Gestión de E/S

Sistemas Operativos
Módulo 4

Índice

Bibliografía:
Sistemas Operativos, J. Carretero [et al.]
Modern Operating Systems, Andrew S. Tanenbaum
Linux Device Drivers, Cap. 3 "Char Drivers".

Índice

Introducción

- El corazón de una computadora lo constituye la UCP, pero no serviría de nada sin:
  - Dispositivos de almacenamiento no volátil:
    - Secundario: discos
    - Terciario: cintas
  - Dispositivos periféricos que le permitan interactuar con el usuario (los teclados, ratones, micrófonos, cámaras, etc.)
  - Dispositivos de comunicaciones: permiten conectar a la computadora con otras a través de una red

Dispositivos de salida: Impresora, monitor, ...

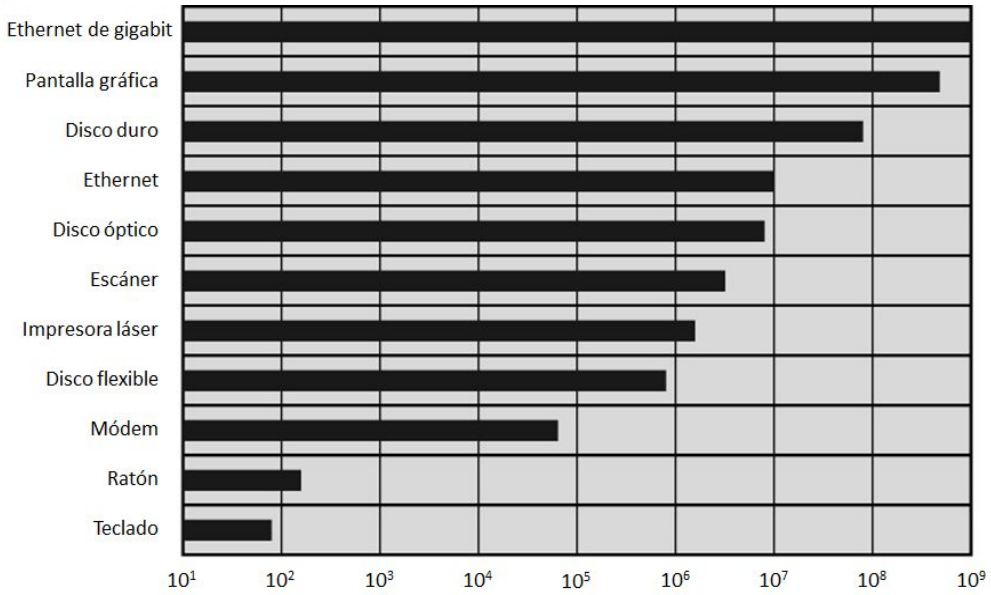
Unidad principal (UCP, registros, memoria RAM, Entrada/Salida (discos internos, red,...))

Dispositivos de entrada (teclado, ratón)

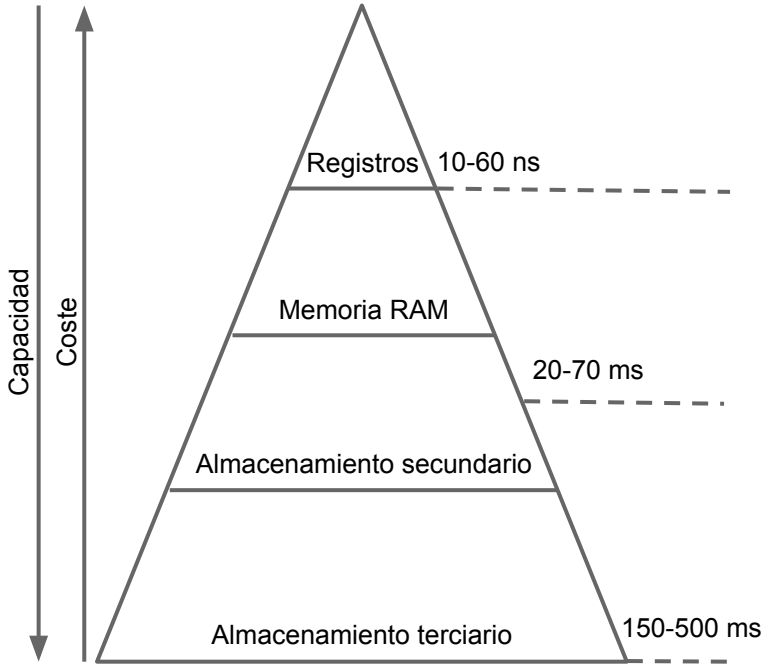
Dispositivos de E/S (discos, cintas, modem, ...)

Velocidad de los dispositivos

- El gran problema de todos estos dispositivos de E/S es que son muy lentos:
  - La UCP procesa instrucciones a >1 GHz (<1 ns/ciclo) y la memoria RAM tiene un tiempo de acceso de nanosegundos (10^{-9})
  - Los dispositivos de E/S más rápidos tienen una velocidad de acceso del orden de milisegundos (10^{-3})
  - **...**



Velocidades de datos [bps] de dispositivos de E/S típicos.



Visión del sistema de E/S

- La visión del sistema de E/S puede ser muy distinta dependiendo del nivel de detalle necesario en su estudio
  - Para los programadores: el sistema de E/S es una caja negra que lee y escribe datos en dispositivos externos a través de una funcionalidad bien definida
  - Para los fabricantes de dispositivos: un dispositivo es un instrumento muy complejo que incluye cientos o miles de componentes electrónicos o electro-mecánicos.
- Los diseñadores de sistemas operativos y drivers se encuentran en un lugar intermedio entre los dos anteriores:
  - Les interesa la funcionalidad del dispositivo, aunque a un nivel de detalle mucho mayor que el requiere el programador de apps.
    - Necesitan información sobre su comportamiento interno para poder optimizar los métodos de acceso a los mismos
  - Requieren conocer la arquitectura del software de E/S del SO para poder exponer cada dispositivo al usuario

Funciones del sistema de E/S

- Facilitar el manejo de los dispositivos periféricos. Para ello debe ofrecer una interfaz entre los dispositivos y el resto del sistema que sea sencilla y fácil de utilizar
- Optimizar la E/S del sistema, proporcionando mecanismos de incremento de prestaciones donde sea necesario
- Proporcionar dispositivos virtuales que permitan conectar cualquier tipo de dispositivo físico sin que sea necesario remodelar el sistema de E/S del sistema operativo
- Permitir la conexión de dispositivos nuevos de E/S, solventando de forma automática su instalación usando mecanismos del tipo plug&play

Índice

Drivers (1/3)

- Driver: componente software del SO destinado a gestionar un tipo específico de dispositivo de E/S
  - También llamado controlador SW o manejador de dispositivo
- Cada driver se divide en dos partes:
  - Código independiente del dispositivo para dotar al nivel superior del SO de una interfaz
    - Interfaz similar para acceso a dispositivos muy diferentes
    - Simplifica la labor de portar SSOOs y aplicaciones a nuevas plataformas hardware
  - Código dependiente del dispositivo necesario para interactuar con dispositivo de E/S a bajo nivel
    - Interacción con controlador HW
    - Manejo de interrupciones
    - …

Drivers (2/3)

¿Qué clases de funciones debe implementar un driver genérico? ¿Qué tipos de operaciones debe ofrecer a los programas de usuario?

- Acciones comunes dispositivos E/S
  - Un dispositivo puede generar y/o recibir datos (lectura/escritura)
    - Operaciones de L/E en el driver
  - Algunos dispositivos pueden necesitar un control específico
    - Ejemplo: *rebobinar una cinta*
    - Operación de control en el driver
  - Muchos dispositivos generan interrupciones
    - Driver debe realizar procesamiento ligado a una interrupción

Drivers (3/3)

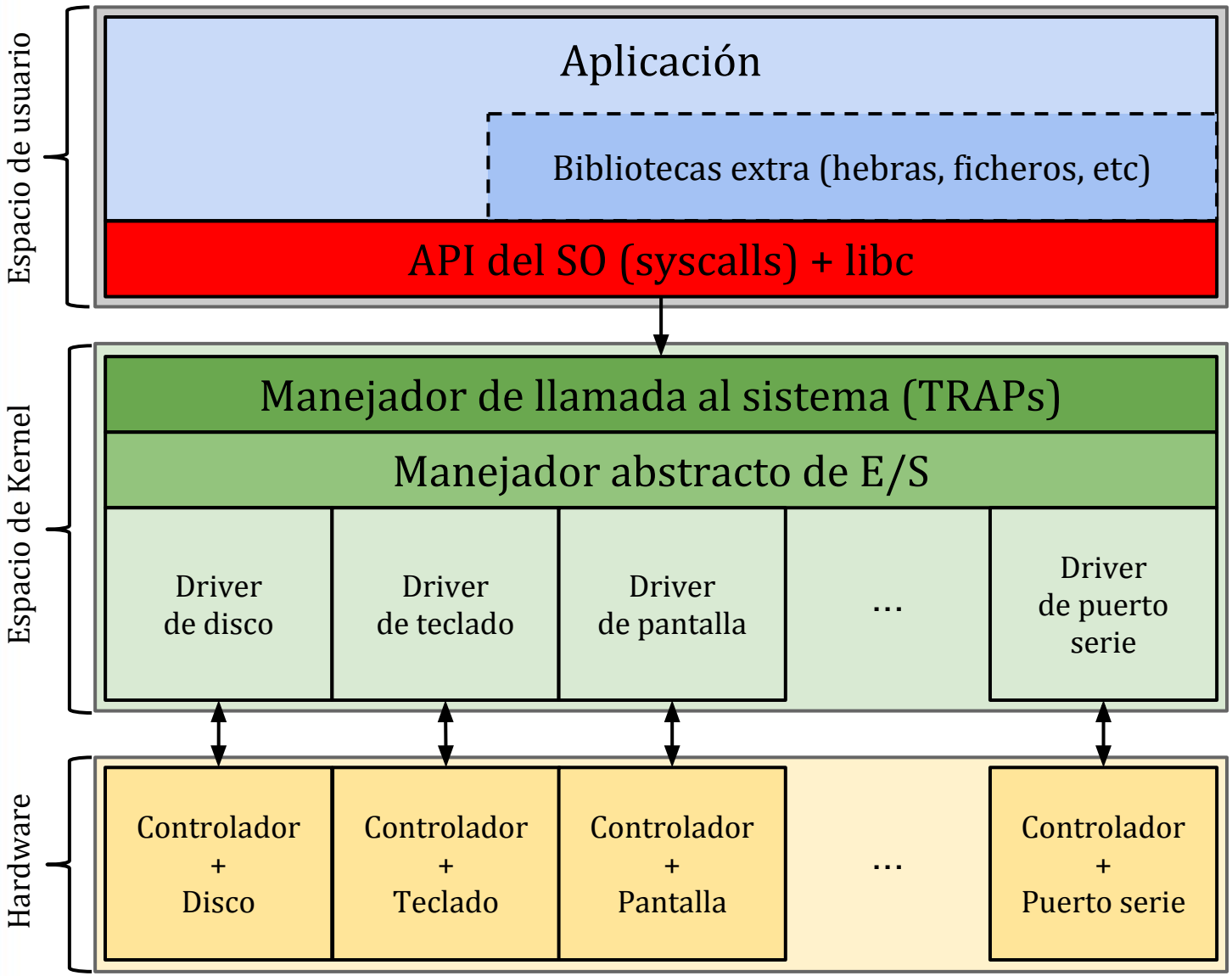
- Sin embargo, no podemos ser muy, muy genéricos
  - Por ejemplo., no podemos acceder a nivel de byte a un disco (acceso a nivel de bloque)
- Al final, los dispositivos se tienen que dividir en un pequeño número de clases:
  - Dispositivos de carácter
    - puerto serie, teclado, ratón, …
  - Dispositivos de bloque
    - disco, red, pantalla, …
- Estas clases o categorías se deben incluir para definir diferentes protocolos en la interfaz abstracta o genérica del driver y así ganar en rendimiento de la E/S

Índice

Ficheros de dispositivo

- Casi todos los dispositivos de E/S se representan como ficheros especiales (que pueden ser de bloque o carácter)
  - /dev/sda1 para la primera partición del primer disco SATA o USB
  - /dev/tty0 para el primer terminal/consola de texto
  - /dev/lp0 para la impresora
- El acceso a estos ficheros especiales se realiza mediante las llamadas al sistema open(), read(), write() y close()
  - Un programa de usuario puede acceder al dispositivo de E/S, siempre y cuando usuario tenga permisos de acceso al fichero especial
  - Excepcionalmente puede requerirse ioctl() para realizar operaciones de control

Ficheros de dispositivo



Cuando un programa de usuario invoca una llamada al sistema sobre un fichero especial (ej. read()), se ejecuta una función del driver que hace el trabajo correspondiente.

Por cada fichero especial hay asociado un driver.

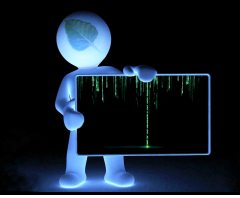
Ficheros de dispositivo

- Los ficheros de dispositivo se alojan *por convenio* en el directorio `/dev`.
- Para ver los ficheros de dispositivo presentes en el sistema basta con listar el contenido del directorio /dev

Terminal

Major and minor
(ver más adelante)

```
$ ls -l /dev
...
brw-r----- 1 root disk 3, 0 Nov 19 10:20 hda
brw-r----- 1 root disk 3, 1 Nov 19 10:20 hda1
brw-r----- 1 root disk 3, 2 Nov 19 10:20 hda2
...
crw-rw---- 1 root uucp 4, 64 Nov 19 10:20 ttyS0
crw-rw---- 1 root uucp 4, 65 Nov 19 10:20 ttyS1
...
crw-rw---- 1 root audio 14, 3 Dec 2 00:31 dsp
crw-rw---- 1 root audio 14, 4 Dec 2 00:31 audio
...
crw-rw-rw- 1 root root 1, 8 Nov 19 10:20 random
```



Ficheros de dispositivo

- Los ficheros de dispositivo son un potente mecanismo de trabajo con varios dispositivos hardware tal y como si fuesen ficheros ordinarios.

```
# dd if=/dev/hda of=mbr.bin bs=512 count=1
```

  - **Descripción**: El comando leerá los primeros 512 bytes desde el comienzo del disco duro (el Master Boot Record, MBR) y lo almacenará en el archivo mbr.bin (dd es un comando que copia parte de un fichero en otro).

```
# dd if=/dev/zero of=/dev/hda
```

  - **Descripción**: Escribe ceros en todo el disco duro, eliminando toda la información existente

- ¿Cómo sabe el sistema operativo a qué dispositivo está asociado un fichero de dispositivo? ⇒ (major, minor)

Ficheros de dispositivo

- Se puede crear un nuevo fichero de dispositivo usando el comando mknod

```
# mknod /dev/<nombre> <tipo> <núm_major> <núm_minor>
```

- donde:
  - `<nombre>`: Nombre de archivo de dispositivo
  - `<tipo>`: **c** para dispositivos tipo carácter y **b** para dispositivos tipo bloque
  - `<núm_major>` y `<núm_minor>`: major y minor del driver del dispositivo al que este fichero queda asociado (se explican a continuación)
- Se pueden crear ficheros de dispositivo con cualquier major y minor, sólamente útil si existe un driver asociado con los mismos números.

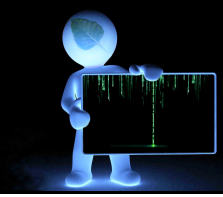
Major and minor number

- Los dispositivos se agrupan en clases. Cada clase tiene un número de dispositivo principal (major) que la identifica
  - https://www.kernel.org/doc/Documentation/devices.txt
- Cada fichero de dispositivo tiene asociado un par (major, minor) que lo identifica de forma biunívoca
  - major: ID de la clase de dispositivos a la que pertenece
  - minor: ID local para que el driver pueda distinguir al dispositivo en caso de gestionar varios
- Ejemplo: Driver que gestiona 2 discos duros
  - Discos representados mediante ficheros de dispositivo
    - /dev/sda, /dev/sdb
  - Ambos ficheros especiales tendrán el mismo major number pero distinto minor number (misma clase, distinta instancia, por analogía con POO)

Major and minor number

- El comando `stat` permite consultar el tipo de dispositivo asociado al fichero así como el major y minor number del mismo

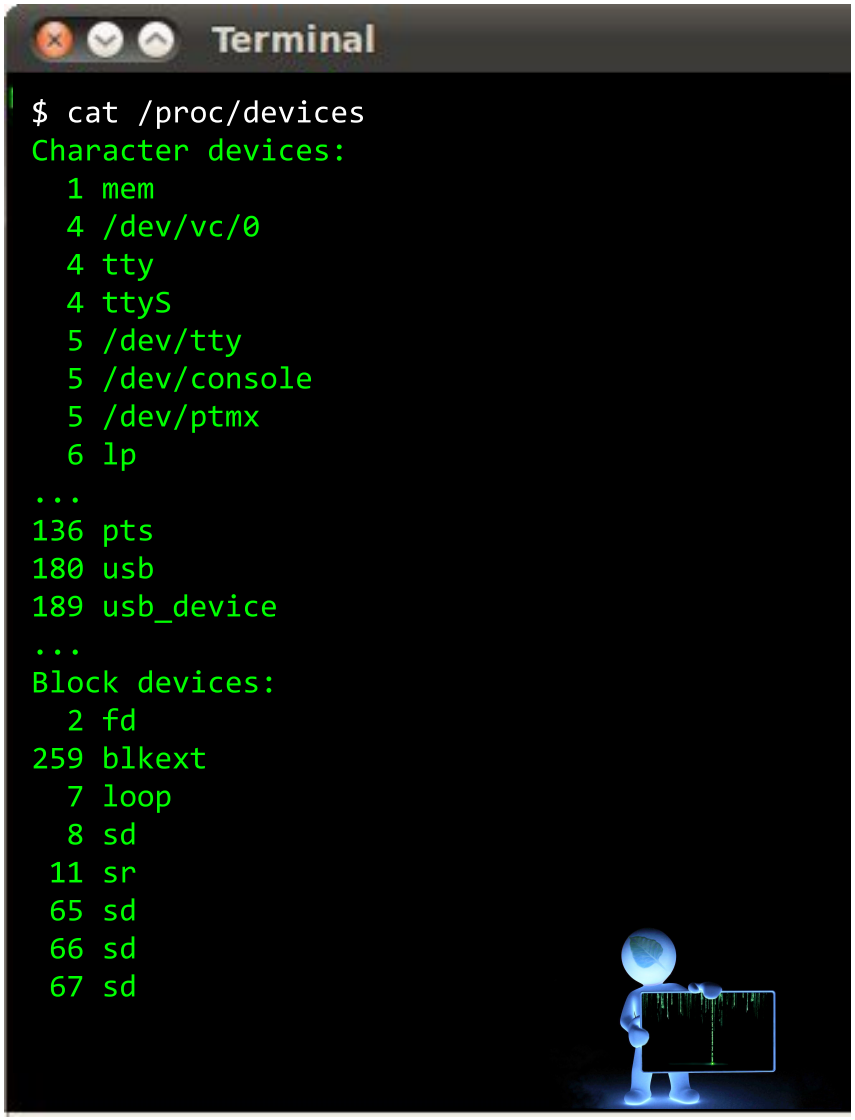
```
Terminal
$ stat /dev/tty1
  File: «/dev/tty1»
  Size: 0  Blocks: 0  IO Block: 4096  fichero especial de caracteres
Device: 5h/5d  Inode: 1259  Links: 1  Device type: 4,1
Access: (0600/crw-------) Uid: (0/root)  Gid: (0/root)
Access: 2014-02-24 08:18:59.663968939 +0100
Modify: 2014-02-24 08:18:59.523954207 +0100
Change: 2014-02-24 08:18:59.523954207 +0100
$ stat /dev/sda1
  File: «/dev/sda1»
  Size: 0  Blocks: 0  IO Block: 4096  fichero especial de bloques
Device: 5h/5d  Inode: 1708  Links: 1  Device type: 8,1
Access: (0660/brw-rw----) Uid: (0/root)  Gid: (6/disk)
Access: 2014-02-24 08:18:59.663968939 +0100
Modify: 2014-02-24 08:18:59.523954207 +0100
Change: 2014-02-24 08:18:59.523954207 +0100
```



Representación de (major, minor)

- En el kernel Linux el par (major,minor) está representado mediante el tipo dev_t
  - dev_t: número de 32 bits (12 bits major, 20 bits minor)
- Por motivos históricos el empaquetamiento es complejo, se utilizan macros:
  - Acceso a números: MAJOR(dev_t dev), MINOR(dev_t dev)
  - Construcción de par: MKDEV(int major, int minor)

Asociación entre driver y major



- La asociación entre el driver del dispositivo y el número de versión mayor asignado puede consultarse en /proc/devices
- La mayor parte de los drivers de dispositivo se implementan como módulos cargables del kernel

Módulos

- Un driver puede ser estáticamente enlazado “dentro” del kernel o compilado como módulos del kernel.
  - Entonces, un módulo es una parte del kernel que puede cargarse y descargarse del kernel bajo demanda.
  - Esto es más conveniente que el enlazado estático, porque de esta manera se puede añadir nueva funcionalidad al kernel cuando se necesite
- Gestión de módulos en Linux
  - `lsmod`: lista los módulos cargados actualmente.
  - `modinfo`: nos da información sobre un módulo.
  - `insmod`: carga un módulo. Interfaz de bajo nivel.
  - `rmmod`: descarga un módulo.
  - `modprobe`: interfaz de alto nivel para cargar módulos.
    - Busca en /etc/modprobe información y ruta de los módulos

Módulos vs. Aplicaciones

Aplicaciones	Módulos
Modo usuario	Modo kernel
Cualquier función de biblioteca disponible	Sólo símbolos exportados por el kernel • /proc/kallsyms • libc no disponible • printk()
Realizan su función de principio a fin (main)	Ejecutan su función de inicio "`init_module`" al registrarse y quedan residentes para dar servicio

En los módulos no podemos usar printf (libc). El kernel proporciona una función similar, printk, que permite escribir mensajes en los ficheros de log y por consola virtual (no terminal).

Ejemplo - Implementación (hello.c)

File Edit Options Buffers Tools Help

```c
/*
* hello.c - El módulo más simple y menos original posible.
*/
#include <linux/module.h> /* Requerido por todos los módulos */
#include <linux/kernel.h> /* Requerido por KERN_INFO */
MODULE_LICENSE("GPL");
int init_module(void) {
  printk(KERN_INFO "Hello world.\n");
  /*
  * Retorno != 0 implica fallo de init_module; el módulo no
  * puede cargarse.
  */
  return 0;
}
void cleanup_module(void) {
  printk(KERN_INFO "Goodbye world.\n");
}
```

-U:--- **ejemplo** All L1 (Fundamental)----------

Ver símbolos (variables y funciones) exportados por el kernel:
`$ cat /proc/kallsyms`
Ver salida de printk:
`# cat /var/log/messages`

Ejemplo - Compilación (Makefile)

- Los módulos del kernel deben compilarse de forma diferente a los ficheros C habituales.

File Edit Options Buffers Tools Help

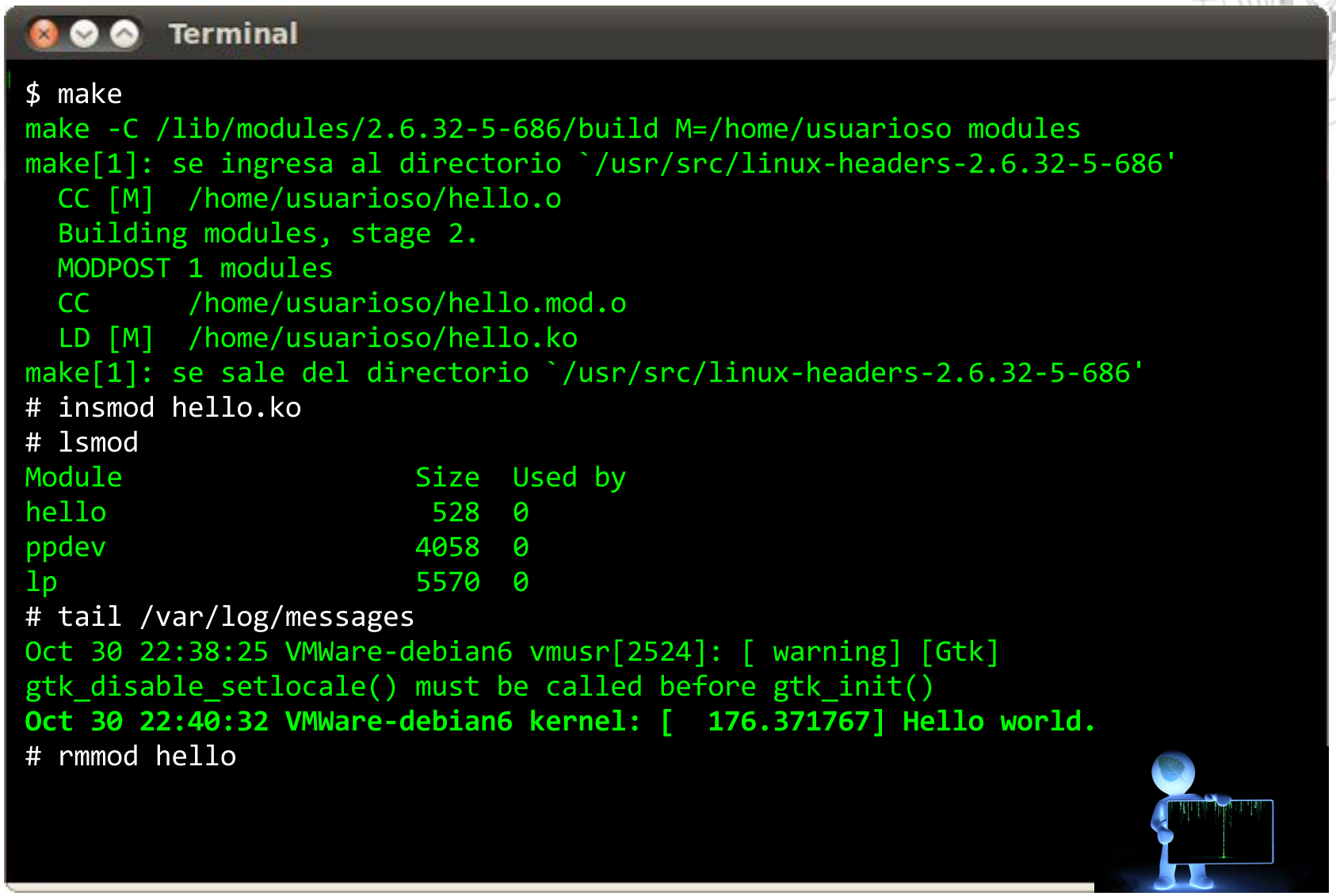
```
MODULE_NAME=hello

obj-m := ${MODULE_NAME}.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
```

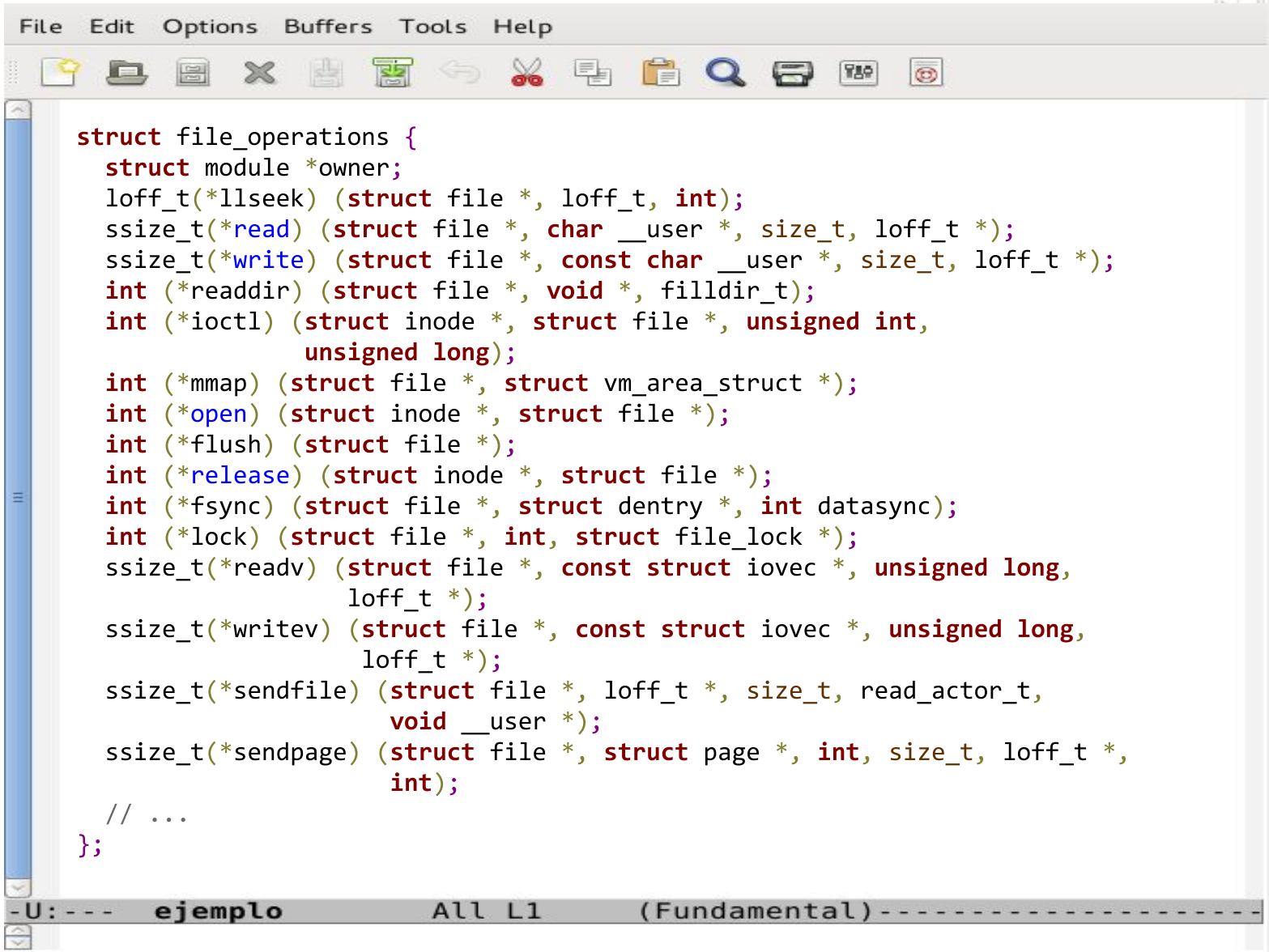
Ejemplo - Test



Implementar un driver como módulo

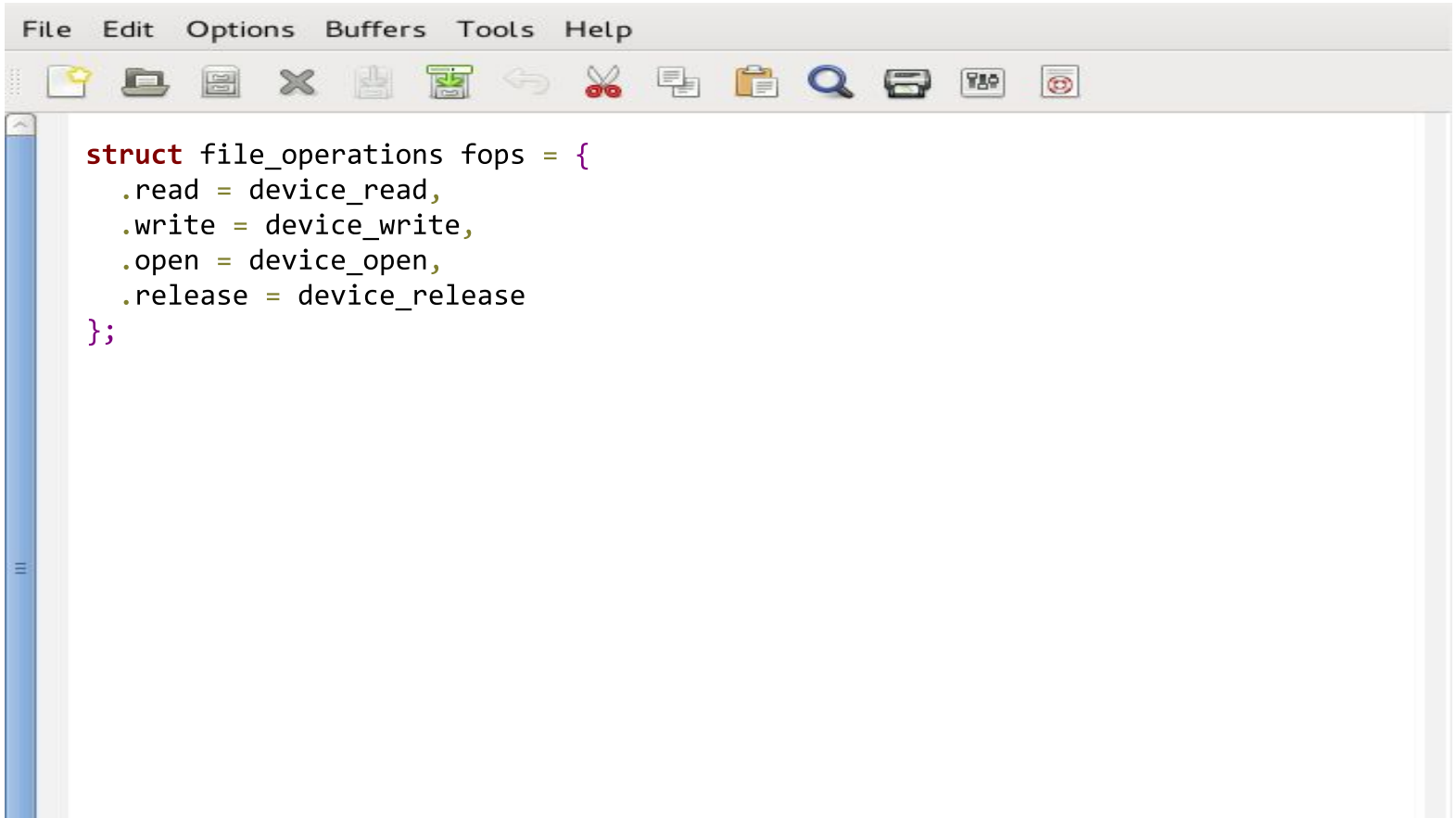
1. Crear un módulo del kernel con funciones `init_module()` y `cleanup_module()`
2. Implementar las operaciones de la interfaz del dispositivo de caracteres: `struct file_operations`
3. En la función de inicialización:
   - Reservar major number y rango de minor numbers para el driver
     - `alloc_chrdev_region()`
   - Crear una estructura cdev_t y asociarle las operaciones y el rango de major/minor
     - Usar `cdev_alloc()`, `cdev_init()` y `cdev_add()`
4. En la función de descarga:
   - Destruir estructura `cdev_t`: `cdev_del()`
   - Liberar el rango (major, minor): `unregister_chrdev_region()`

La estructura file_operations



La estructura file_operations

- No todos los campos de esta estructura deben inicializarse, sólamente aquellas que se corresponden con operaciones soportadas por el driver del dispositivo, por ejemplo:



alloc_chrdev_region

```
#include <linux/fs.h>
int alloc_chrdev_region (dev_t *first, unsigned int firstminor,
                         unsigned int count, char *name)
```

- Parámetros
  - `first`: Parámetro de retorno. Primer par (major,minor) que el kernel reserva para el driver.
  - `firstminor`: Menor minor number a reservar dentro del rango consecutivo que otorga el kernel
  - `count`: Número de minor numbers a reservar para el driver
  - `name`: Nombre del driver (cadena de caracteres arbitraria.) Es el valor que aparecerá en /proc/devices al cargar el driver
- Valor de retorno
  - 0 en caso de éxito.
  - En caso de fallo, devuelve valor negativo que codifica el error.

register_chrdev_region

```
#include <linux/fs.h>
int register_chrdev_region (dev_t first,
                            unsigned int count, char *name);
```

- Parámetros
  - `first`: Primer par (major,minor) que el driver desea reservar.
  - `count`: Número de minor numbers a reservar para el driver.
  - `name`: Nombre del driver (cadena de caracteres arbitraria.) Es el valor que aparecerá en `/proc/devices` al cargar el driver
- Valor de retorno
  - 0 en caso de éxito.
  - En caso de fallo, devuelve valor negativo que codifica el error.

unregister_chrdev_region

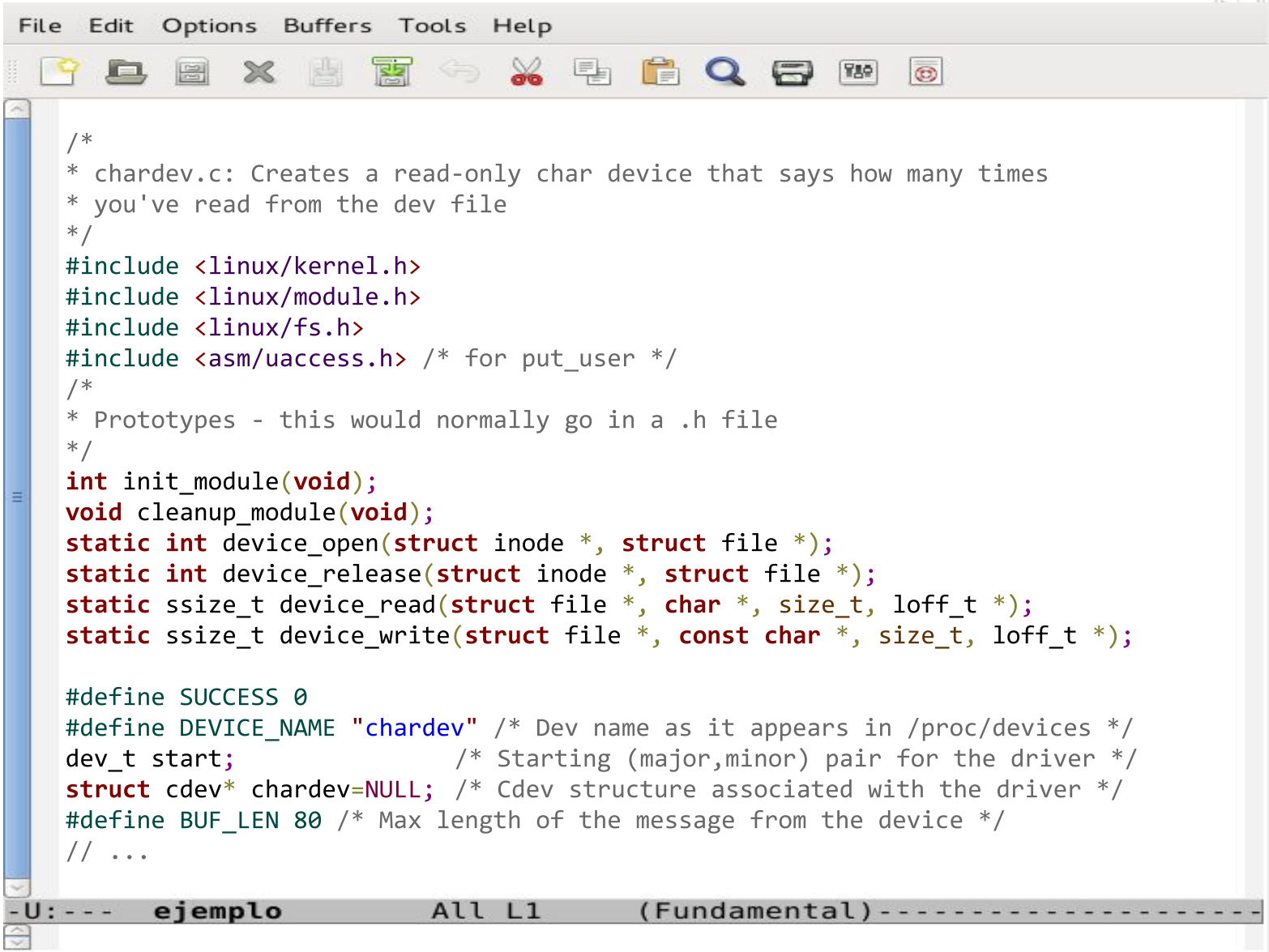
```
#include <linux/fs.h>
int unregister_chrdev_region (dev_t first, unsigned int count)
```

- Parámetros
  - `first`: Primer par (major,minor) que el driver había reservado previamente
  - `count`: Número de minor numbers consecutivos que el driver había reservado
- Valor de retorno
  - 0 en caso de éxito.
  - En caso de fallo, devuelve valor negativo que codifica el error.

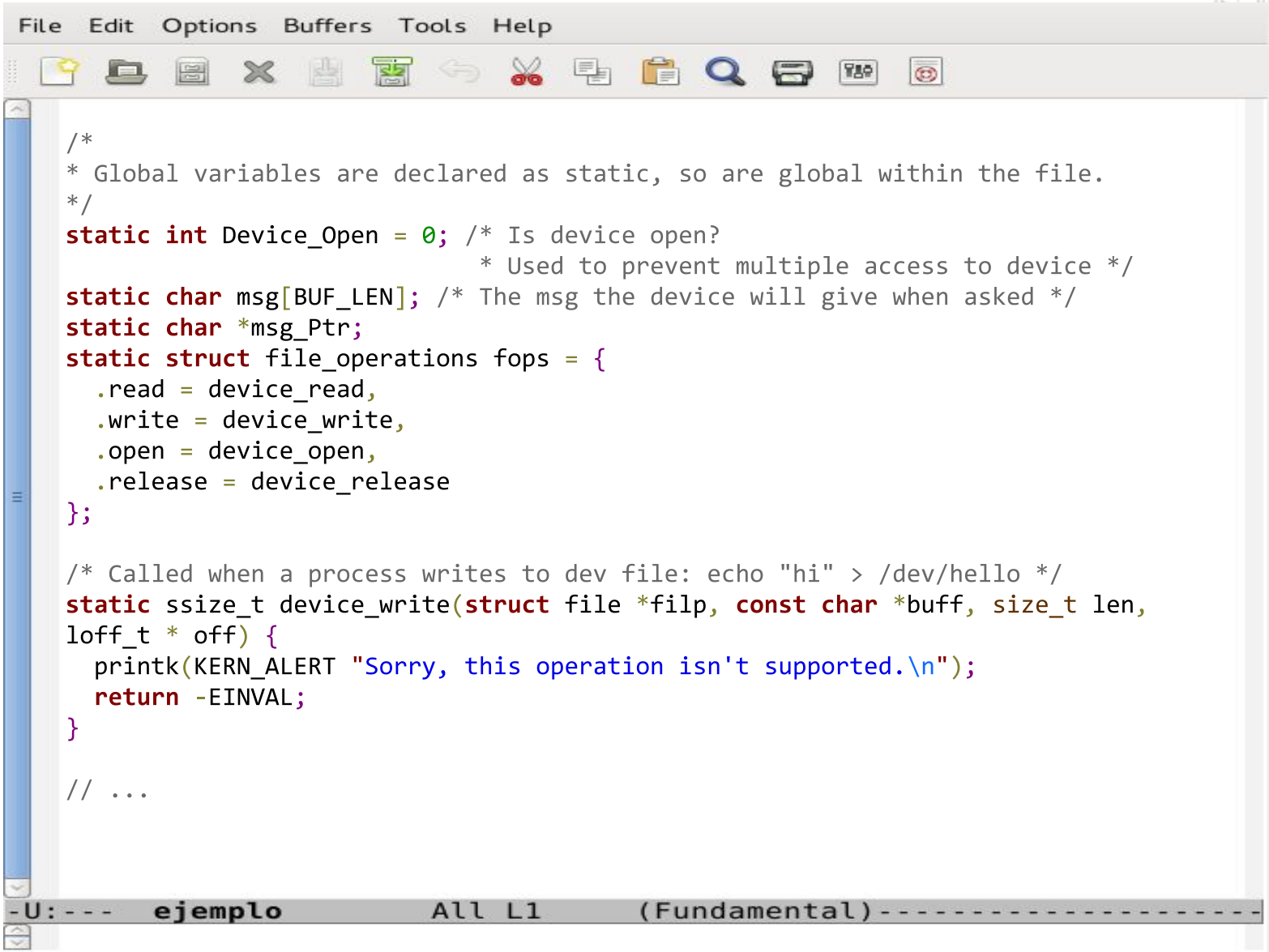
Estructura cdev

- Para que el driver reciba peticiones de los programas de usuario, debe crear una estructura `cdev`
  - **struct** cdev *cdev_alloc (**void**);
    - Crea estructura cdev y retorna un puntero no nulo a la misma en caso de éxito
  - **void** cdev_init (**struct** cdev *p, **struct** file_operations *fops);
    - Asocia interfaz de operaciones del driver a estructura cdev
  - **int** cdev_add (**struct** cdev *p, dev_t first, **unsigned** count);
    - Permite que peticiones de programas de usuario sobre el rango de (major,minor) especificado mediante parámetros first y count sean redirigidas al driver que gestiona estructura cdev
  - **struct** cdev *cdev_del (**void**);
    - Elimina asociaciones de estructura cdev (rangos de major/minor) y libera memoria asociada a la estructura

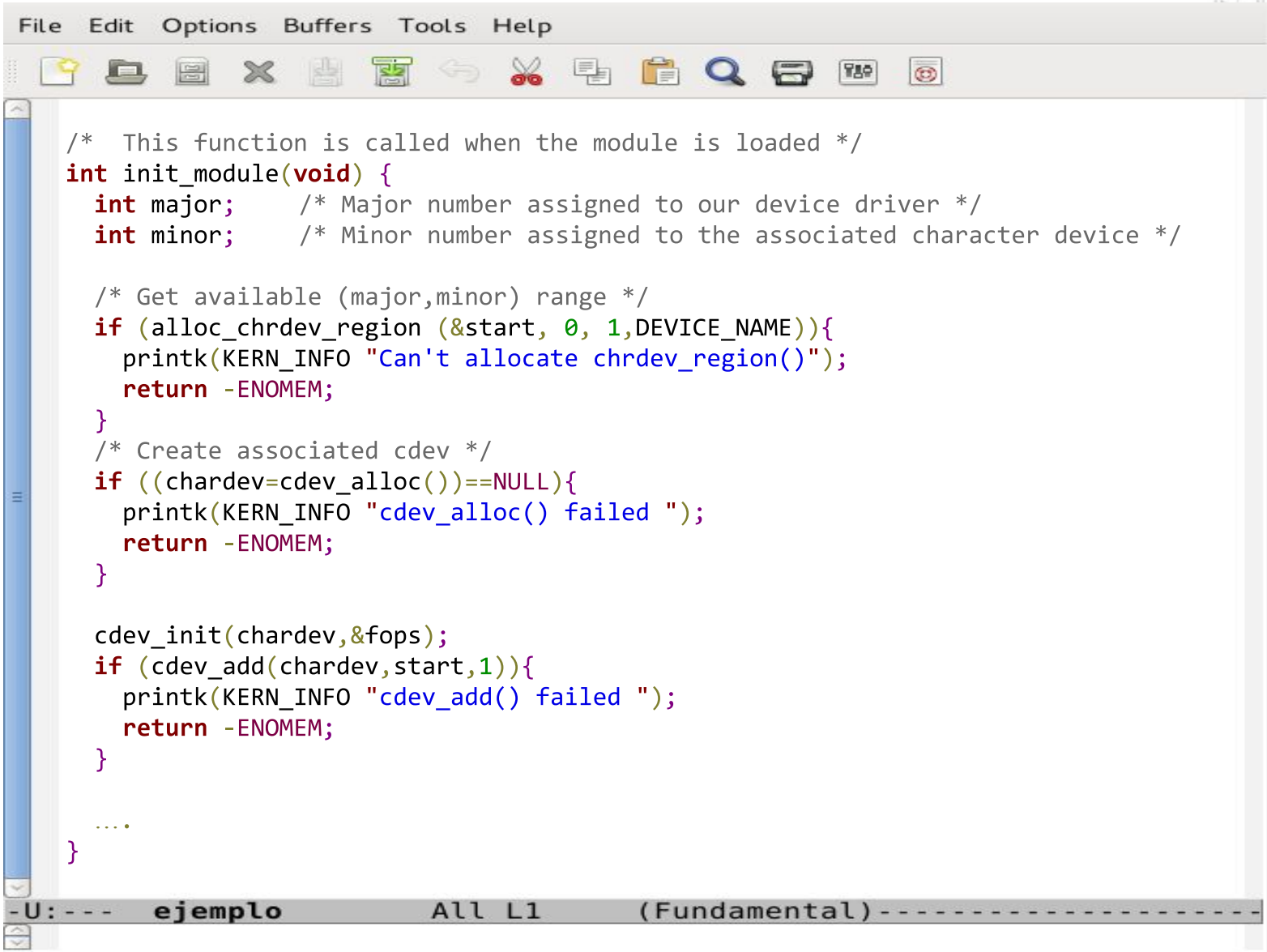
chardev.c: Dispositivo tipo carácter



chardev.c: Dispositivo tipo carácter



chardev.c: Dispositivo tipo carácter



chardev.c: Dispositivo tipo carácter

File Edit Options Buffers Tools Help

```
/*  This function is called when the module is loaded */
int init_module(void) {
      ....
  major=MAJOR(start);
  minor=MINOR(start);
  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 "'sudo mknod -m 666 /dev/%s c %d %d'.\n", DEVICE_NAME,
major,minor);
  printk(KERN_INFO "Try to cat and echo to the device file.\n");
  printk(KERN_INFO "Remove the device file and module when done.\n");
  return SUCCESS;
}

/* This function is called when the module is unloaded */
void cleanup_module(void) {
  if (chardev) /* Destroy chardev */
    cdev_del(chardev);
  /* Unregister the device */
  unregister_chrdev_region(start, 1);
}
```

-U:--- **ejemplo** All L1 (Fundamental)----------

chardev.c: Dispositivo tipo carácter

File Edit Options Buffers Tools Help

```
/* Called when a process tries to open the device file, like
* "cat /dev/mycharfile" */
static int device_open(struct inode *inode, struct file *file) {
  static int counter = 0;
  if (Device_Open)
    return -EBUSY;
  Device_Open++;
  sprintf(msg, "I already told you %d times Hello world!\n", counter++);
  msg_Ptr = msg;
  try_module_get(THIS_MODULE);
  return SUCCESS;
}

/* Called when a process closes the device file. */
static int device_release(struct inode *inode, struct file *file) {
  Device_Open--;
  /* We're now ready for our next caller */
  /* Decrement the usage count, or else once you opened the file, you'll
  * never get get rid of the module. */
  module_put(THIS_MODULE);
  return 0;
}
// ...
```

-U:--- ejemplo Al

Previenen el borrado de un módulo cuando está en uso. Incrementa/Decrementa un contador interno del kernel para cada módulo. Si el contador es distinto de cero, el módulo no puede eliminarse. Se pueden ver los valores de estos contadores con lsmod (Columna “Used by”).

chardev.c: Dispositivo tipo carácter

File Edit Options Buffers Tools Help

```
/* Called when a process, which already opened the dev file, attempts to
* read from it. */
static ssize_t device_read(struct file *filp, /* see include/linux/fs.h */
                           char *buffer, /* buffer to fill with data */
                           size_t length, /* length of the buffer */
                           loff_t * offset) {
  /* Number of bytes actually written to the buffer */
  int bytes_to_read = length;
  /* If we're at the end of the message, return 0 signifying end of file */
  if (*msg_Ptr == 0)
    return 0;
  if (bytes_to_read > strlen(msg_Ptr))
    bytes_to_read=strlen(msg_Ptr);
  copy_to_user(buffer,msg_Ptr,bytes_to_read);
  msg_Ptr += bytes_to_read;
  /* Most read functions return the number of bytes put into the buffer */
  return bytes_read;
}
// ...
```

-U:--- **ejemplo** All L1 (Fundamental)----------

copy_from_user, copy_to_user

- Las operaciones `read` y `write` de un fichero de dispositivo tienen como parámetro un puntero buffer del espacio de usuario
- No debemos confiar en los punteros del espacio de usuario (puede pertenecer a una región de memoria a la que el proceso asociado al driver no tenga acceso)
- Siempre se ha de trabajar con una copia privada de los datos en el espacio del kernel, usando:
  - **unsigned long** copy_from_user (**void*** to, **const void** __us
    unsigned long n);
  - **unsigned long** copy_to_user (**void** __user* to, **const void***
    unsigned long n);
  - Ambas funciones devuelven el número de bytes que NO pudieron copiarse

chardev.c: Makefile

File Edit Options Buffers Tools Help

```
MODULE_NAME=chardev

obj-m := ${MODULE_NAME}.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
```

-U:--- **ejemplo** All L1 (Fundamental)----------

chardev.c: Test

Terminal

```
$ make
  ...
# insmod chardev.ko
# tail /var/log/messages
Buscar en el archivo de log el major asignado a este módulo (251)
# mknod -m 666 /dev/chardev c 251 0
# cat /dev/chardev
I already told you 0 times Hello world!
# cat /dev/chardev
I already told you 1 times Hello world!
# cat /dev/chardev
I already told you 2 times Hello world!
Si intentamos escribir en el dispositivo obtendremos un mensaje de error, en
el terminal o en el archivo “log”:
# echo "Hello" > /dev/chardev
bash: echo: write error: Invalid argument
# rmmod /dev/chardev
¿Por qué aparece este error?
No obstante, enhorabuena, ¡acabamos de crear nuestro primer driver!
```