Capítulo 6

Xestión de dispositivos

Xa dixemos anteriormente que en Linux *todo é un arquivo*, e polo tanto os dispositivos tamén serán manexados como arquivos. Por iso empezaremos este tema falando dos *device files*, que serán uns arquivos especiais, pero arquivos ao fin e ao cabo.

Unha vez que entendamos ese concepto poderemos baixar no nivel de abstracción e falar dos *device drivers* que serán no fondo os programas cos que o *kernel* se comunica cos dispositivos.

Finalmente falaremos do sistema *udev*, empregado polos *kernels* posteriores ao 2.6 para xestionar os dispositivos dun xeito moito mais flexible.

6.1. Os *device files*

Os *device files* son arquivos especiais, localizados na carpeta */dev* que conteñen a información necesaria para acceder aos diferentes dispositivos, que poden ser reais (o disco duro, o teclado, etc) ou ben *virtuais* (un xerador de números aleatorios, un *burato negro*, etc).

Se executamos o camando *ls -l /dev* obteremos un longo listado con liñas como as seguintes:

```
brw-rw----  1 root disk        8,     0 feb 25 21:23 sda
brw-rw----  1 root disk        8,     1 feb 25 21:23 sda1
crw-rw-rw-  1 root tty         5,     0 mar  6 13:24 tty
crw--w----  1 root tty         4,     0 feb 25 21:23 tty0
crw-rw-rw-  1 root root        1,     9 feb 25 21:23 urandom
```

Imos explicar o significado da información que amosan esas liñas. Cada unha delas corresponde á un dispositivo diferente. Por exemplo a primeira liña corresponde ao primeiro disco duro. A primeira letra identifica o tipo de dispositivos. Os que levan unha *b* son dispositivos de bloque, nos que a información se transfire en bloques (de un 1Kb por exemplo). Son dispositivos deste tipo todos os de almacenamento (discos duros, memorias usb, etc) e tamén os *loop devices* que xa empregastes en algunha práctica. A segunda liña corresponde á primeira partición do disco, que tamén se trata como un dispositivo.

Os que levan unha *c* son dispositivos de caracteres, nos cales a información flue *byte* a *byte*. Exemplos de dispositivos deste tipo son o teclado, as terminais ou a memoria. No listado do exemplo vemos dúas terminais, *tty* e *tty0* e tamén *urandom* que é un xerador de números aleatorios.

Hai un terceiro tipo de dispositivo, os *sockets* [1], que teñen que ver con comunicación entre procesos, pero dos que non imos a falar aquí.

A continuación ven a máscara de permisos. Moi importante porque define o que os usuarios van a poder facer con eses dispositivos. Por exemplo os discos so teñen permiso de lectura/escritura para root, o seu propietario. Eso impide a un usuario normal acceder directamente ao disco duro (farao por el o *kernel* do sistema). Sen embargo vemos que *tty* si ten permiso para todos. Esto é asi porque correponde co terminal asociado ao proceso en curso, que debe poder escribir os seus mensaxes na terminal.

Os campos cecais mais importantes son os dous números que aparecen a continuación. O primeiro deles denomínase *major code* e indica o tipo de dispositivo. Na documentación de Linux existe un arquivo denominado *devices.txt*. [2] que recolle todos os *major code* permitidos e indicanos de que tipo de dispositivo se trata.

Imos a fixarnos na primeira liña do listado anterior. Vemos que é un dispositivo de bloques con *major code* 8. No arquivo *devices.txt* indícase o seguinte:

```
8 block SCSI disk devices (0-15)
    0 = /dev/sda First SCSI disk whole disk
    16 = /dev/sdb Second SCSI disk whole disk
    32 = /dev/sdc Third SCSI disk whole disk
```

[1] https://es.wikipedia.org/wiki/Archivo_de_dispositivo

[2] https://github.com/torvalds/linux/blob/master/Documentation/admin-guide/devices.txt

```
...
240 = /dev/sdp Sixteenth SCSI disk whole disk
```

Como vemos indícanos que se trata dun disco duro de tipo SCSI, pero tamén se aplica aos discos SATA, os empregados maioritariamente hoxe. E tamén nos indica que ese mesmo *major code* se empregaria para o segundo, terceiro .. ata o dezaseisavo disco.

Volvendo ao listado anterior, fixémonos agora no segundo número, un 0 nese caso. A este número denomínaselle *minor code*. Este número xa non define o tipo do dispositivo senón que permite ter varias *instancias* do mesmo tipo de dispositivo.

Segundo o arquivo *devices.txt* o *minor code* 0 indicaría que se trata do primeiro disco. Vedmos tamén que o 16 se reserva para o segundo disco. A razón neste caso é porque os números do 1 ao 15 se reservan para as posibles particións dese primeiro disco, que son tratados como dispositivos tamén. Volvendo ao listado da páxina anterior vemos que efectivamente o *major code* 8 e *minor code* 1 corresponde coa partición *sda1*.

No listado anterior vemos por exemplo que ao dispositivo *urandom* lle corresponde o *major code* 1, que indica que se trata de un dispositvo de memoria. Dentro dese conxunto de *pseudodispositvos* [3] a este en concreto correpondelle o *minor code* 9. Outros dispositvos de memoria serían por exemplo o */dev/zero*, un xerador de ceros, que ten o *minor code* 5 ou */dev/null* que ten *minor code* 3.

O conxunto de todos os dispositivos que é capaz de manexar o *kernel* actual atopase listado no arquivo */proc/devices*. No meu caso sería algo como isto:

```
Character devices:
  1 mem
  4 /dev/vc/0
  4 tty
 ...
Block devices:
  7 loop
  8 sd
  ...
```

[3] https://es.wikipedia.org/wiki/Archivo_de_dispositivo

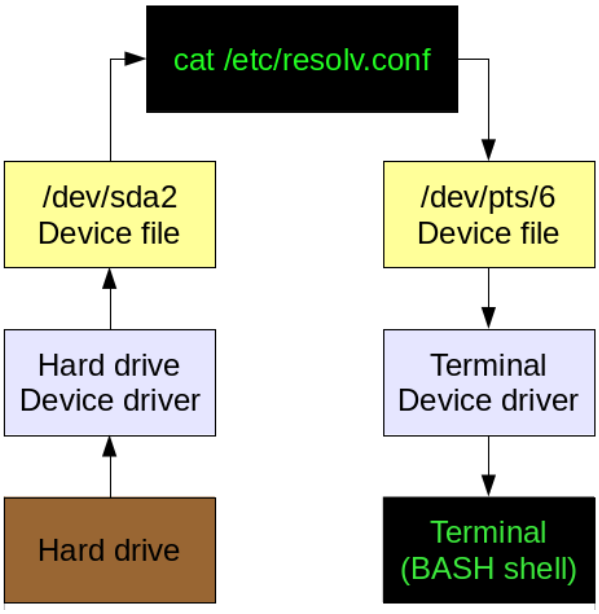

Figura 6.1: *Device file* e *device driver* (sacado de [4])

6.2. Os *device drivers*

Para entender mellor o concepto de *device file* e *device driver* imos apoioarnos na figura 6.1.

Supoñamos que executamos o comando *cat /etc/resolv.conf* na consola. O comando *cat* permite visualizar o contido do arquivo indicado no argumento. O que fai polo tanto é mandar unha petición ao *kernel* para que acceda a dito arquivo. O xestor de arquivos do *kernel* determina que ese arquivo en concreto está na partición aloxada en */dev/sda2*. Consultando a táboa de inodos do propio *file system* poderá determinar o lugar exacto do disco duro na que está dito arquivo.

A partir de aí, para ler o seu contido precisamos un programa específico capaz de interacionar co disco duro, co hardware, para colocar os cabezais do disco axeitadamente e ler o contido dos sectores precisos. Os programas capaces de realizar este tipo de accións son os denominados *device drivers*.

Os *device driver* ou ben forman parte do propio *kernel* ou mais habitualmente son módulos, aloxados polo tanto na carpeta */lib/modules*. Cada *major code* ten asociado un *device driver* diferente. No noso caso o *device file sda2* ten o *major code* 8, que ten asociado un *device driver* específico capaz de ler o contido de particións aloxadas en discos SCSI ou SATA. Empregando ese *device driver*, cos

[4] https://opensource.com/article/16/11/managing-devices-linux

parámetros correspondentes o *kernel* le o contido dese arquivo.

Para amosar o seu contido pola consola o intérprete de comandos segue o proceso inverso. En primeiro lugar determina cal é o *device file* ou dispositivo asociado á consola. Neste caso determina que é */dev/pts/6*. Segundo o arquivo *devices.txt* a ese dispositivo corresponderalle un *major code* 136 e un *minor code* 6. Suponse que ese programa saberá como transformas os códigos ASCII en información axeitada para que a tarxeta gráfica a convirta en pixels ilumandos na consola. Esa será a información que envie ao terminal e o que veremos finalmente na nosa pantalla.

En definitiva, os *device drivers* permiten manexar os dispositivos como arquivos ordinarios, dos cales podemos recibir ou enviar datos, como calquera outro arquivo, sin preocuparnos polos detalles específicos do dispositivo [5].

6.3. Xestión dinámica de dispositivos

Como dixemos na sección anterior, a carpeta */dev* contén os *device files* correspondentes a todos os dispositivos que manexa o sistema. Tradicionalmente esa información era creada de xeito *estático*, con información de todos os dispositivos posibles que sexa capaz de manexar o núcleo, estén ou non presentes nese momento.

Algúns dos problemas que plantexa esta solución son os seguintes [6]:

- A carpeta */dev* crece desmesuradamente é faise difícil de manexar
- O listado de *major codes* e *minor codes* posibles (números entre 0 e 255) estábase esgotando.
- O feito de que o os nomes dos dispositivos dependa da orde de conexión ao sistema, complica a xestións dos usuarios. Por exemplo, o segundo disco pode pasar de chamarse */dev/sdb* a */dev/sda* se se desconecta o primeiro disco.
- Os programas de usuario poderian mellorar se puidesen detectar o momento da conexión de novos dispositivos e tomar accións específicas nese momento.

[5] http://etutorials.org/Linux+systems/red+hat+linux+9+professional+secrets/Part+IV+Managing+Red+Hat+Linux/Chapter+20+Advanced+System+Administration/Managing+Devices/

[6] https://es.wikipedia.org/wiki/Udev

Por iso no *kernel* 2.3.46 apareceu o sistema *devfs* que solucionaba algúns destes problemas, pero non todos. Dende o *kernel* 2.6 emprégase o sistema *udev* que soluciona todos estes problemas.

6.3.1. Fundamento do sistema *udev*

O sistema udev ten os seguintes obxectivos [7]:

- Execución en espazo de usuario. Ao separalo do *kernel* reducimos o tamaño deste o que é beneficioso por exemplo para sistemas embebidos.
- Empregar nomes de dispositivos persistentes e permitir reglas específicas para cada dispistivo conectado.
- Facer que a carpeta */dev* se actualice dinamicamente, contendo so nomes de dispositivos conectados ao sistema
- Proporcionar unha API que permita acceder á información dos dispositivos dende o espazo de usuario.

Para entender o funcionamento do sistema *udev* imos explicar os seus tres piares fundamentas.

O sistema de arquivos virtuais *sys*

Como xa comentamos nun tema previo existen varias carpetas *virtuais* dentro do sistema de arquivo de Linux. Unha delas é a carpeta */sys* que contén información do sistema, dos dispositivos do sistema. Esta carpeta é creada polo núcleo ao arrancar o sistema e actualízase dinámicamente cada vez que haxa un cambio no estado dos dispositivos. Pode ser que se conecte ou desconecte un dispositivo, ou que cambie algún dos seus atributos.

A vantaxe é que esta carpeta é accesible dende o espazo de usuario, o que fai que os procesos deste podan acceder a esa información, non só os do *kernel*. Esto permitirá por exemplo ao sistema *udev* acceder a esa información.

Estructurado en formato de árbore de arquivos atoparemos información como os *major* e *minor codes*, o nome do fabricante do dispositivo, o seu tamaño, datos do bus ao que está conectado, permisos, o seu nome único (UUID), etc.

[7] https://www.linux.com/news/udev-introduction-device-management-modern-linux-system/

A canle de comunicación *Netlink*

Netlink é un protocolo de comunicación entre procesos (un *socket*), presente no *kenel* de Linux dende a versión 2.2. A súa misión é comunicar eventos que suceden no sistema, e que en principio so sería accesibles para o *kernel*, ao espazo de usuario. Unha especie de pasarela que vai a permitir que procesos do *kernel* se comuniquen con procesos do espazo de usuario.

Esto é o que sucede por exemplo cando se conecta un novo dispositivo ao sistema. Nese momento *Netlink* enviará unha mensaxe ao demonio *udevd* do que falaremos a continuación, que se executa no espazo de usuario, para que poda actuar en consecuencia.

O demonio *udevd*

O demonio *udevd* execútase no espazo de usuario. A súa misión é escoitar os eventos transmitidos por *netlink* informando de cambios nos dispositivos do sistema. As decisións a tomar virán determinadas por un sistemas de regras, cuxa sintaxis explicaremos na seguinte sección.

Hai que ter en conta que *udevd* vai ter acceso a información dos dispositivos a través da carpeta */sys*. Esto vai permitir que as súas reglas podan incluir condicionantes sobre calquera das características dos dispositivos.

Podería ser, por exemplo, que cando reciba a mensaxe de desconexión dun dispositivo comprobe se se trata da cámara chamada *camara2*, empregada para vixiar unha zona. E nese caso poderia ordear que se execute un proceso específico, que inclua o envio dun correo electrónico ao administrador.

A continuación imos explicar a sintaxe básica das regras no sistema *udev* e a poñer algúns exemplos prácticos de regras.

6.3.2. Regras *udev*

As reglas *udev*, como todas as reglas teñen unha parte de condicional, o antecedente, e unha parte operativa, o consecuente. Ou sexas as regras serán do tipo"se ... e se ... entón ...".

Supoñamos por exemplo que queremos que cando encenda a impresora que teño conectada polo porto USB ao meu ordenador se comprobe a hora, e se esta é anormal, fora do horario de oficina, se me envie un correo electrónico de aviso.

Cando encenda a miña impresora este evento será capturado polo kernel e enviado mediante *Netlink ao espazo de usuario*.

Para ver o tipo de eventos que se xeneran nesta situación (chamados *uevents*) é moi útil o comando **udevadm** [8]. En concreto a orde *udevadm monitor* abre un monitor que escribe na consola menssaxes con información sobre os *uvents* transmitidos. No caso mencionado anteriormente aparece algo como o seguinte:

```
KERNEL add      /devices/pci0000:00/0000:00:14.0/usb1/1-9 (usb)
KERNEL add      /devices/pci0000:00/0000:00:14.0/usb1/1-9/1-9:1.0 (usb)
KERNEL bind     /devices/pci0000:00/0000:00:14.0/usb1/1-9 (usb)
UDEV   add      /devices/pci0000:00/0000:00:14.0/usb1/1-9 (usb)
KERNEL add      /bus/usb/drivers/usblp (drivers)
KERNEL add      /module/usblp (module)
....
```

Como vemos emiténse mensaxe do tipo "add" e tamén de tipo "bind". Vemos tamén que uns son emitidos polo *kernel* e outros polo sistema *udev*. Non imos entrar en detalle na explicación de todos os eventos porque é complexo e esta fora da profundidade coa que queremos abordar este capítulo.

Outro comando interesante para ter información dos dispositivos conectados ao sistema é *udevadm info* que permite extraer información dun dispositivo concreto. Se queremos obter información non so do dispositivo senón tamén de toda a "cadea" ata a raiz deberemos empregar a opción *-a*.

Por exemplo no caso da impresora estará o propio dispositivo, o porto USB ao que está conectada, o bus USB, o bus "pai" PCI, etc. E tamén debemos indicar o nome do dispositivo, para iso podemos empregar a opción *-p path*, sendo *path* o camiño que identifica o dispositivo na carpeta */sys*. Tamén podemos empregar a opción *-n name* sendo *name* o nome do dispositivo na carpeta */dev*.

No exemplo que nos atinxe podemos coller o *path* do dispositivo das mensaxes emitidas por *Netlink*. E concreto se facemos:

```
udevadm info -a -p /devices/pci0000:00/0000:00:14.0/usb1/1-9
```

ou

```
udevadm info -a -n /dev/usb/lp1
```

[8] https://www.freedesktop.org/software/systemd/man/udevadm.html

obteremos algo como o seguinte:

```
looking  at device '/devices/pci0000:00/0000:00:14.0/usb1/1-9':
  KERNEL=="1-9"
  SUBSYSTEM=="usb"
  DRIVER=="usb"
  ...
  ATTR{idVendor}=="04f9"
  ATTR{manufacturer}=="Brother"
  ....
looking at parent device '/devices/pci0000:00/0000:00:14.0/usb1':
  KERNELS=="usb1"
  SUBSYSTEMS=="usb"
  DRIVERS=="usb"
  ...
  ATTRS{idVendor}=="1d6b"
  ATTRS{manufacturer}=="Linux 5.13.0-30-generic xhci-hcd"
  ....
looking at parent device '/devices/pci0000:00':
  KERNELS=="pci0000:00"
  SUBSYSTEMS==""
  DRIVERS==""
  ....
```

Vemos que en primeiro lugar danos información da propia impresora, que pododemos identificar polo atributo *manufacturer*. Despois aparece o propio bus e asi baixando ata o bus *pci*. Hai moita mais información da que se amosa neste listado e toda esa información pode ser empregada nas regras *udev* para identificar ao dispositivo concreto.

Sintaxe das regras

As regras *udev* constan dunha serie de pares "clave-valor" separados por comas. Para a parte do antecente, para as condicións, podense empregar os seguintes operadores:

- "==" Compara a clave co valor indicado, se é igual a condición cumprese.

- "!=" Compara a clave co valor indicado, se non é igual a condición cumprese.

Para a parte do consecuente, para as accións os operadores permitidos son:

- "=" Asignar un calor a unha clave.
- "+=" Engadir un valor a unha clave de tipo lista.
- ":=" Asignar un valor a unha clave e impedir cambios posteriores.

A continuación imos analizar alguns exemplos de regras.

```
 KERNEL=="video[0-9]*", SUBSYSTEM=="video4linux", SUBSYSTEMS=="usb",
ATTRS{idVendor}=="05a9", ATTRS{idProduct}=="4519", SYMLINK+="video-cam"
```

Este primeir exemplo [9] identifica unha *webcam* concreta (fixarse en todas as condicións) e engadelle como nome de dispositivo *video-cam*. Se mirasemos o contido da carpeta */dev* despois diso veriamos que aparece */dev/video-cam*, como enlace simbólico ao nome "primario" do dispositivo.

Outro exemplo sacado da mesma páxina:

```
ACTION=="remove", SUBSYSTEM=="usb", ENV{ID_VENDOR_ID}=="05a9",
ENV{ID_MODEL_ID}=="4519", RUN+="/path/to/your/script"
```

Neste caso é unha regra que se executará cando se retire o dispositivo USB indicado. Neste caso a acción é engadir á posible lista de executables (*RUN+=*) un script concreto.

E un terceir exemplo sacado da mesma fonte:

```
ACTION=="add", SUBSYSTEMS=="usb", SUBSYSTEM=="block",
ENV{ID_FS_USAGE}=="filesystem", RUN{program}+="/usr/bin/systemd-mount
--no-block --automount=yes --collect $devnode /media"
```

Neste caso, a regra lánzase cando se detecta a conexión dun dispositivo USB de tipo "bloque" e para o cal a variable *ID_FS_USAGE* vale "filesystem". Nese caso execútase o comando, que indica como montar o novo dispositivo. Ademais das opcións que non imos a comentar fixarse en que monta o dispositivo indicado pola variable *$devnode* en */media*.

[9] https://wiki.archlinux.org/title/udev