Universidad Autónoma de Chihuahua

UACH
UNIVERSIDAD AUTÓNOMA DE
CHIHUAHUA

Trabajo Sistemas Operativos 1

final:

Explicación módulo de Kernel

Iván Miguel Chavero Jurado

Jesús Alan González Murillo; 357600

Angel Torres Loya; 365354

Efraín Gilberto Domínguez Banda; 348453

05/06/2024

Explicación del módulo de Kernel

Código fuente de nuestro modulo

Comenzamos definiendo las partes del código fuente de nuestro Kernel.

Archivos de cabecera y algunas definiciones que forman parte del boiler plate de cualquier módulo de Kernel.

```c
Modulo > C Bohemian_Rhapsody.c > ...
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/fs.h>
#include <linux/cdev.h>
#include <linux/miscdevice.h>

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Angel");
MODULE_DESCRIPTION("Farrokh Bulsara");
MODULE_VERSION("0.01");

#define DEVICE_NAME "Bohemian_Rhapsody"
```

Prototipos de nuestras funciones que son funciones básicas de Kernel: abrir, leer, escribir y cerrar.

```c
static int Bohemian_Rhapsody_open(struct inode*, struct file*);
static int Bohemian_Rhapsody_release(struct inode*, struct file*);
static ssize_t Bohemian_Rhapsody_read(struct file*, char*, size_t, loff_t*);
static ssize_t Bohemian_Rhapsody_write(struct file*, const char*, size_t, loff_t*);
int write_to_user(char *, char *);
```

Especificábamos mediante la estructura file_operations de cuales serán las funciones que ha de ejecutar cuando quiera realizar cada una de las funciones básicas de archivo.

```c
static struct file_operations fops = {
    .owner = THIS_MODULE,
    .open = Bohemian_Rhapsody_open,
    .release = Bohemian_Rhapsody_release,
    .read = Bohemian_Rhapsody_read,
    .write = Bohemian_Rhapsody_write,
};
```

La definición de nuestras funciones.

```c
static int __init Bohemian_Rhapsody_init(void) {
    int result;
    major = register_chrdev(0, DEVICE_NAME, &fops);

    if (major < 0) {
        printk(KERN_ALERT "Bohemian_Rhapsody Module load failed\n");
        return major;
    }

    printk(KERN_INFO "Bohemian_Rhapsody Module has been loaded\n");

    printk(KERN_INFO "I was assigned major number %d. To talk to\n", major);
    printk(KERN_INFO "the driver, create a dev file with\n");
    printk(KERN_INFO "'mknod /dev/%s c %d 0'.\n", DEVICE_NAME, major);
    printk(KERN_INFO "Try various minor numbers. Try to cat and echo to\n");
    printk(KERN_INFO "the device file.\n");
    printk(KERN_INFO "Remove the device file and module when done.\n");

    /*cdev_init(&char_device, &fops);…
    return 0;
}

static void __exit Bohemian_Rhapsody_exit(void) {
    printk(KERN_INFO "Ya no hay rola, removiendo módulo!!\n");
}

static int Bohemian_Rhapsody_open(struct inode *inodep, struct file *filep) {
   printk(KERN_INFO "Super Module device opened\n");
   return 0;
}
```

```c
static ssize_t Bohemian_Rhapsody_write(struct file *filep, const char *buffer, size_t len, loff_t *offset) {
    //int errors = 0;
    static char message[256] = {0};
    if (copy_from_user(message, (void *)buffer, len))
        return -EFAULT;
    /*if (len > 1024) {…
    strcpy(user_message, message);
    printk(KERN_INFO "From userland: %s\n", (char *)message);
    return len;
}

static int Bohemian_Rhapsody_release(struct inode *inodep, struct file *filep) {
   printk(KERN_INFO "Super Module device closed\n");
   return 0;
}

static ssize_t Bohemian_Rhapsody_read(struct file *filep, char *buffer, size_t len, loff_t *offset) {
    int errors = 0;
    char *message = "Is this the real life? Is this just fantasy?\nCaught in a landslide, no escape from reali
    int message_len = strlen(message);
    errors = copy_to_user(buffer, message, message_len);
    printk(KERN_INFO "user_message: %s\n", message);
    return errors == 0 ? message_len : -EFAULT;
}

module_init(Bohemian_Rhapsody_init);
module_exit(Bohemian_Rhapsody_exit);
```

- Bohemian_Rhapsody_init: nos imprimirá por consola información básica de nuestro módulo de Kernel. Registra el dispositivo de caracteres y asigna un número mayor. Añade el dispositivo de caracteres al sistema.
- Bohemian_Rhapsody_exit: Desregistra el dispositivo y libera recursos.
- Bohemian_Rhapsody_open: Simplemente imprime un mensaje al abrir el dispositivo.
- Bohemian_Rhapsody_release: Imprime un mensaje al cerrar el dispositivo.
- Bohemian_Rhapsody_read: Copia un mensaje predefinido al espacio de usuario.
- Bohemian_Rhapsody_write: Copia un mensaje del espacio de usuario al Kernel y lo guarda en user_message.

El registro y desregistro del módulo se realiza mediante

```
106  module_init(Bohemian_Rhapsody_init);
107  module_exit(Bohemian_Rhapsody_exit);
```

Makefile

Este se encarga de automatizar el proceso de creación y compilación del módulo de Kernel.

```
obj-m += Bohemian_Rhapsody.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

old:
    make -C /lib/modules/6.1.13-200.fc37.x86_64/build M=$(PWD) modules
```

Nuestro modulo se compone de las siguientes partes:

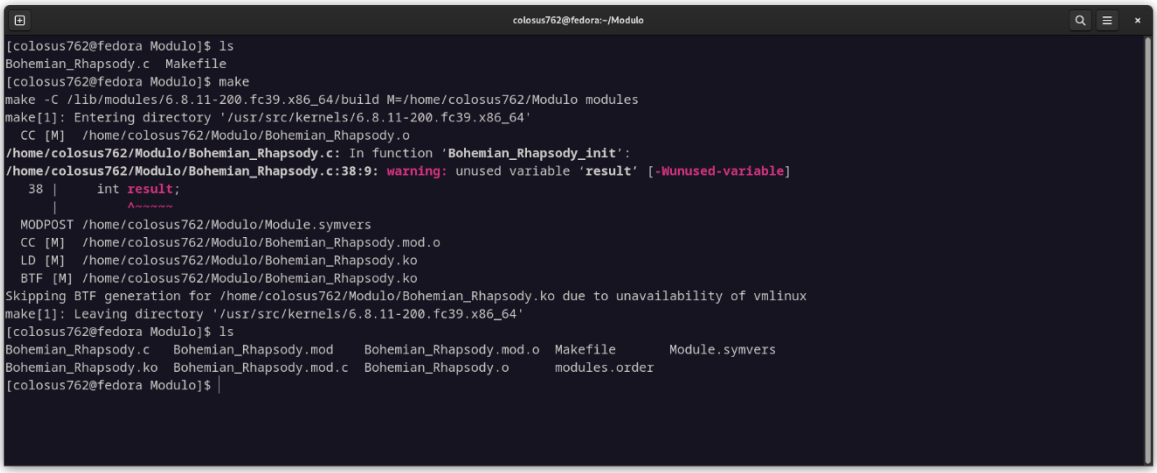
- obj-m += Bohemian_Rhapsody.o: Indica que el archivo objeto 'Bohemian_Rhapsody.o' debe ser compilado como un módulo del Kernel.
- make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules: Cambia al directorio que contiene los archivos de cabecera e invoca el objetivo modulo, lo que desencadena el proceso de compilación de los módulos en el directorio.
- make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean: Cambia al directorio que contiene los archivos de cabecera y scripts necesarios para compilar módulos e invoca el objetivo 'clean', lo que desencadena el proceso de limpieza de los archivos generados durante la compilación en el directorio.
- make -c /lib/modules/6.1.13-200.fc37.x86_64/build M=$(PWD) modules: Cambia al directorio que contiene los archivos de cabecera y scripts necesarios para compilar módulos e invoca el objetivo 'modules' para compilar los módulos del Kernel ubicados en el directorio actual ('/lib/modules/6.1.13-200.fc37.x86_64/build').

Compilación del modulo

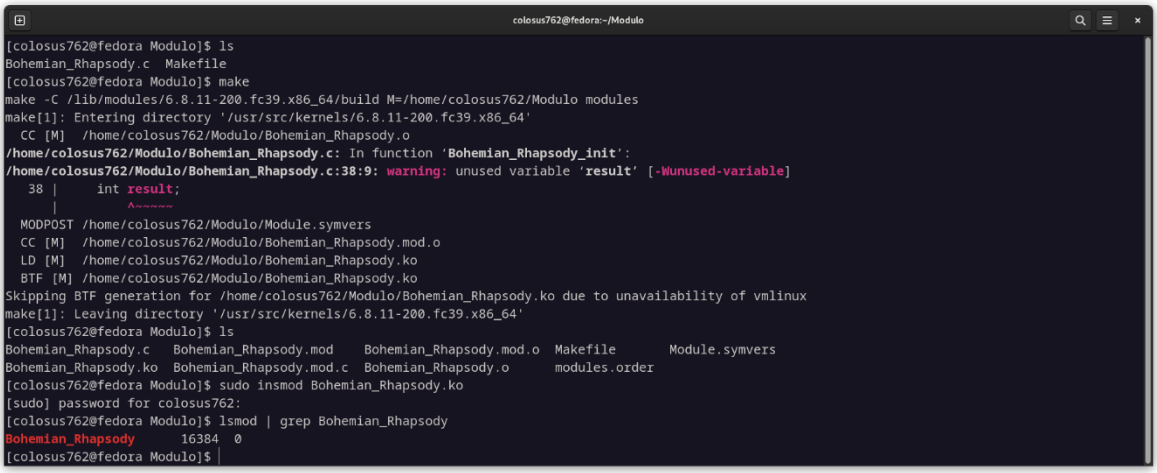
Nos posicionamos en el directorio que contiene nuestro archivo Makefile y el código fuente de nuestro modulo.

Ejecutamos el comando make lo que nos compilara nuestro modulo en base a las instrucciones del archivo Makefile.

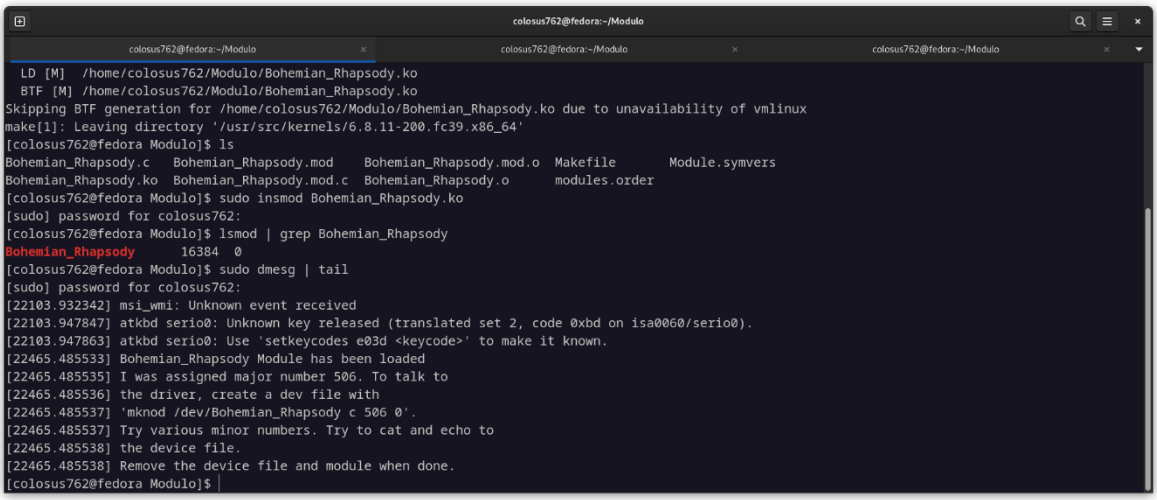
Con esto se crea una serie de archivo entre ellos Bohemina_Rhapsody.ko que será el archivo que podrá leer nuestro Kernel de Linux.



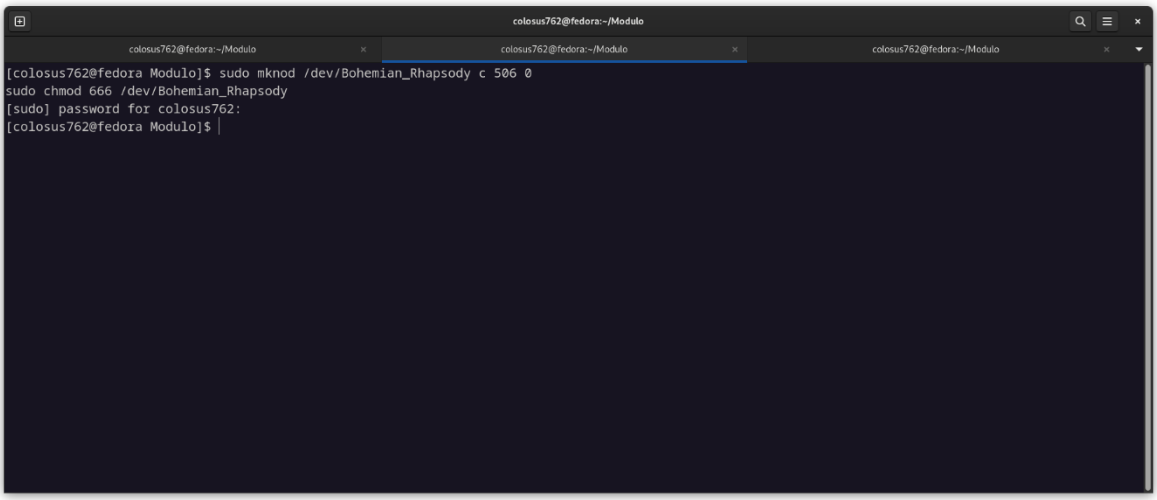
Con el comando sudo insmod Bohemian_Rhapsody.ko estamos indicando que deseamos cargar nuestro modulo a espacio de Kernel. Una vez hecho esto con el comando lsmod | grep Bohemian_Rhapsody podemos verificar si nuestro modulo se encuentra cargado en espacio de Kernel.



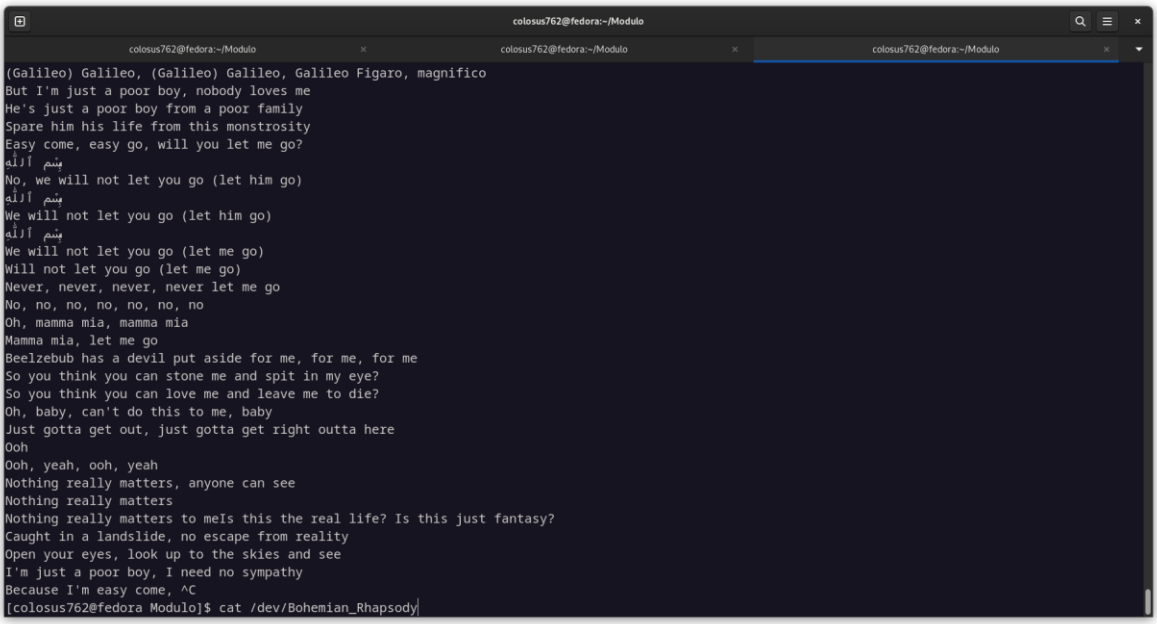
Con el comando sudo dmesg | tail muestra los últimos mensajes recibidos desde el Kernel, de aquí se extrae el major number (506 en este caso) que usa el espacio de Kernel y el espacio de usuario para comunicarse.



Creamos un archivo de dispositivo en /dev, que permiten la comunicación entre el software del usuario y el hardware o los controladores del Kernel. El comando chmod cambia los permisos de acceso de archivos y directorios.



Y finalmente podemos hacer que nuestro modulo cante con cat /dev/Bohemian_Rhapsody



Y finalmente retiramos nuestro modulo de los mdulos cargados con el comando rmmod Bohemian_Rhapsody