

Capítulo 6

Controlando dispositivos com IOCTL

Como continuação do anterior, este capítulo explica o conceito de IOCTL (I/O Control) e demonstra, em alguns exemplos, esta chamada de sistema que complementa o paradigma Unix de acesso aos dispositivos como arquivos.

6.1 Acesso a dispositivos

O conceito de acesso a dispositivos usado no Unix e seus clones, em nosso caso, o Linux, é uma abstração para que se comportem como arquivos, com operações normais, tais como `open`, `read`, `write`. Como os dispositivos podem apresentar características diferentes entre si, deve haver um modo de trabalhar estas diferenças, que nem sempre são na abertura, na escrita ou no recebimento de dados. Para tanto, existe uma operação denominada **`ioctl`**.

Cada driver contém, sempre que necessário, um conjunto de comandos que podem ser utilizados por meio da chamada de um `ioctl`.

No capítulo 4 utilizamos `ioctl` para acessar a porta serial e modificar suas características. Este é um dos casos mais comuns de aplicação.

Para demonstrar este princípio de comandos, dois exemplos: o primeiro utilizando o drive de CD-ROM e o segundo, com os LEDs do teclado.

O exemplo dos LEDs do teclado mostra de forma correta como fazer o que foi demonstrado no capítulo 3, quando acessamos diretamente o controlador do teclado.

6.2 Manipulação do drive de CD-ROM

Além das operações normais de arquivo, o driver do kernel `cdrom.o` provê alguns comandos para o controle, tal como selecionar faixa, iniciar o play, pause e eject. Com isso, podemos montar um programa simples, que apenas controla algumas operações básicas do CD-ROM, sem se preocupar com a parte de áudio, que pode ser controlada do seu mixer predileto. Se o aparelho estiver conectado corretamente à entrada digital de áudio de sua placa de som, a reprodução deve funcionar normalmente.

O objetivo é demonstrar como enviar comandos a um dispositivo, e o nosso player vai carecer de várias características básicas presentes na concorrência. Por isso, o nome dele não é nem player, é `CDControl`.

`cdcontrol.c`

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/ioctl.h>
#include <fcntl.h>
#include <linux/cdrom.h>

#define MAXCOM 4      /* tamanho máximo do comando */
void printheIp();
void cd_play(int fd);
void cd_stop(int fd);
void cd_pause(int fd);
void cd_resume(int fd);
void cd_ejec(int fd);

int main (int argc, char **argv) {

    char comando[256];
    int fd;

    /* Checa se algum parâmetro foi passado */
    if (argc <= 1) {
        fprintf(stderr, "Uso : %s /dev/cdrom ou device correto\n", argv[0]);
        exit(-1);
    }
    if ((fd = open(argv[1], O_RDONLY)) < 0) {
        perror("open");
        exit(1);
    }

    fprintf(stdout, "Cd control - digite ? para os comandos\n");
```

```

/* mantém loop enquanto não for digitado quit */
while (1) {
    memset(comando, 0, 256);          /* limpa buffer de comandos */
    fprintf(stdout, "> ");
    fgets(comando, MAXCOM + 3, stdin);
    if (comando[0]!='?') printhelp();
    else if (!strncmp("play", comando, MAXCOM)) cd_play(fd);
    else if (!strncmp("stop", comando, MAXCOM)) cd_stop(fd);
    else if (!strncmp("paus", comando, MAXCOM)) cd_pause(fd);
    else if (!strncmp("resu", comando, MAXCOM)) cd_resume(fd);
    else if (!strncmp("ejec", comando, MAXCOM)) cd_ejec(fd);
    else if (!strncmp("quit", comando, MAXCOM)) break;
    else fprintf(stdout, "comando invalido\n");
}
fprintf(stdout, "Fechando o device\n");
close (fd);
}

void printhelp() {
    fprintf(stdout, "CDControl \n");
    fprintf(stdout, "play - Inicia \n");
    fprintf(stdout, "stop - Para \n");
    fprintf(stdout, "paus - Pause \n");
    fprintf(stdout, "resu - Volta da pausa \n");
    fprintf(stdout, "ejec - Ejeta o CD\n");
}

void cd_play(int fd) {
    struct cdrom_tochdr toc_header;
    struct cdrom_ti track_index;
    int io;
    /* recupera informações do TOC do cd (table of contents) */
    io = ioctl(fd, CDROMREADTOCHDR, (void *) &toc_header);
    if (io == -1) {
        perror("ioctl: cdromreadtochdr");
        return;
    }
    /* preenche o índice de faixas a serem tocadas. no caso, todas */
    fprintf(stdout, "Faixa inicial: %d - Faixa final %d\n", toc_header.cdth_trk0,
    toc_header.cdth_trk1);
    track_index.cdti_trk0=toc_header.cdth_trk0;
    track_index.cdti_ind0=0;
    track_index.cdti_trk1=toc_header.cdth_trk1;
    track_index.cdti_ind1=255;
    /* aciona o play */
    io=ioctl(fd, CDROMPLAYTRKIND, (void *) &track_index);
    if (io==-1) {
        perror("ioctl: cdromplaytrkind");
        return;
    }
}
}

```

```
void cd_stop (int fd) {
    if ((ioctl(fd,CDROMSTOP,0)) == -1)
        perror("ioctl: CDROMSTOP");
}
void cd_pause (int fd) {
    if ((ioctl(fd,CDROMPAUSE,0)) == -1)
        perror("ioctl: CDROMPAUSE");
}
void cd_resume (int fd) {
    if ((ioctl(fd,CDROMRESUME,0)) == -1)
        perror("ioctl: CDROMRESUME");
}
void cd_ejec (int fd) {
    if ((ioctl(fd,CDROMEJECT,0)) == -1)
        perror("ioctl: CDROMEJECT");
}
```

Para compilar, digite em um arquivo chamado de `cdcontrol.c` e execute:

```
gcc cdcontrol.c -o cdcontrol -O2
```

Se tudo foi digitado corretamente, não deve haver nenhum erro na compilação.

Agora, verifique as permissões do dispositivo que corresponde ao seu CD-ROM, ou execute como root: `./cdcontrol /dev/cdrom`.

O dispositivo varia de acordo com a máquina de cada um, ou, ainda, se existe um link `/dev/cdrom`, este pode ser utilizado. Se não houver um CD no drive, o programa abortará, indicando um erro do tipo “no media found”. Como foi dito, o programa é bem simples e não inclui muitas checagens para não poluir o código e distrair do objetivo principal.

Após a execução com um CD no drive, deve aparecer a mensagem de entrada e o prompt:

```
Cd control - digite ? para os comandos
>
```

Digitando `?`, a lista de comandos disponíveis será exibida. Caso o CD no drive seja de dados, o comando `play` emitirá uma mensagem de erro, pois estes CDs são manipulados de forma diferente. Sendo um CD de áudio, o comando `play` ativará a reprodução de áudio, que deverá ser controlado em sua intensidade e volume pelo mixer.

A seqüência de operação do programa consiste em abrir o dispositivo (`/dev/cdrom`, por exemplo), e o envio de comandos específicos por `ioctl` para cada função requisitada. Existem mais comandos do driver do que os apresentados neste exemplo.

6.3 LEDs do teclado

Utilizando `ioctl`s para trabalhar com os LEDs do teclado, apesar de existirem as funções corretas na especificação POSIX, inclusas no driver de `tty`, é outra oportunidade para demonstrar o conceito dos `ioctl`s.

Os comandos que vamos utilizar são internos ao kernel, e podem ser mudados. Portanto, caso você realmente precise incorporá-los ao seu programa, deve utilizar os comandos da especificação POSIX. O aviso é válido mesmo sabendo que tais comandos não mudam há um bom tempo.



`pisca_leds.c`

```
#include <sys/ioctl.h>
#include <linux/kd.h>

/*
 * KDSETLED - Modifica os valores dos LEDs
 * KDGETLED - Lê os valores dos LEDs
 */

int main(int argc, char **argv) {
    int leds = 0;
    /* loop variando os 3 bits mais baixos de leds */
    while (1) {
        leds = 2 * leds + 1;
        if (leds & 8)
            leds ^= 9;
        if (ioctl(0, KDSETLED, leds))
            exit(1);
        sleep(1);
    }
}
```

Este programa aplica um `ioctl` em um arquivo já aberto, que tem o `fd` (file-descriptor) de número 0. Este arquivo é justamente `stdin`, ou a entrada-padrão. Todo processo recebe 3 `fd`'s já abertos, 0 – `stdin`; 1 – `stdout` e 2 – `stderr`. Assim, aproveitamos o “dispositivo” já aberto e executamos o comando desejado. Este detalhe mostra bem a abstração entre dispositivos no Linux.

Para o programa funcionar corretamente, execute-o de um console de texto, não um xterm. O programa `set1eds`, do pacote `console-tools`, executa a mesma função, mas da forma correta, por isso deve ser a referência de código-fonte.

Sendo assim, existem comandos corretos a serem enviados para cada dispositivo, além da própria forma de trabalho, como notamos no CD-ROM, que necessita de dados para iniciar a função `play`. Esta interface deve ser explorada separadamente para cada dispositivo/finalidade desejada.

Até mesmo em situações como programando para um `socket` dispomos de vários comandos que podem ser utilizados, alguns obrigatoriamente, para configurar a conexão de forma correta.

Para dispositivos como uma placa de captura de vídeo, a operação toda é feita usando comandos enviados por `ioctl`. Assim, é mais importante conhecer o conceito de funcionamento do driver e do dispositivo, antes de programar, do que assumir que todos os drivers funcionarão de forma semelhante. A interface `ioctl` possibilita esta flexibilidade dentro do modelo de acesso a dispositivos do Unix/Linux.