2007-09-26 08:45 /dev/hda
crw-r--r-- 1 root root 1, 9 2007-09-26 08:45 /dev/urandom
```

En la primera columna podemos observar los permisos de acceso de los archivos y notamos una primer diferencia entre ambos. En el caso de */dev/hda* hay una *b* que nos indica que se trata de un dispositivo de bloques; la *c* en */dev/urandom* nos dice que es un dispositivo de caracteres.

En la quinta columna, encontramos lo que se ha dado en llamar el *major number* de un dispositivo. Este nmero es el que le indica al ncleo cul controlador es el encargado de manejar este dispositivo. Cada controlador en el sistema registra un *major number* y el ncleo se encarga

de llevar una tabla que los asocia. As, el sistema sabe fcilmente qu operaciones se pueden realizar sobre un dispositivo.

Finalmente, en la sexta columna se halla el *minor number* del dispositivo. Este nmero indica la instancia del dispositivo que se est accediendo. Por ejemplo, dos particiones de un disco duro son manejadas por el mismo controlador, pero son distintas instancias del dispositivo:

```
[usuario@localhost]$ ls -l /dev/hda*
brw------- 1 root root 3, 1 2007-09-26 08:45 /dev/hda1
brw------- 1 root root 3, 2 2007-09-26 08:45 /dev/hda2
```

Para crear las entradas en */dev* se utiliza el programa *mknod*. Su uso se da de la siguiente manera:

```
[usuario@localhost]$ mknod -m 0666 /dev/nombre c M m
```

El modificador *-m* indica el modo de acceso; la *c*, que es dispositivo de caracteres. *M* y *m* son el *major* y *minor* del dispositivo respectivamente.

Se puede encontrar una lista completa de la lista de dispositivos y sus *major* y *minor numbers* consultando */usr/src/linux/Documentation/devices.txt*.

# 5. Interfaz de programacin

El ncleo de Linux brinda una interfaz de programacin que debe ser implementada por cada controlador del sistema. As, los controladores se comportan como verdaderos tipos abstractos y pueden interactuar con el resto del sistema con gran flexibilidad y extensibilidad.

El desarrollador deber asociar el nmero *major* del dispositivo en */dev* a una estructura *file_operations* con punteros a cada una de las funciones que soporta el controlador. Dicha asociacin se debe realizar al inicializar el controlador. Cuando el controlador se descarga, es necesario deshacer esta asociacin para que el ncleo no intente invocar funciones que ya no estn disponibles.

A continuacin vemos las funciones que nos permiten acceder al dispositivo y liberarlos, asi como las que permiten enviar y recibir datos.

# 6. Espacio de Usuario vs. Espacio de Ncleo

Es muy importante resaltar que el argument *buf* de los mtodos *read* y *write* es un puntero de espacio usuario y no puede ser desreferenciado directamente por el ncleo.

Las direcciones de memoria del ncleo estn protegidas del alcance de los programas en espacio usuario. Por lo tanto, para leer y escribir datos desde estos programas es necesario usar un par de macros que hacen esta tarea posible.

# 7. Sincronizacin de procesos

Para solucionar los problemas que surgen con las condiciones de carrera, el ncleo nos provee de semforos. Los semforos de Linux tienen el tipo `struct semaphore` estn definidos en `<asm/semaphore.h>` y un controlador slo debe acceder a su estructura utilizando las primitivas provistas. Los semforos deben inicializarse antes de su uso pasando un valor numrico a *sema_init*.

Cuadro 5: register_chrdev

| | |
|---|---|
| **Funcin** | `int register_chrdev(unsigned int major, const char *name, struct file_operations *fops)` |
| **Cabecera** | `#include <linux/fs.h>` |
| **Sinopsis** | Realiza el registro de un dispositivo de caracteres. |
| **Descripcin** | Esta funcin debe ser llamada por todo controlador para realizar el registro de un dispositivo de caracteres. *major* es el nmero mayor del dispositivo que se quiere controlar (entre 0 y 255). Si es 0, el nmero es asignado dinmicamente por el ncleo. *name* es el nombre del dispositivo como aparecer en */proc/devices*. *fops* es la estructura que contiene referencias a las operaciones que se pueden realizar sobre el dispositivo. |
| **Retorna** | En caso de xito: 0 si *major* es distinto de 0; nmero de *major* en caso contrario.<br>En caso de error: -EBUSY si *major* ya ha sido solicitado por otro controlador -EINVAL si no es nmero *major* vlido. |

Cuadro 6: unregister_chrdev

| | |
|---|---|
| **Funcin** | `void unregister_chrdev(unsigned int major, const char *name)` |
| **Cabecera** | `#include <linux/fs.h>` |
| **Sinopsis** | Realiza el desregistro de un dispositivo de caracteres. |
| **Descripcin** | Esta funcin debe ser llamada por todo controlador para realizar el desregistro de un dispositivo de caracteres. *major* es el nmero mayor del dispositivo controlado (el mismo que se pas o se obtuvo en *register_chrdev*). *name* es el nombre del dispositivo como apareca en */proc/devices*. |
| **Retorna** | Nada |

Cuadro 7: open

| | |
|---|---|
| **Funcin** | `int open(struct inode *ip, struct file *fp)` |
| **Cabecera** | `#include <linux/fs.h>` |
| **Sinopsis** | Apertura de un dispositivo. |
| **Descripcin** | An cuando es la primera operacin que se realiza sobre un dispositivo, su implementacin no es obligatoria por parte de un controlador. En caso de no estar definida, el controlador no es avisado sobre la apertura del dispositivo, pero sta se realiza en forma satisfactoria. |
| **Retorna** | En caso de xito: 0.<br>En caso de error: Cdigo de error apropiado. |

Cuadro 8: release

| | |
|---|---|
| **Funcin** | `int release(struct inode *ip, struct file *fp)` |
| **Cabecera** | `#include <linux/fs.h>` |
| **Sinopsis** | Liberacin de un dispositivo. |
| **Descripcin** | Esta operacin se invoca una vez que se libera el dispositivo. Es decir, que no es llamada cada vez que un programa ejecuta la llamada al sistema *close*. Cada vez que una estructura es compartida (por ejemplo, despus de un *fork* o *dup*), *release* no ser invocado hasta que todas las copias estn cerradas. En caso de no estar definida, el controlador no es avisado sobre el cierre del dispositivo, pero sta se realiza en forma satisfactoria. |
| **Retorna** | En caso de xito: 0.<br>En caso de error: Cdigo de error apropiado. |

Cuadro 9: read

| | |
|---|---|
| **Funcin** | `ssize_t read(struct file *fp, char *buf, size_t length, loff_t *offset)` |
| **Cabecera** | `#include <linux/fs.h>` |
| **Sinopsis** | Obtiene datos desde un dispositivo. |
| **Descripcin** | Este mtodo es llamado cada vez que se intenta leer datos desde un dispositivo. Los datos ledos deben ser copiados en *buf*. Se debe prestar especial atencin en esto, ya que es un puntero a una direccin de memoria en espacio usuario. *length* es la cantidad de bytes que deben leerse y *offset* el desplazamiento desde el inicio del archivo. Si el mtodo no se define, al ser invocado se retorna -EINVAL. |
| **Retorna** | En caso de xito: La cantidad de bytes ledos  0 para indicar EOF.<br>En caso de error: Cdigo de error apropiado. |

Cuadro 10: write

| | |
|---|---|
| **Funcin** | `ssize_t write(struct file *fp, const char *buf, size_t length, loff_t *offset)` |
| **Cabecera** | `#include <linux/fs.h>` |
| **Sinopsis** | Escritura en un dispositivo. |
| **Descripcin** | Esta operacin enva datos al dispositivo. Los datos a escribir estn la direccin apuntada por *buf*. Se debe prestar especial atencin en esto, ya que es un puntero a una direccin de memoria en espacio usuario. *length* es la cantidad de bytes que deben escribirse y *offset* el desplazamiento desde el inicio del archivo. Si el mtodo no se define, al ser invocado se retorna -EINVAL. |
| **Retorna** | En caso de xito: La cantidad de bytes escritos  0 para indicar EOF.<br>En caso de error: Cdigo de error apropiado. |

Cuadro 11: copy_from_user

| | |
|---|---|
| **Funcin** | `unsigned long copy_from_user(void *to, const void *from, unsigned long count)` |
| **Cabecera** | `#include <asm/uaccess.h>` |
| **Sinopsis** | Copia memoria de espacio usuario a espacio ncleo. |
| **Descripcin** | Esta funcin se comporta como *memcpy* y copia memoria de espacio usuario (*from*) a espacio ncleo (*to*) *count* bytes. Si el puntero *to* es una referencia invlida, no se realiza copia alguna. Si durante la copia se encuentra una referencia invlida, se devuelve la cantidad de bytes que falta leer. |
| **Retorna** | En caso de xito: 0<br>En caso de error: La cantidad de bytes que restan por leer. |

Cuadro 12: copy_to_user

| | |
|---|---|
| **Funcin** | `unsigned long copy_to_user(void *to, const void *from, unsigned long count)` |
| **Cabecera** | `#include <asm/uaccess.h>` |
| **Sinopsis** | Copia memoria de espacio ncleo a espacio usuario. |
| **Descripcin** | Esta funcin se comporta como *memcpy* y copia memoria de espacio ncleo (*from*) a espacio usuario (*to*) *count* bytes. Si el puntero *to* es una referencia invlida, no se realiza copia alguna. Si durante la copia se encuentra una referencia invlida, se devuelve la cantidad de bytes que falta escribir. |
| **Retorna** | En caso de xito: 0<br>En caso de error: La cantidad de bytes que restan por escribir. |

Cuadro 13: sema_init

| | |
|---|---|
| **Funcin** | `void sema_init(struct semaphore *sem, int val)` |
| **Cabecera** | `#include <asm/semaphore.h>` |
| **Sinopsis** | Inicializa un semforo. |
| **Descripcin** | Inicializa el semforo *sem* con el valor *val*. |
| **Retorna** | Nada |

Si bien la primitiva *DOWN* est implementada con varios comportamientos, nosotros slo veremos la funcin *down_interruptible* que permite al proceso ser interrumpido mientras lleva a cabo la operacin. Si se interrumpe, el proceso no habr adquirido el semforo y entonces no ser necesario ejecutar *UP*.

Cuadro 14: P

| **Funcin** | `int down_interruptible(struct semaphore *sem)` |
|---|---|
| **Cabecera** | `#include <asm/semaphore.h>` |
| **Sinopsis** | Operacin P interrumpible. |
| **Descripcin** | Este mtodo realiza la operacin P sobre el semforo *sem* y permite que ste reciba seales en el transcurso de la llamada. |
| **Retorna** | En caso de xito: 0.<br>En caso de error: -EINTR (llamada interrumpida). |

Su uso est dado de la siguiente manera:

```
if (down_interruptible(&s)) {
        /* La llamada fue interrumpida */
        return -ERESTARTSYS;
}
```

La operacin *UP* est implementada con un nombre realmente muy sugestivo: *up* y su comportamiento es el tradicional.

Cuadro 15: V

| **Funcin** | `void up(struct semaphore *sem)` |
|---|---|
| **Cabecera** | `#include <asm/semaphore.h>` |
| **Sinopsis** | Operacin V. |
| **Descripcin** | Este mtodo realiza la operacin V sobre el semforo *sem*. |
| **Retorna** | Nada |

# 8. Entrada y salida sncrona

Un problema que puede surgir durante una lectura del dispositivo es que no haya datos en ese momento pero no hayamos llegado al fin del archivo. Es decir, se puede asegurar que llegarn datos.

La solucin a esto es "dormir esperando datos". Esta seccin muestra cmo dormir a un proceso, cmo despertarlo durante una lectura y despus aplicamos esos conceptos a la escritura.

## 8.1. Cmo dormir procesos

Cuando un proceso deba esperar que ocurra un evento, lo correcto ser ponerlo a dormir, suspendiendo as su ejecucin y liberando al procesador para otros usos. En algn momento posterior, cuando el evento ocurra, el ncleo despertar al proceso y podr continuar su tarea.

En Linux, hay varias formas de manejar la suspensin de procesos, cada una de ellas para un caso particular. Sin embargo, todas utilizan el mismo tipo de datos: una cola de espera (*wait_queue_head_t*). Las colas de espera son declaradas e inicializadas de la siguiente manera:

```
wait_queue_head_t cola;
init_waitqueue_head(&cola);
```

Cuando la cola de espera es declarada estticamente (o sea, no es automtica o dinmicamente alojada), tambin es posible declarar la cola e inicializarla en tiempo de compilacin:

```
DECLARE_WAIT_QUEUE_HEAD(cola);
```

La suspensin de procesos se consigue invocando alguna de estas variantes de *sleep_on*:

Cuadro 16: sleep_on

| | |
|---|---|
| **Funcin** | `sleep_on(wait_queue_head_t *queue)` |
| **Cabecera** | `#include <linux/wait.h>` |
| **Descripcin** | Pone a dormir el proceso en forma ininterrumpible. Por lo tanto, el proceso no podr recibir seales y puede quedar bloqueado si el evento que espera nunca ocurre. El argumento *queue* es una referencia a la cola donde dormir el proceso. |

Cuadro 17: interruptible_sleep_on

| | |
|---|---|
| **Funcin** | `interruptible_sleep_on(wait_queue_head_t *queue)` |
| **Cabecera** | `#include <linux/wait.h>` |
| **Descripcin** | Pone a dormir el proceso y permite que reciba seales. El argumento *queue* es una referencia a la cola donde dormir el proceso. |

Si bien estas primitivas son suficientes para conseguir el bloqueo de procesos, la forma ms deseable de suspender un proceso es haciendo que espere por el cumplimiento de una condicin booleana. Para esto existen un par de macros que se expanden a un ciclo while y por lo tanto aceptan cualquier expresin booleana de C vlida.

Cuadro 18: wait_event

| | |
|---|---|
| **Funcin** | `wait_event(wait_queue_head_t queue, boolean condicion)` |
| **Cabecera** | `#include <linux/wait.h>` |
| **Descripcin** | El argumento *queue* es la cola donde dormir el proceso hasta que se cumpla *condicion*. |

## 8.2. Cmo despertar procesos

Por ltimo, veamos cmo se despierta a los procesos que estn durmiendo en una cola de espera.

# Referencias


[1] Kwan Lowe, "*Kernel Rebuild Guide*", 2004.

[2] Jonathan Corbet, Alessandro Rubini y Greg Kroah-Hartman, "*Linux Device Drivers*", 3rd Edition, O'Rielly, 2005.

Cuadro 19: wait_event_interruptible

| | |
|---|---|
| **Funcin** | `int wait_event_interruptible(wait_queue_head_t queue, boolean condicion)` |
| **Cabecera** | `#include <linux/wait.h>` |
| **Descripcin** | El argumento *queue* es la cola donde dormir el proceso hasta que se cumpla *condicion*. |
| **Retorna** | En caso de xito: 0.<br>En caso de error: -ERESTARTSYS si el proceso es interrumpido por una seal. |

Cuadro 20: wake_up

| | |
|---|---|
| **Funcin** | `wake_up(wait_queue_head_t *queue)` |
| **Cabecera** | `#include <linux/wait.h>` |
| **Descripcin** | Despierta a todos los procesos que duermen en la cola *queue*. Causa una inmediata replanificacin del CPU y por lo tanto un nuevo proceso puede ejecutarse antes de que retorne. |

Cuadro 21: wake_up_interruptible

| | |
|---|---|
| **Funcin** | `wake_up_interruptible(wait_queue_head_t *queue)` |
| **Cabecera** | `#include <linux/wait.h>` |
| **Descripcin** | Despierta a todos los procesos que duermen en la cola *queue* y que estn en estado interrumpible. Si un proceso espera por una condicin en un ciclo, no ser despertado. Causa una inmediata replanificacin del CPU y por lo tanto un nuevo proceso puede ejecutarse antes de que retorne. |

Cuadro 22: wake_up_interruptible_sync

| | |
|---|---|
| **Funcin** | `wake_up_interruptible_sync(wait_queue_head_t *queue)` |
| **Cabecera** | `#include <linux/wait.h>` |
| **Descripcin** | Despierta a todos los procesos que duermen en la cola *queue* y que estn en estado interrumpible. Si un proceso espera por una condicin en un ciclo, no ser despertado. Modifica el estado de los procesos suspendidos, pero no replanifica el CPU. |