Acesso a dispositivos

Dispositivos

Um computador não serve para muita coisa se não puder interagir com o mundo que o cerca, recebendo dados externos, armazenando informações e produzindo resultados para seus usuários. Toda essa interação é baseada em dispositivos de entrada/saída. Teclado, mouse, interface de rede, monitor, portas USB são exemplos de dispositivos típicos que um núcleo de sistema deve acessar e gerenciar para os processos.

Tipos de dispositivos

Apesar da grande diversidade de dispositivos físicos existentes, o UNIX classifica a maioria dos dispositivos com os quais interage em três grandes grupos, cuja estrutura acaba se refletindo na organização do núcleo e nos drivers de dispositivos:

- **Dispositivos de caracteres** (*character devices*): um dispositivo de caracteres pode ser acessado com um fluxo (*stream*) seqüencial de bytes, sendo visto da mesma forma que um arquivo de acesso seqüencial. Uma característica importante desse tipo de dispositivo é que, uma vez lido um byte, ele não pode ser "devolvido" ao dispositivo na mesma posição. São exemplos típicos desse tipo de dispositivo a console, os terminais e as portas seriais e paralelas. Um *driver* para dispositivos de caracteres implementa usualmente as chamadas de sistema `open`, `close`, `read` e `write`. Para alguns dispositivos a chamada `lseek` também está disponível.
- **Dispositivos de blocos** (*block devices*): um dispositivo de bloco permite acesso a informações em blocos de tamanho fixo (usualmente 512 ou 1024 bytes), permitindo o acesso a um ou mais blocos em cada operação. Como o acesso é aleatório, os dados podem fluir em ambas as direções a qualquer instante, Por permitir acesso aleatório e a transferência de dados em blocos, um dispositivo de blocos geralmente é adequado para a construção de um sistema de arquivos. Discos rígidos, disquetes e CDs são exemplos típicos de dispositivos de blocos. As mesmas chamadas de sistema acima citadas são oferecidas para dispositivos de blocos.
- **Interfaces de rede** (*network interfaces*): uma interface de rede é capaz de enviar e receber informações de outros computadores. A informação é transportada em pacotes de dados de tamanho variável. Este modelo não se encaixa com o anteriores, por isso esses dispositivos são geralmente tratados à parte pelo sistema. Ao invés de ler e escrever blocos ou caracteres, as chamadas de sistema relacionadas às interfaces de rede estão relacionadas ao envio e recepção de pacotes de rede.

Além das classes tradicionais acima, uma nova classe de dispositivos vem ganhando espaço nos sistemas de computação, embora ainda seja vista como um caso especial de dispositivo de blocos:

- **Dispositivos de fluxo** (*stream devices*): um dispositivo de fluxo recebe ou produz dados em uma taxa regular e constante. Esta classe de dispositivos mapeia bem o comportamento de placas de som e de vídeo. O sistema operacional precisa gerenciar buffers para alimentar ou retirar dados desse tipo de dispositivo.

Drivers

Grande parte das chamadas de sistema estão relacionadas a dispositivos físicos. Cada um desses dispositivos exige uma programação específica para ser acessado e controlado. O núcleo do sistema operacional precisa conter código para acessar e controlar todos os dispositivos com os quais possa se relacionar diretamente. O código que efetua o acesso e controle de um dispositivo físico, interagindo diretamente com ele, é chamado de

driver de dispositivo (*device driver*).

O *driver* de dispositivo tem vários objetivos:

- Interagir com o dispositivo, permitindo enviar e/ou receber dados do mesmo
- Controlar as funcionalidades e possibilidades de configuração do dispositivo
- Integrar a interface do dispositivo ao restante do núcleo, através de uma interface abstrata.

Na medida do possível, a interface abstrata de dispositivo provida pelo *driver* deve ser homogênea com as interfaces abstratas de outros dispositivos com finalidades similares. Isso permite agrupar os dispositivos em uma mesma "família", oferecendo uma única interface para o usuário. Essa é a abordagem seguida pelos diversos tipos de discos, por exemplo.

É importante observar que o driver deve oferecer **mecanismos** de acesso aos dispositivos, mas não devem impor **políticas** de acesso aos mesmos. O *driver* deve proporcionar acesso aos dispositivo sem restringir suas funcionalidades. Toda a definição de política de acesso deve ser deixada a cargo de camadas mais altas do sistema operacional.

Os *drivers* são parte integrante do núcleo do sistema operacional. Eles podem estar **compilados estaticamente** no núcleo, sendo carregados juntamente com este na inicialização da máquina, ou podem ser carregados posteriormente, sob demanda, na forma de **módulos ligados dinamicamente** ao núcleo. O Linux suporta ambos os modelos de drivers.

Muitas vezes os *drivers* são complementados por **bibliotecas e ferramentas em modo usuário** (fora do núcleo) para facilitar a configuração do dispositivo e seu uso pelos programadores. Por exemplo, o programa `tunelp` permite ajustar parâmetros da porta paralela, senquanto o comando `setserial` permite configurar as portas seriais.

Arquivos de dispositivos e o diretório /dev

No UNIX a maioria das abstrações do núcleo é oferecida aos processos no espaço de usuário sob a forma de arquivos ou de descritores de arquivos (exemplos clássicos dessa filosofia são os *sockets* e *pipes*). Da mesma forma, um dispositivo físico é mapeado para os processos sob a forma de um ou mais arquivos, que podem ser acessados em leitura e/ou escrita. Esses arquivos não correspondem a dados em disco, mas a portas de acesso aos dispositivos, controladas por seus respectivos drivers.

Assim, as entradas de diretório `/dev/audio` e `/dev/audioctl` correspondem à placa de som do computador. Um processo desejando fazer uso da placa de som deverá abrir e ler/escrever nesses "arquivos" (que tratam respectivamente dos dados e do controle da placa de som). O driver da placa de som será o responsável por implementar as chamadas `open`/`read`/`write`/`close` sobre esses arquivos, mapeando essas chamadas em operações sobre os dispositivos físicos.

Em um sistema UNIX (e portanto no Linux), os arquivos de dispositivos estão concentrados no diretório `/dev`. Exemplos de alguns arquivos ali encontrados:

arquivo	**dispositivo**
`/dev/hda`	hard disk A
`/dev/hda3`	hard disk A, partition 3
`/dev/audio`	sound card (data, channel 0)
`/dev/audioctl`	sound card (control)
`/dev/cua1`	serial port 2 (COM2)
`/dev/fb0`	first frame buffer
`/dev/fd0h1440`	floppy disk 0, high density, 1440 KBytes
`/dev/ippp3`	4th syncPPP device

arquivo	dispositivo
/dev/nst1	2nd SCSI tape
/dev/parport1	2nd parallel port
/dev/ptye0	pseudo-terminal
/dev/sda2	SCSI disk A, partition 2
/dev/tty1	virtual console 1

No Linux, a Linux Assigned Names And Numbers Authority mantém uma relação de dispositivos mapeados no diretório /dev e seus respectivos nomes e *major/minor numbers*.

Na listagem de diretório é possível observar o tipo de cada dispositivo: orientado a blocos (b) ou a caracteres ©:

```
crw-------    1 root      14,   4 Jan 30  2003 /dev/audio
crw-rw----    1 uucp       5,  64 Jan 30  2003 /dev/cua0
brw-rw----    1 6          3,   1 Jan 30  2003 /dev/hda1
brw-rw----    1 6          8,   2 Jan 30  2003 /dev/sda2
```

Alguns arquivos presentes no diretório /dev não correspondem a dispositivos físicos externos, mas a dispositivos lógicos providos pelo núcleo para oferecer serviços específicos às aplicações. Alguns exemplos de dispositivos desse gênero são:

Nome	Dispositivo
/dev/mem	acesso à memória física
/dev/kmem	acesso à memória do núcleo
/dev/null	dispositivo nulo: gera E0F (*end-of/file*) ao ler e descarta dados ao gravar (útil para descartar saídas de comandos ou scripts)
/dev/port	acesso às portas de I/O
/dev/zero	Fonte de zeros: sempre retorna char (0) ao ler o arquivo
/dev/full	Sempre retorna o erro ENOSPC ao escrever
/dev/random	gerador de números aleatórios

Major & Minor numbers

Os dispositivos são identificados no diretório /dev por seu nome, seu tipo (*block/character*) e por dois números associados a cada arquivo, denominados *major & minor numbers*.

Os *major numbers* são associados ao *driver* responsável pelo dispositivo, que geralmente controla um ou mais dispositivos assemelhados. Por exemplo, os dispositivos /dev/null e /dev/zero têm major number 1, terminais seriais e consoles têm *major number* 4, discos IDE na controladora primária têm *major number* 3, e assim por diante. O núcleo usa o *major number* para escolher o *driver* para o qual deve enviar um pedido de acesso ao dispositivo.

O *minor number* é usado somente pelo *driver*, para discernir qual dos dispositivos que controla está sendo referenciado por um determinado pedido. A listagem a seguir ilustra os *major & minor numbers* dos discos hda e hdb, ligados à controladora IDE primária, e algumas de suas respectivas partições:

```
brw-rw----    1 6          3,   0 Jan 30  2003 /dev/hda
brw-rw----    1 6          3,   1 Jan 30  2003 /dev/hda1
brw-rw----    1 6          3,   2 Jan 30  2003 /dev/hda2
brw-rw----    1 6          3,   3 Jan 30  2003 /dev/hda3
...
brw-rw----    1 6          3,  64 Jan 30  2003 /dev/hdb
brw-rw----    1 6          3,  65 Jan 30  2003 /dev/hdb1
brw-rw----    1 6          3,  66 Jan 30  2003 /dev/hdb2
```

```
...
```

Nos núcleos Linux até a versão 2.4, os *major & minor* numbers variam entre 0 e 255. A atribuição de *major numbers* a novos dispositivos e seus respectivos drivers é gerenciada pela LANANA e está indicada em sua relação de dispositivos (uma relação simplificada está presente em `/usr/src/linux/include/linux/major.h`). Os *major numbers* entre 240 e 254 são reservados para uso local e podem ser usados em projetos específicos. Os *minor numbers* podem ser definidos pelos *drivers* sem restrições.

Geralmente os arquivos presentes em `/dev` são criados por default durante a instalação do sistema. Os arquivos default podem ser criados/restaurados através do utilitários `MAKEDEV` e `MAKEDEV.local` presentes nesse diretório. Também é possível criar arquivos isoladamente, através do utilitário `mknod`. Por exemplo, o comando

```
$ mknod /dev/ttyS0 c 4 64
```

irá criar um arquivo para um dispositivo orientado a caracteres chamado ttyS0, com *major number* 4 e *minor number* 64 (esse dispositivo corresponde à porta serial COM1). Após a criação do arquivo, seu usuário, grupo e permissões devem ser ajustados de acordo com a política de acesso ao dispositivo.

Importante: a existência de um arquivo de dispositivo não depende nem implica na existência do dispositivo físico correspondente, nem de seu *driver*. O exemplo abaixo ilustra isso:

```
$ mknod xxx0 c 244 3

$ ll xx*
crw-r--r--    1 root     root     244,   3 Out 18 23:49 xxx0

$ cat xxx0
cat: xxx0: Dispositivo inexistente
```

Alguns mecanismos foram propostos para eliminar a necessidade de criar manualmente arquivos para representar dispositivos. Uma das iniciativas mais importantes foi o *device filesystem* (*devfs*), no qual os drivers podem registrar os pontos de acesso aos dispositivos sob sua responsabilidade. Dessa forma, a criação e remoção dos arquivos em `/dev` acontece de forma automática, quando da ativação ou remoção do *driver*. Além disso, o diretório `/dev` somente conterá arquivos correspondentes a dispositivos existentes e ativos no sistema local.

Com o advento do núcleo Linux 2.6, o sistema *devfs* foi considerado obsoleto, sendo substituido pelo sistema *udev*. Este último sistema é bastante similar ao *devfs* em seu conceito, mas trabalha inteiramente fora do núcleo, no espaço de usuário, sendo portanto menos intrusivo.

Controle de dispositivos

Além de permitir a entrada e/ou saída de dados sobre os dispositivos, um *driver* deve ainda exercer funções de gerência sobre o mesmo, permitindo que opções de controle e configuração sejam aplicadas por processos de usuário autorizados a configurar ou controlar o dispositivo. Alguns exemplo desse tipo de controle são:

- mudar o fonte usado em um terminal
- ejetar um CD
- rebobinar uma fita magnética
- colocar uma interface de rede em modo promíscuo

As principais formas de configurar e/ou controlar um dispositivo no mundo UNIX são:

- através de comandos identificados por seqüências de caracteres ou bytes reservados no fluxo de dados escrito no dispositivo. O *driver* reconhece essas seqüências e aplica os controles correspondentes sobre os dispositivos. Essa técnica é utilizada sobretudo no controle de terminais, onde as conhecidas "seqüências de escape" permitem modificar configurações e controlar a posição do cursor, por exemplo.
- através da chamada de sistema `ioctl` (*IO Control*), que permite enviar comandos específicos para o *driver*.

A chamada `ioctl` está descrita a seguir:

```
#include <sys/ioctl.h>
int ioctl (int filedes, int command, ...)
```

Essa chamada aplica o comando `command` sobre o dispositivo indicado pelo descritor de arquivo aberto `filedes`. O terceiro parâmetro é opcional e depende do comando aplicado, podendo ser um número ou ponteiro para uma estrutura (específica para cada comando). A chamada retorna -1 em caso de falha.

Os comandos são também chamados *IOCTLs* e são definidos por constantes inteiras. A grande maioria dos *IOCTLs* é específica para cada dispositivo e sua definição também depende do sistema operacional e variante considerados. A melhor fonte de informação a respeito dos comandos disponíveis sobre um determinado dispositivo são as páginas de manual ou documentação do dispositivo e seu respectivo *driver*. Um exemplo é a página de manual dos dispositivos `/dev/fd*` (`man 4 fd`). Outra boa fonte de informação sobre os comandos possíveis são os arquivos-fonte do núcleo do sistema. Por exemplo, o arquivo `/usr/src/include/linux/cdrom.h` traz as *IOCTLs* definidas para leitores de CD. Ao lê-lo, pode-se deduzir o comando necessário para ejetar um CD que está inserido em `/dev/hdc` (disco mestre da controladora IDE secundária):

```
...
cd = open ("/dev/hdc", O_RDWR) ;
ioctl (cd, CDROMEJECT, 0) ;
...
```

O sistema /proc

Além dos dispositivos externos, outro foco de atenção do programador de sistema é o núcleo do sistema operacional. O diretório virtual `/proc` contém uma hierarquia de arquivos e diretórios que representam a situação atual do sistema de forma estruturada e detalhada. Alguns arquivos também aceitam operações de escrita por usuários autorizados, permitindo mudar características e configurações do núcleo sem o auxílio de utilitários especiais (muitos dos utilitários como `ps`, `top`, `free` e `uptime` retiram suas informações de arquivos presentes em `/proc`).

Como os arquivos desse diretório são virtuais (não estão mapeados em disco), seu tamanho normalmente é indicado como zero (0). Apesar disso, é possível consultá-los usando os utilitários padrão `cat`/`more`. Por exemplo, pode-se conhecer as portas de IO ativas no sistema consultado o arquivo `/proc/ioports` (observe que a listagem é hierarquizada):

```
espec:/> cat /proc/ioports
0000-001f : dma1
0020-003f : pic1
0040-005f : timer
0060-006f : keyboard
0070-007f : rtc
0080-008f : dma page reg
00a0-00bf : pic2
00c0-00df : dma2
```

```
00f0-00ff : fpu
01f0-01f7 : ide0
03c0-03df : vga+
03f6-03f6 : ide0
03f8-03ff : serial(auto)
08b0-08bf : ServerWorks CSB5 IDE Controller
   08b0-08b7 : ide0
   08b8-08bf : ide1
08c0-08c3 : ServerWorks CSB5 IDE Controller
0cf8-0cff : PCI conf1
cce0-ccff : Intel Corp. 82544EI Gigabit Ethernet Controller (Copper)
   cce0-ccff : e1000
d800-d8ff : Adaptec AIC-7899P U160/m (#2)
dc00-dcff : Adaptec AIC-7899P U160/m
e800-e8ff : ATI Technologies Inc Rage XL
ecc0-ecff : Intel Corp. 82557/8/9 [Ethernet Pro 100]
   ecc0-ecff : e100
```

A grande maioria dos arquivos em `/proc` é acessível somente em leitura, mas é possível alterar o conteúdo de alguns deles através de redireções ("`echo valor > arquivo`"). Por exemplo:

```
$ echo www.teste.com > /proc/sys/kernel/hostname
```

esse comando permite alterar dinamicamente o *hostname* do sistema. Em outro exemplo:

```
$ echo 1 > /proc/sys/net/ipv4/ip_forward
```

esse comando ativa a funcionalidade de encaminhamento de pacotes (*packet forwarding*) da pilha TCP/IP do sistema. Alguns dos arquivos mais relevantes encontrados em `/proc` estão listados a seguir:

arquivo	**conteúdo**
`/proc/procinfo`	Descrição do(s) processador(es) do computador
`/proc/devices`	Lista dos drivers de dispositivos configurados (*block* & *character*)
`/proc/dma`	Canais DMA em uso
`/proc/filesystems`	Sistemas de arquivos suportados pelo núcleo
`/proc/interrupts`	Mostra interrupções em uso e quantas vezes cada uma ocorreu
`/proc/ioports`	Portas de E/S em uso no momento
`/proc/ksyms`	Tabela de símbolos do núcleo
`/proc/meminfo`	Informações sobre o uso da memória
`/proc/modules`	Módulos carregados (equivale ao comando `lsmod`)
`/proc/net/`	Muitas informações sobre redes, protocolos, interfaces, etc.

A maior parte do diretório `/proc` é ocupada por diretórios cujos nomes são números, que correspondem aos PIDs dos processos existentes no sistema. Em cada diretório de processo, as seguintes informações estão presentes:

arquivo	**conteúdo**
`/proc/`*pid*`/cmdline`	Linha de comando usada para lançar o processo
`/proc/`*pid*`/cpu`	CPU atual e última CPU usada
`/proc/`*pid*`/cwd`	Link para o diretório de trabalho
`/proc/`*pid*`/environ`	Valores das variáveis de ambiente
`/proc/`*pid*`/exe`	Link para o executável
`/proc/`*pid*`/fd/`	Diretório contendo os descritores de arquivos abertos
`/proc/`*pid*`/maps`	Mapas de memória para o executável e bibliotecas

arquivo	**conteúdo**
`/proc/`*pid*`/mem`	Memória ocupada pelo processo
`/proc/`*pid*`/root`	Link ao diretório root (/) do processo
`/proc/`*pid*`/stat`	Status do processo
`/proc/`*pid*`/statm`	Status da memória do processo
`/proc/`*pid*`/status`	Status do processo em formato texto, legível por humanos

Atividades

- Com base nas informações disponíveis no sistema `/proc`, construa um programa que gere a mesma saída que o utilitário `uptime`.
- Qual seria a invocação de chamada necessária para rebobinar a fita magnética 0 ?