Reporte de nueva Llamada al sistema al Kernel 3.11.8

Raúl García Cortés
(contacto@raulgc.mx) (www.llamada-kernel-3118.raulgc.mx)

¿Qué es una llamada al sistema?

Es un mecanismo usado por una aplicación que consiste en un método o una función que puede invocar un proceso para solicitar un servicio al sistema operativo.

En forma muy simple, podemos definir a el sistema operativo como el medio de comunicación o interfaz entre el hardware y los programas de usuario ya que es quien directamente se encarga de administrar los recursos disponibles en la maquina como dispositivos, memoria, etc y la comunicación entre estos.

Como los programas de usuario pueden presentar errores e incluso estar desarrollados para ocasionar daños en el sistema, se hace necesario establecer dos modos de operación diferentes para el procesador: el modo núcleo (Kernel Mode) y el modo usuario (User Mode), de esta manera el procesador está continuamente alternando entre un modo de operación núcleo y usuario.

En el modo núcleo se tienen a disposición todas las instrucciones y funcionalidades que proporciona la arquitectura del procesador, siendo el núcleo el único que opera en este modo.

El modo usuario solamente dispone de un conjunto muy reducido de las instrucciones y funcionalidades que ofrece la arquitectura y es el modo en el cual operan las aplicaciones de usuario.

El núcleo (o kernel) parte esencial del sistema operativo es quien está en contacto directo con todo el hardware de la máquina y por lo tanto es el único que puede manipularlos, como por ejemplo leer o escribir en el disco... Por lo tanto, si un programa de usuario desea realizar algún proceso sobre el hardware debe solicitarlo al núcleo.

El medio que permite la comunicación (o interfaz) entre los procesos de un programa de usuario y el núcleo es lo que se denomina peticiones o llamadas al sistema , también llamadas en inglés como system call o syscall las cuales se encuentran comúnmente implementadas como librerías de C, así, cuando un proceso se comunica con el núcleo por medio de una syscall, es el núcleo quien toma el control de la solicitud, la verifica, la ejecuta y después de realizado el proceso, vuelve a retornar el control al proceso que hizo la solicitud.

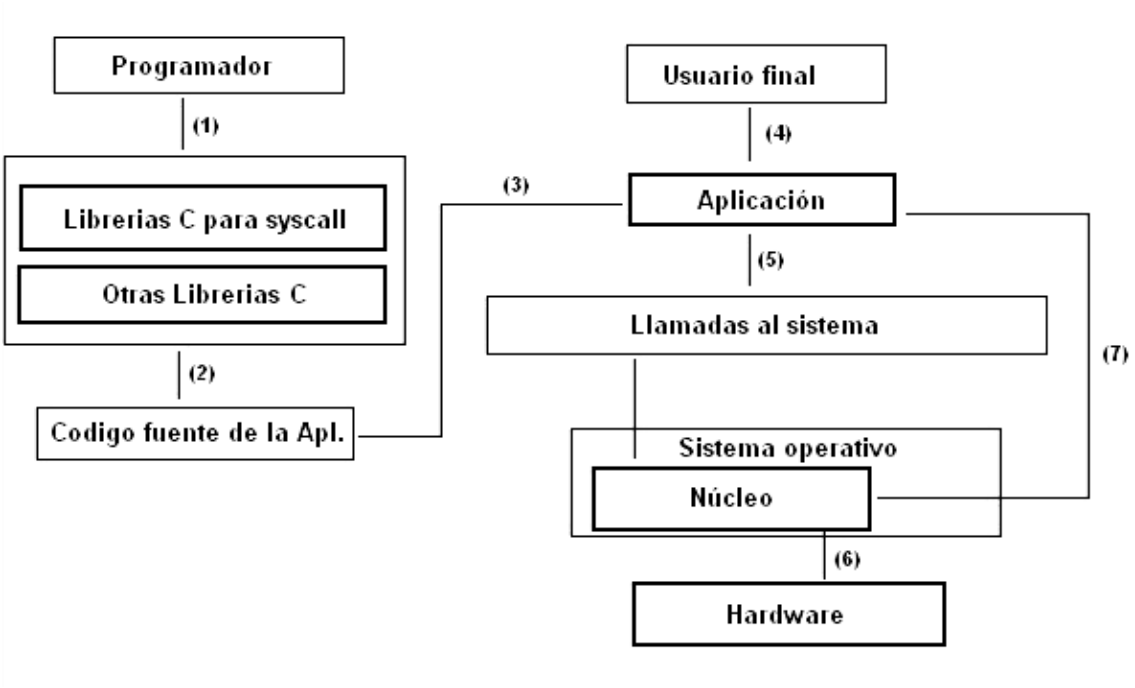
Para hacer uso de las llamadas al sistema desde el lenguaje de programación C, los sistemas operativos que trabajan con el núcleo Linux ponen a

disposición del usuario varias funciones o procedimientos de librería que representan a las llamadas del sistema. Los prototipos relativos a estas funciones o procedimientos pueden encontrarse listados en el archivo de cabecera unistd.h (este se encuentra en el directorio /usr/include/asm/, aquí también pueden encontrarse los archivos unistd_32.h y unistd_64.h, archivos relativos a las arquitecturas de 32 y 64 bits respectivamente.)

El sistema operativo de núcleo Linux cuenta con aproximadamente 200 funciones relacionadas con cada llamada al sistema, algunos de los cuales pueden agruparse en ciertas categorías que permiten el manejo o control de: procesos, señales, archivos, tiempo, etc.

¿Cómo funciona una llamada al sistema?

Los conceptos explicados anteriormente pueden resumirse con el siguiente esquema:



El programador (1): usa las librerías C para llamadas al sistema y otras librerías (2): para implementar el código fuente de la aplicación (3): y a partir de él generar el archivo ejecutable de dicha aplicación.

El usuario final (4): ejecuta la aplicación, (5): la cual se comunica a través de llamadas al sistema con el núcleo del sistema operativo (6): el cual toma el control y se encarga de manipular el hardware para realizar el proceso solicitado.

Cuando el proceso se ha completado, el núcleo (7): retorna el control a la aplicación.

a) Kernel implementado
La versión utilizada del kernel es la última versión estable hasta noviembre de 2013: 3.11.8

Para comenzar se instala CentOS en su versión 6.4 de 32 bits, posteriormente se descargan e instalan las actualizaciones correspondientes. Luego se instalan los programas necesarios:

Compiladores GNU C / C+

```
yum install gcc gcc-c++ autoconf automake
```

Herramientas y Librerías de Desarrollo

```
yum install ncurses-*
```

b) Forma de descompresión
Se descargan los fuentes del kernel 3.11.8:

```
wget http://www.kernel.org/pub/linux/kernel/v3.0/linux-3.11.8.tar.xz
```

Y se descomprime en el directorio del área de trabajo:

```
tar -Jxvf linux-3.11.8.tar.xz /usr/src/
```

-J: Descomprime el archivo con bzip2.
-x: Extrae los archivos.
-v: Muestra por pantalla las operaciones que va realizando archivo por archivo.
-f: indica a tar que el siguiente argumento es el nombre del fichero.

c) Llamada al sistema
A continuación se añaden los códigos fuente de la nueva llamada al sistema en el directorio emphkernel:

```
cd /usr/src/linux-3.11.8/kernel
```

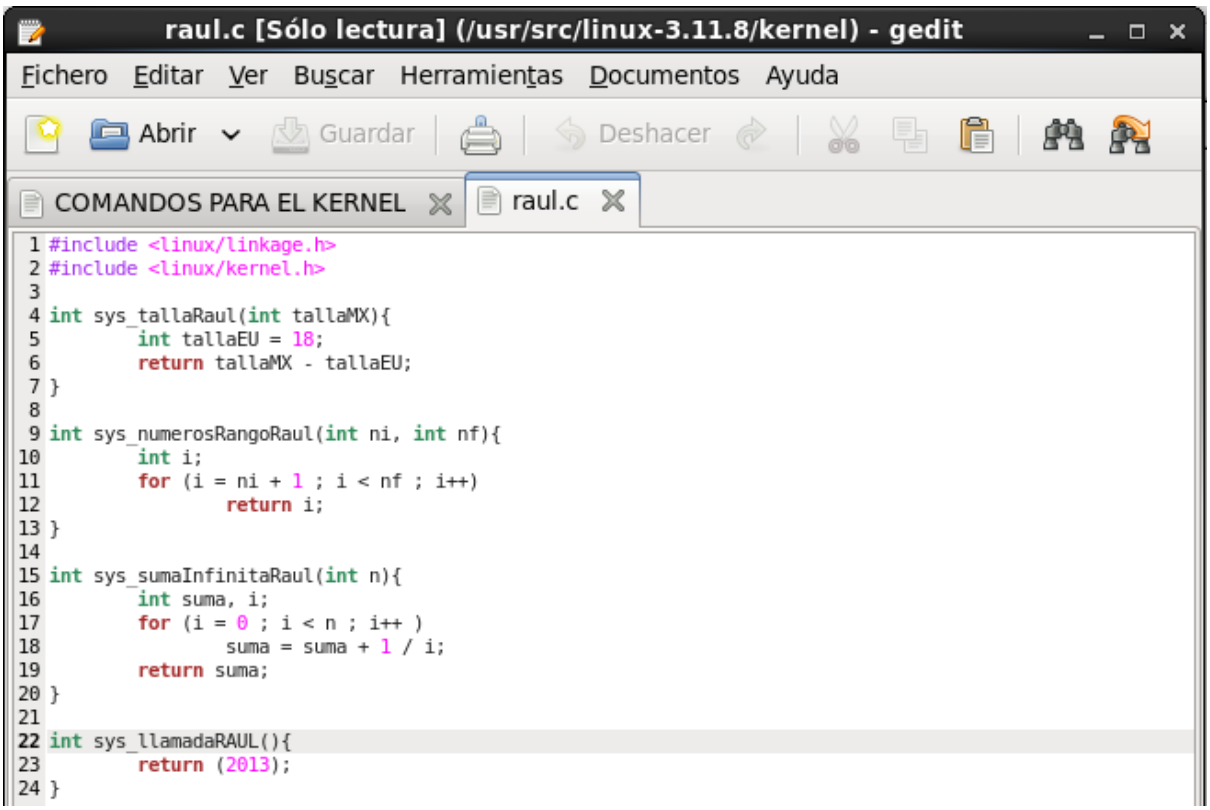
Se crea un archivo en blanco cuyo contenido incluye el código el lenguaje C:

```
gedit raul.c &
```

```
#include <linux/linkage.h>
#include <linux/kernel.h>
int sys_llamadaRAUL(){
      return (2013);
}
```

El header **<linux/linkage.h>** contiene la definición de la macro asmlinkage que se encarga de definir la función como visible afuera del archivo en donde se define. Asimismo, el header **<linux/kernel.h>** contiene definiciones para funciones utilitarias como printk.

Por otro lado, cualquier llamada al sistema debe ser nombrada con el prefijo "**sys**_" de lo contrario el Kernel no compilará correctamente. En este caso, la llamada al a sistema se llama **llamadaRAUL**, por lo que la función implementada en el kernel se llama **sys_llamadaRAUL**.

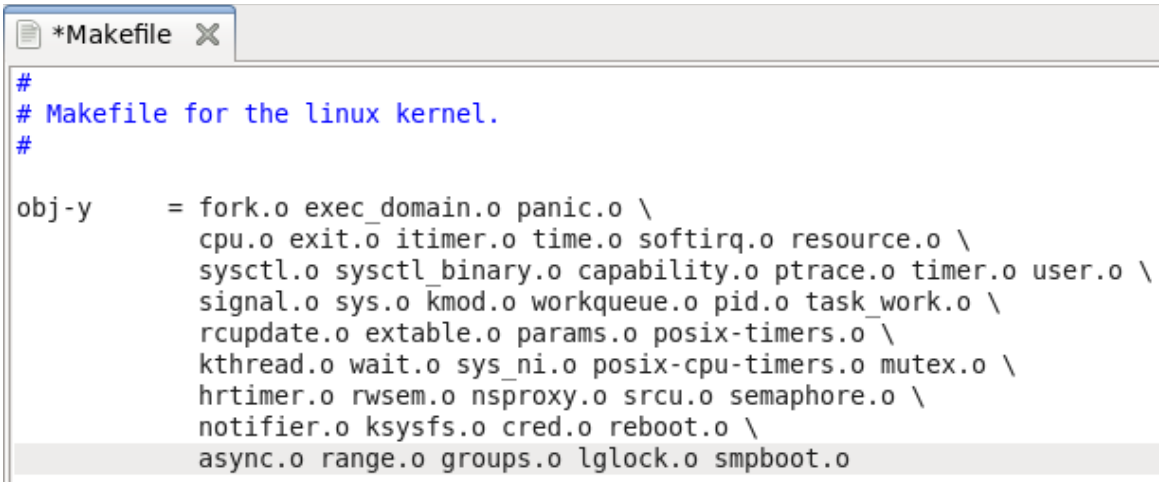


d) Archivos modificados

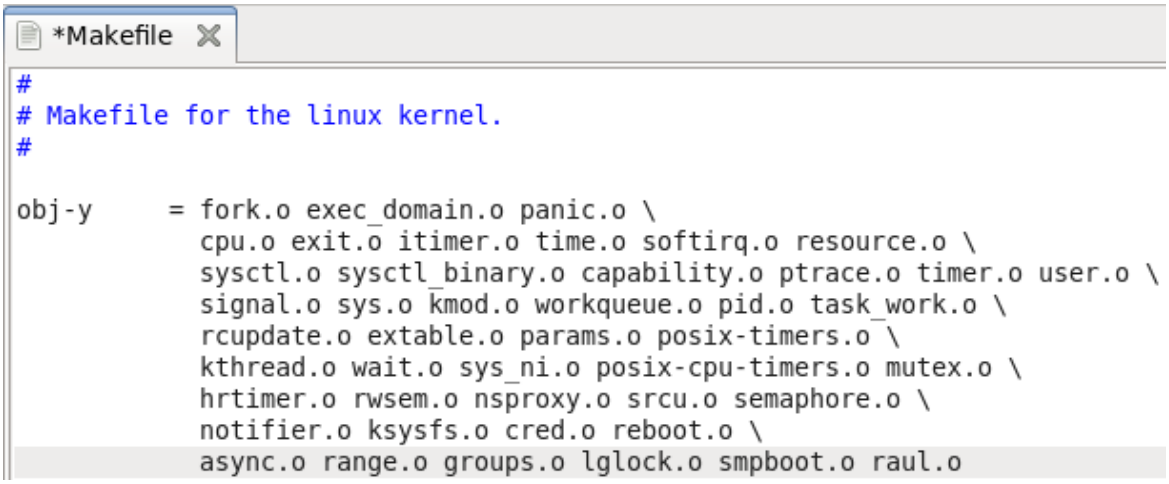
Una vez que se insertó el código de la llamada, se debe modificar el "**Makefile**" de esta carpeta para que el archivo recién creado se compile.:

```
gedit Makefile &
```

ANTES:



DESPUÉS:



Es importante destacar que se añadió el archivo creado anteriormente (raul.c) con extensión **.o** al final de la lista **obj-y**, pues el proceso de compilación sabrá que para generar ese **.o** hay que compilar el **.c** que tiene el mismo nombre.

Ahora se debe registrar la nueva llamada al sistema en el kernel, en la carpeta *syscalls*, de la arquitectura correcta.

```
cd ..
cd arch/x86/syscalls
ls
```

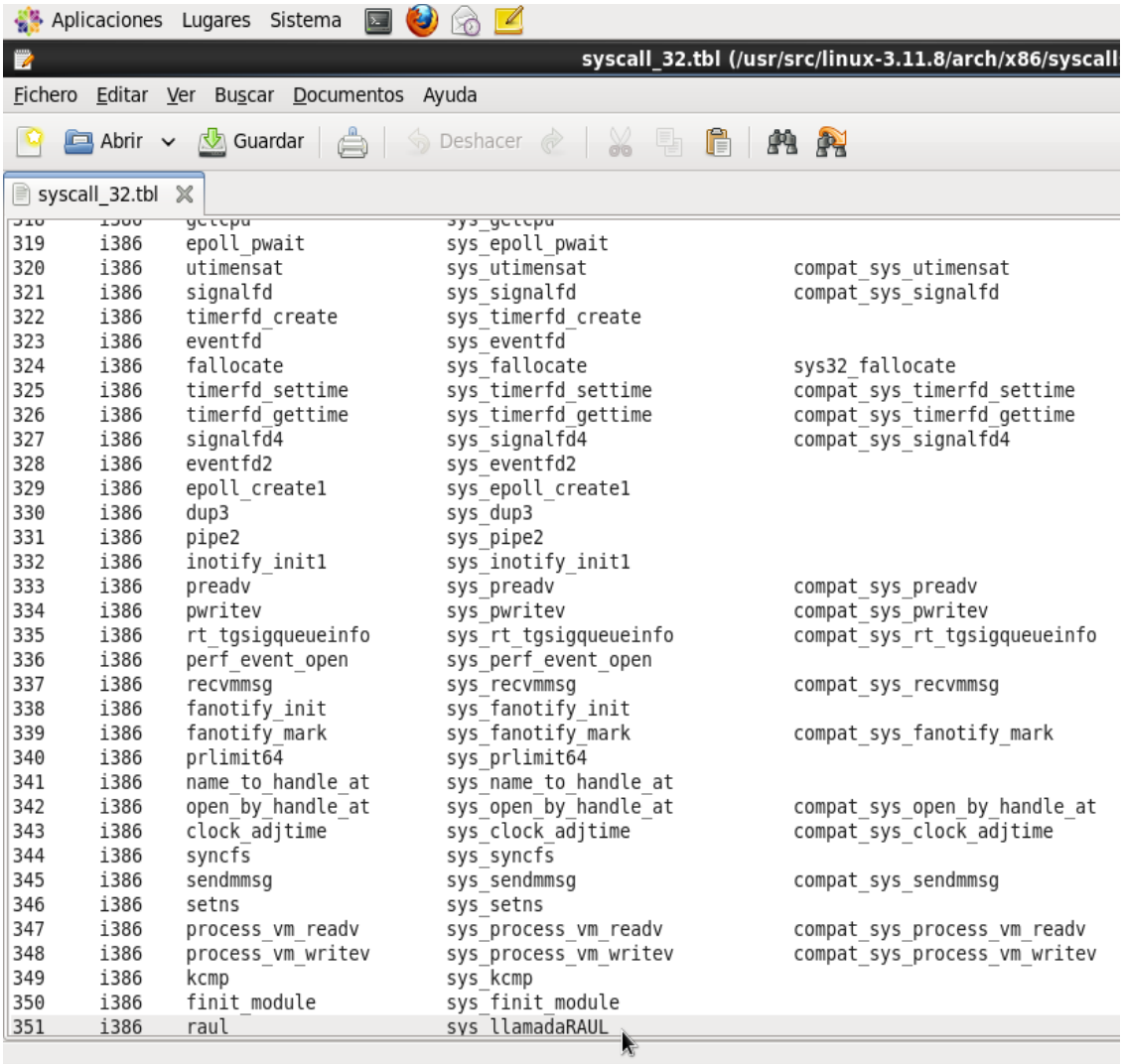
En esta carpeta es posible notar 2 archivos importantes **syscall_32.tbl** y **syscall_64.tbl.** En estos archivos se definen los códigos de llamadas al sistema para cada arquitectura (32 y 64 bits respectivamente).

En este caso será editado **syscall_32.tbl**.

```
gedit syscall_32.tbl &
```

Casi al final del archivo, se debe añadir la línea correspondiente a la nueva llamada a sistema, junto con su número, nombre de la llamada y la función del kernel que la implementa:

núm. *arqu.* *nombre* *función*
`351  i386  raul  sys_llamadaRAUL`



e) Forma de compilación

Una vez terminada la modificación de los archivos, se procede a compilar el kernel, regresando al directorio principal de la versión:

```
cd ../../..
```

NOTA: Si se ha compilado ya una versión anterior y aún permanecen los archivos, será mejor aplicar el siguiente comando para eliminar todo lo anterior:

```
make mrproper && make clean
```

make mrproper Deja las fuentes del kernel limpias e impolutas. Borran hasta los ficheros ocultos, incluyendo .config y .depend. Este comando sirve para asegurarse de que todo está realmente limpio, cuando se desee rehacer una compilación de raíz.

make clean Limpia las fuentes del kernel eliminando todos los archivos ***.o** encontrados. Este comando se usa en caso de error para limpiar los archivos intermedios y volver a comenzar o cuando se comenzará a recompilar un kernel para asegurarse de que no queden restos indeseados de la última compilación. El archivo **.config** no se toca, ni **.depend** tampoco, pero como sí desaparecen el **Makefile** y otros archivos deberán volver a correr **make dep** tras cargar o retocar la configuración.

Primero hay que copiar las configuraciones de compilación del Kernel que se está ejecutando, para que se compilen en el Kernel modificado :

```
make oldconfig
```

make oldconfig Usa el mismo archivo de configuración que la última vez, pero sin abrir menús, si no hay ninguno anterior inicia la configuración en modo pregunta-respuesta en la terminal.

NOTA: Es posible que éste comando solicite configuraciones para el nuevo kernel, pero se pueden elegir las predefinidas manteniendo pulsada la tecla “Enter” durante 30 segundos aproximados.

```
scripts/kconfig/conf --oldconfig Kconfig
#
# using defaults found in /boot/config-2.6.32-358.23.2.el6.i686
#
/boot/config-2.6.32-358.23.2.el6.i686:590:warning: symbol value 'm' invalid for PCCARD_NONSTATIC
/boot/config-2.6.32-358.23.2.el6.i686:2824:warning: symbol value 'm' invalid for MFD_WM8400
/boot/config-2.6.32-358.23.2.el6.i686:2825:warning: symbol value 'm' invalid for MFD_WM831X
/boot/config-2.6.32-358.23.2.el6.i686:2826:warning: symbol value 'm' invalid for MFD_WM8350
/boot/config-2.6.32-358.23.2.el6.i686:2839:warning: symbol value 'm' invalid for MFD_WM8350_I2C
/boot/config-2.6.32-358.23.2.el6.i686:2841:warning: symbol value 'm' invalid for AB3100_CORE
*
* Restart config...
*
*
* General setup
*
Cross-compiler tool prefix (CROSS_COMPILE) [] (NEW)
```

Y ahora se ejecutan los siguientes comandos para comenzar la compilación:

```
make dep && make bzImage && make && make install && make modules && make modules_install
```

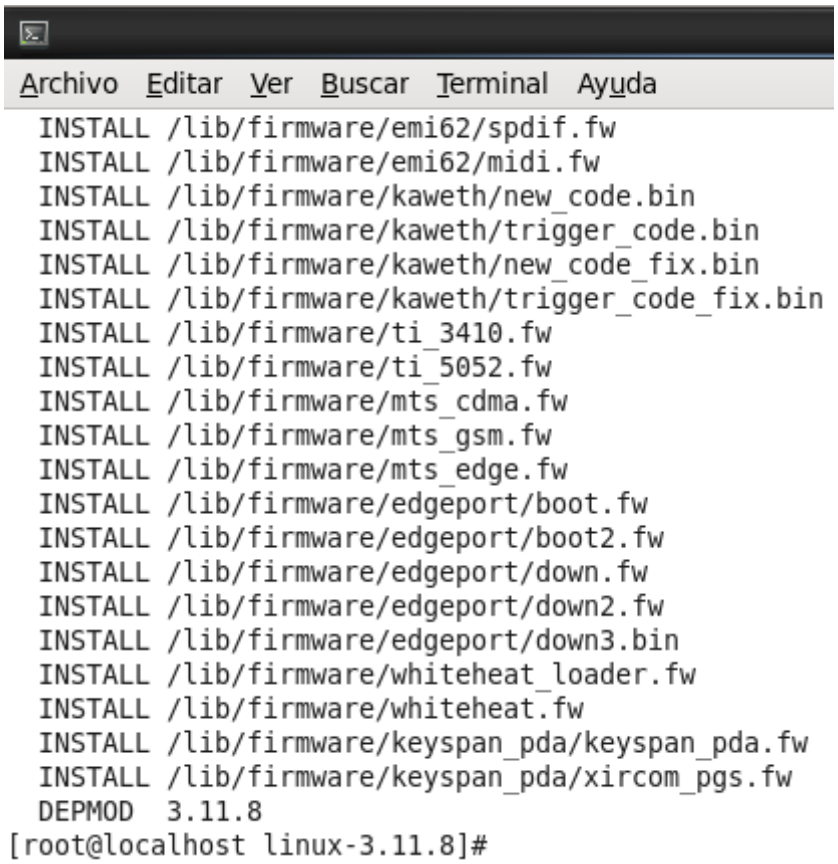
make dep Crea las **dependencias**; esto quiere decir que crea un archivo de configuración oculto llamado **.depend** que contiene las indicaciones para las herramientas de compilación. Allí se especifican, por ejemplo, dónde están las cabeceras de las librerías del sistema, dónde está el ensamblador y qué flags usarán las diferentes herramientas (especificaciones para cada arquitectura, etc).

make bzImage Crea la imagen **comprimida del kernel**, o sea; el kernel mismo. El archivo generado se guarda en /usr/src/linux/arch/i386/boot y se llama bzImage

make modules Compila los módulos, que son en realidad ficheros objeto (fichero.o).

make modules_install Se copian los módulos a a /lib/modules/versión_del_kernel

NOTA: El proceso de compilación en un PC puede demorar entre 30 minutos y 1 hora, o en una máquina virtual entre 3 y 5 horas, dependiendo de las características de hardware del equipo y los programas que además se estén ejecutando.



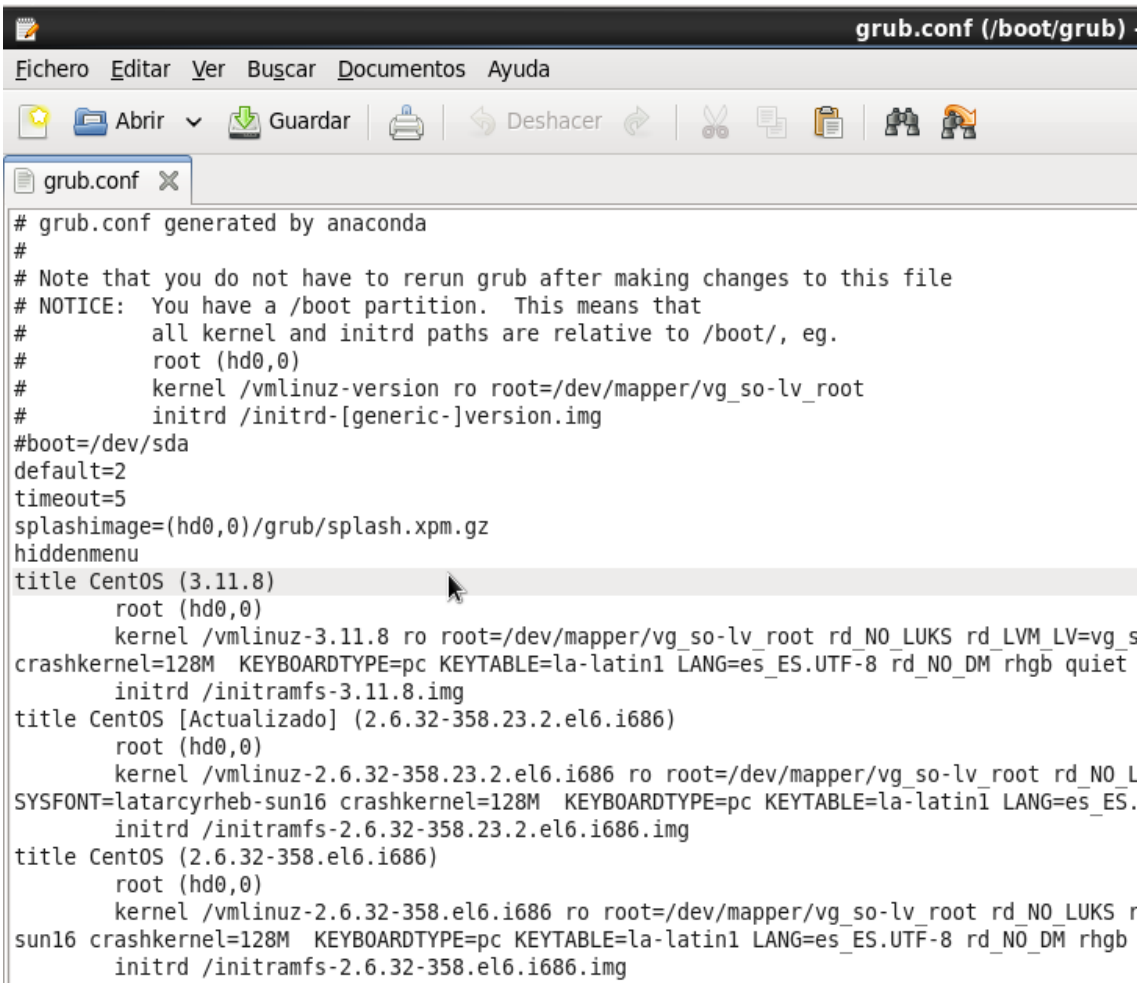
Para continuar, hay que entrar a la carpeta boot del sistema actual y crear la imagen de los módulos de los dispositivos:

```
cd /boot
mkinitrd -f initrd-3.11.8.img  3.11.8
```

mkinitrd Crea una imagen que usa el kernel para cargar los módulos de los dispositivos que son necesarios para acceder al root filesystem. Mkinitrd carga los módulos del sistema de ficheros, los IDE modules y todas las entradas relacionadas con los scsi hostadapter del /etc/modprobe.conf y otros módulos.
La forma de uso es mkinitrd imagen.img version.del.kernel.

Después, es importante confirmar que las configuraciones del nuevo kernel se han cargado al GRUB:

```
gedit /boot/grub/grub.conf &
```



Ahora hay que **actualizar las dependencias** y la **base de datos de los módulos**. Para esto se usa:

```
depmod -ae
```

depmod escanea los módulos en los subdirectorios de /lib/modules para buscar el kernel en el que usted está trabajando y actualiza la información sobre dependencia.
Puede manejar la carga automática de múltiples módulos cuando algunos dependen de otros. Las dependencias se conservan en el archivo modules.dep en el subdirectorio /lib/modules para el kernel correcto, según lo determinado por el comando uname -r. Este archivo, junto a varios archivos map, es generado por el comando depmod. La -a(porall [“todo” en inglés]) ahora es opcional.

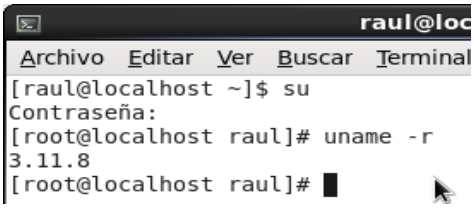
Y para terminar, es necesario reiniciar el equipo:

```
shutdown -r now
```

Al iniciar CentOS, se debe elegir el kernel recién creado, cuyo título debió ser “CentOS (3.11.8)”, donde el kernel es 3.11.8, que puede ser comprobado con el comando:

```
su
uname -r
```

su Se utiliza para acceder a la terminal como Super Usuario, comúnmente usado para acceder como Administrador del Sistema.
uname Muestra información sobre el sistema y el kernel
-r Muestra la edición del kernel.



O dirigirse al “Menú Sistema > Acerca de esta Computadora”

Con ambos métodos, se puede apreciar el resultado “**3.11.8**”.

f) Llamada al sistema con un programa en lenguaje C

Para comprobar que el nuevo kernel ya tiene implementada la llamada al sistema que se creó desde un inicio, puede probarse este código:

prueba.c

```
#include <linux/unistd.h>
#include <stdio.h>
#include <sys/syscall.h>
#include <errno.h>


int main(){
        int aux ;
        printf("\n\n ---------- Ejemplo de llamada al kernel especial ---------- \n") ;
        aux = syscall(351) ;
        printf(" > Retorno de la llamada: = %d , ErrNo = %d " , aux , errno ) ;
        printf("\n\n");
}
```

Hay que observar que usa la primitiva de la **libc syscall**, la cual se encarga hacer la llamada al sistema con el código que se le pasa de parámetro (en este caso **351**, la llamada que se implementó, que retornará el número 2013 cuando se mande a llamar).

Para compilar el programa, se usan los siguientes comandos:

```
gcc prueba.c –o Prueba
```

Si todo salió bien, al compilar este programa y ejecutarlo, deberá dar el siguiente resultado:

```
./Prueba
---------- Ejemplo de llamada al kernel especial ----------
 > Retorno de la llamada: = 2013 , ErrNo = 0
```

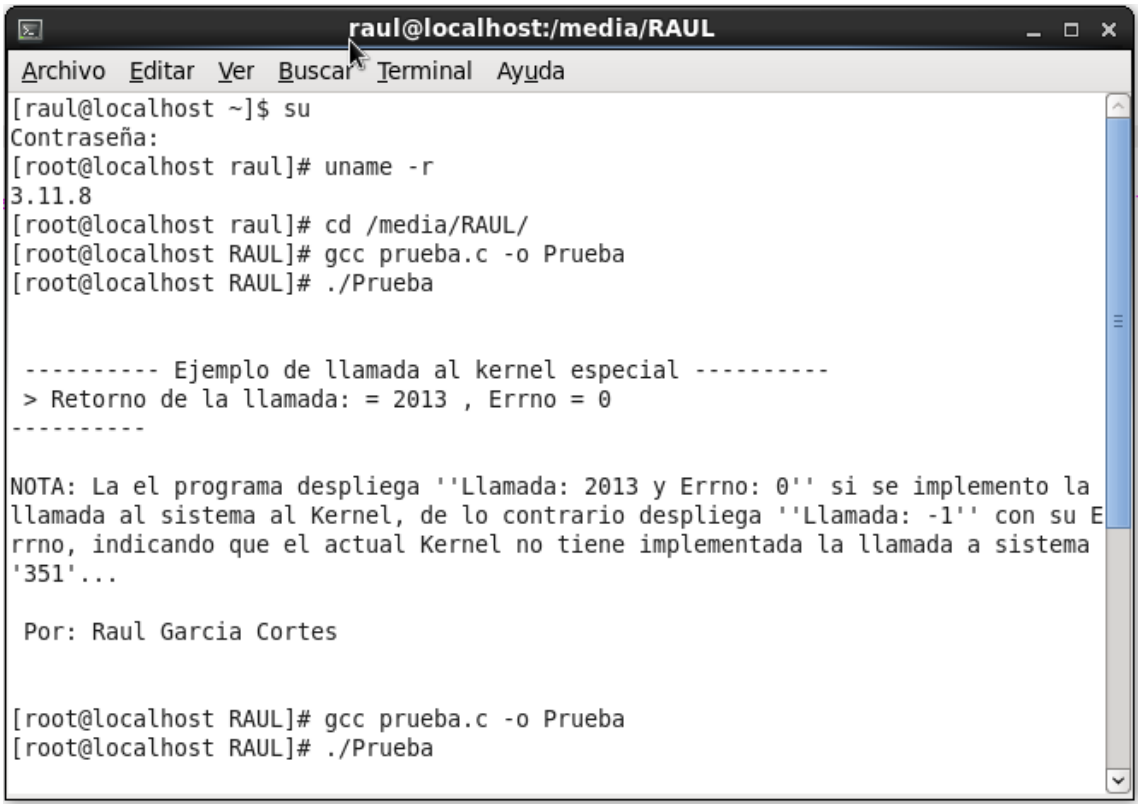
lo que indica que la llamada está funcionando y haciendo lo que debe .

Si se compila y ejecuta el mismo programa en un kernel que no tiene implementada esta llamada a sistema, o se cambia el número 351 por otro, la salida será semejante a:

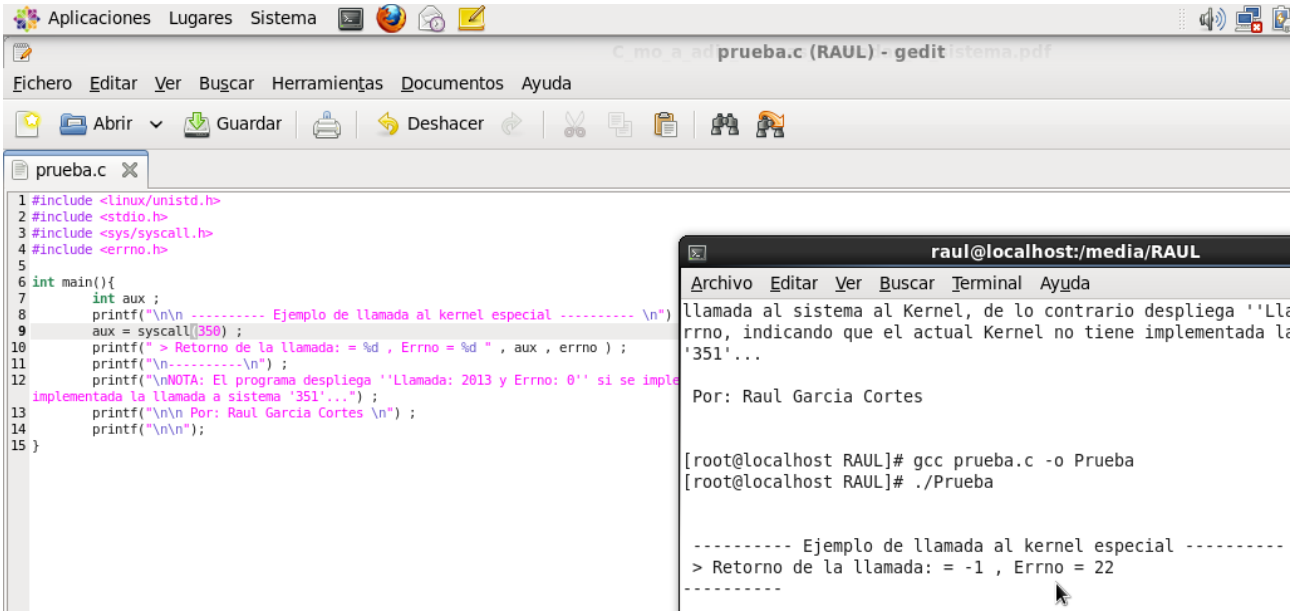
```
./Prueba
---------- Ejemplo de llamada al kernel especial ----------
 > Retorno de la llamada: = -1 , ErrNo = 38
```

```
./Prueba
---------- Ejemplo de llamada al kernel especial ----------
 > Retorno de la llamada: = -1 , ErrNo = 22
```

Invocación correcta:



Invocación incorrecta:



en donde syscall retornó -1, y errno tiene el valor 38 u otro número, que significa *function not implemented* (*función no implementada*) u otro error semejante.

ANEXO 1

Ejemplo de código fuente en lenguaje C para invocar la llamada al sistema usada en este ejemplo.

```
#include <linux/unistd.h>
#include <stdio.h>
#include <sys/syscall.h>
#include <errno.h>

int main(){
        int aux ;
        printf("\n\n ---------- Ejemplo de llamada al kernel especial ---------- \n") ;
        aux = syscall(351) ;

        printf(" > Retorno de la llamada: = %d , ErrNo = %d " , aux , errno ) ;
        printf("\n----------\n") ;

        printf("\nNOTA: El programa despliega ''Llamada: 2013 y ErrNo: 0'' si se
implemento la llamada al sistema al Kernel, de lo contrario despliega ''Llamada: -1''
con su ErrNo, indicando que el actual Kernel no tiene implementada la llamada a sistema
'351'...") ;

        printf("\n\n Por: Raul Garcia Cortes \n\n\n") ;
}
```

REFERENCIAS BIBLIOGRÁFICAS

“3. LLamadas al sistema”, David Esteban Bustamante Tabares, https://sites.google.com/site/sogrupo15/3-llamadas-al-sistema

“Llamada al sistema”, Wikipedia, http://es.wikipedia.org/wiki/Llamada_al_sistema

“Como añadir una nueva llamada al sistema en Linux 3.5”, Diego Arturo Guillermo Alejandro Rivera Villagra, http://1984.lsi.us.es/wiki-ssoo/index.php/Llamadas_al_sistema

“Empaquetando y comprimiendo”, Ciberaula España, http://linux.ciberaula.com/articulo/linux_shell_parte3/

www.llamada-kernel-3118.mex.tl