一辈子只为一颗豆

xingzuzi.blog.chinaunix.net

和你一起慢慢变老，日出日落，春夏秋冬，forever

xingzuzi

博客访问： 167664
博文数量： 172
博客积分： 2998
博客等级： 少校
技术积分： 1446
用 户 组： 普通用户
注册时间： 2010-10-28 09:39

linux ioctl()函数

2011-05-22 10:18:33

分类： LINUX

我这里说的ioctl函数是指驱动程序里的，因为我不知道还有没有别的场合用到了它，所以就规定了我们讨论的范围。写这篇文章是因为我前一阵子被ioctl给搞混了，这几天才弄明白它，于是在这里清理一下头脑。

一、 什么是ioctl

ioctl是设备驱动程序中对设备的I/O通道进行管理的函数 。所谓对I/O通道进行管理，就是对设备的一些特性进行控制，例如串口的传输波特率、马达的转速等等。它的调用个数如下：

int ioctl(int fd, ind cmd, ...);

其中fd是用户程序打开设备时使用open函数返回的文件标示符，cmd是用户程序对设备的控制命令，至于后面的省略号，那是一些补充参数，一般最多一个，这个参数的有无和cmd的意义相关。

ioctl函数是文件结构中的一个属性分量，就是说如果你的驱动程序提供了对ioctl的支持，用户就可以在用户程序中使用ioctl函数来控制设备的I/O通道。

二、 ioctl的必要性

如果不用ioctl的话，也可以实现对设备I/O通道的控制，但那是蛮拧了。例如，我们可以在驱动程序中实现write的时候检查一下是否有特殊约定的数 据流通过，如果有的话，那么后面就跟着控制命令（一般在socket编程中常常这样做）。但是如果这样做的话，会导致代码分工不明，程序结构混乱，程序员自己也会头昏眼花的。所以，我们就使用ioctl来实现控制的功能。要记住，用户程序所作的只是通过命令码(cmd) 告诉驱动程序它想做什么，至于怎么解释这些命令和怎么实现这些命令，这都是驱动程序要做的事情。

三、 ioctl如何实现

这是一个很麻烦的问题，我是能省则省。要说清楚它，没有四五千字是不行的，所以我这里是不可能把它说得非常清楚了，不过如果读者对用户程序是怎么和驱动程 序联系起来感兴趣的话，可以看我前一阵子写的《write的奥秘》。读者只要把write换成ioctl，就知道用户程序的ioctl是怎么和驱动程序中 的ioctl实现联系在一起的了。我这里说一个大概思路，因为我觉得《Linux设备驱动程序》这本书已经说的非常清楚了，但是得花一些时间来看。

在驱动程序中实现的ioctl函数体内，实际上是有一个switch{case}结构，每一个case对应一个命令码，做出一些相应的操作。怎么实现这些操作，这是每一个程序员自己的事情。 因为设备都是特定的，这里也没法说。关键在于怎样组织命令码，因为在ioctl中命令码是唯一联系用户程序命令和驱动程序支持的途径 。 命令码的组织是有一些讲究的，因为我们一定要做到命令和设备是一一对应的，这样才不会将正确的命令发给错误的设备，或者是把错误的命令发给正确的设备，或 者是把错误的命令发给错误的设备。这些错误都会导致不可预料的事情发生，而当程序员发现了这些奇怪的事情的时候，再来调试程序查找错误，那将是非常困难的 事情。所以在Linux核心中是这样定义一个命令码的：

设备类型	序列号	方向	数据尺寸
8 bit	8 bit	2 bit	8~14 bit

这样一来， 一个命令就变成了一个整数形式的命令码； 但是命令码非常的不直观，所以Linux Kernel中提供了一些宏。 这些宏可根据便于理解的字符串生成命令码，或者是从命令码得到一些用户可以理解的字符串以标明这个命令对应的设备类型、设备序列号、数据传送方向和数据传输尺寸。

这些宏我就不在这里解释了，具体的形式请读者察看Linux核心源代码中的宏，文件里给这些宏做了完整的定义。这里我只多说一个地方，那就是"幻数"。 "幻数"是一个字母，数据长度也是8，用一个特定的字母来标明设备类型，这和用一个数字是一样的，只是更加利于记忆和理解。 就是这样，再没有更复杂的了。 更多的说了也没用，读者还是看一看源代码吧，推荐各位阅读《Linux 设备驱动程序》所带源代码中的short一例，因为它比较短小，功能比较简单，可以看明白ioctl的功能和细节。

四、 cmd参数如何得出

这里确实要说一说，cmd参数在用户程序端由一些宏根据设备类型、序列号、传送方向、数据尺寸等生成 ， 这个整数通过系统调用传递到内核中的驱动程序，再由驱动程序使用解码宏从这个整数中得到设

备的类型、序列号、传送方向、数据尺寸等信息，然后通过switch{case}结构进行相应的操作。 要透彻理解，只能是通过阅读源代码，我这篇文章实际上只是一个引子。cmd参数的组织还是比较复杂的，我认为要搞熟它还是得花不少时间的，但是这是值得的，因为驱动程序中最难的是对中断的理解 。

五、 小结

ioctl其实没有什么很难的东西需要理解，关键是理解cmd命令码是怎么在用户程序里生成并在驱动程序里解析的，程序员最主要的工作量在switch{case}结构中，因为对设备的I/O控制都是通过这一部分的代码实现的。

****

一般的说，用户空间的IOCTL系统调用如下所示：ioctl(int fd, int command, (char *) argstruct); 因为这个调用拥有与网络相关的代码，所以文件描述符号fd就是socket()系统调用所返回的，而command参数可以是/usr/include /linux/sockios.h 头文件中的任何一个。 这些命令根据它可以解决的问题所涉及的方面而被分为多种类型。比如：

改变路由表（SIOCADDRT, SIOCDELRT）

读取或更新ARP/RARP缓存（SIOCDARP, SIOCSRARP）

一般的和网络有关的函数（SIOCGIFNAME, SIOCSIFADDR等等）

Goodies目录中包含了很多展示ioctl用法的示例程序，看这些程序的时候，注意根据ioctl的命令类型来使用具体的调用参数结构 ， 比如：和路由表相关的IOCTL用RTENTRY结构，rtentry结构是被定义在/usr/include/linux/route.h文件中的，和 ARP相关的ioctl调用到的arpreq结构被定义在/usr/include/linux/if_arp.h文件之中。网络接口相关的ioctl命 令最具有代表性的特征的是都以S或G开头，其实就是设置或得到数据，getifinfo.c程序用这些命令去读取IP地址信息，硬件地址信息，广播地址信 息和与网络接口相关的标志。对于这些ioctl，第三个参数是一个IFREQ结构体，这个结构体被定义在/usr/include/linux/if.h 头文件中。在一些情况下，新的ioctl命令可能会被使用 (除了在那个头文件中被定义的除外），比如 WAVELAN无线网卡保持着无线信号强度的信息，这些信息可能会对用户程序有用。用户程序是怎么访问到这些信息的呢？我们的第一反应就是在 sockios.h头文件中定义一个新的命令，比如SIOCGIFWVLNSS，但不幸的是，这个命令在其他的网络接口上是没有任何意义的。另外试图在其 他接口上用这个命令而并非是在无线网口上用会出现违规访问，我们需要的是定义新特性接口命令的机理。幸运的是，LINUX操作系统为此目的而内置了钩子， 如果你再看一下sockios.h这个头文件，你会注意到每一个设备都有一个预定义的SIOCDEVPRIVATE命令，实现它的任务就全权交给了写这个 设备驱动的程序员了。根据常规约定，一个用户程序调用一个特定的ioctl命令如 下：ioctl(sockid, SIOCDEVPRIVATE, (char *) &ifr); 这里ifr是一个ifreq结构体变量，它用一个和这个设备联系的接口名称来填充ifr的ifr NAME域 ，比如前述的无线网卡接口名称为eth1。

不失一般性，一个用户程序将同样要与内核交换命令参数和操作结果，而这些已经通过了对域ifr.ifr_data的填充而做到 了。比如这个网卡的信号强度信息被返回到这个域当中。LINUX源代码已经包含了两个特殊设备：de4x5和ewrk3，他们定义和实现了特殊的 ioctl命令。这些设备的源代码在以下的文件中：de4x5.h, de4x5.c, ewrk3.h, ewrk3.c。两个设备都为在用户空间和驱动间交换数据定义了他们自己的私有结构，在ioctl之前，用户程序需填充需要的数据并且将 ifr.ifr_data指向这个结构体。

在进入代码前， 让我们跟踪一下处理ioctl系统调用的若干步骤。 所 有接口类型的ioctl请求都导致dev_ioctl()被调用，这个ioctl仅仅是个包装，大部分的真实的操作留给了dev_ifsioc()，这个dev_ioctl()要做的唯一一件事情就是检查调用过程是否拥有合适的许可去核发这个命令，然后dev_ifsioc()首先要做的事情之一就是得到 和名字域ifr.ifr_name中所对应的设备结构，这在一个很大的switch语块的代码后实现。

SIOCDEVPRIVATE命令和SIOCDEVPRIVATE+15的命令参数全部交给了默认操作，这些都是switch的分支语句。这里发生的是， 内核检查是否一个设备特殊的ioctl的回调已经在设备结构中被设置，这个回调是保持在设备结构中的一个函数指针。如果回调已经被设置了，内核就会调用 它。

所以，为了实现一个特殊的ioctl，需要做的就是写一个特殊ioctl的回调，然后让device结构中的do_ioctl域指向它。 对于EWK3设备，这个函数叫做ewrk3_ioctl()，对应的设备结构在ewrk3_init()中被初始化，ewrk3_ioctl()的代码清晰的展示了ifr.ifr_data的作用，是为了在用户程序和驱动之间交换信息 。注意，内存的这个区域有双方向交换数据的作用，例如，在ewrk3驱动代码中ifr.ifr_data最初的2个字节被用做向驱动传递预想要的动作。同样第五个字节指向的缓冲区用于交换其他的信息。

当你浏览ewrk3_ioctl()代码的时候，记住在一个应用中用户空间的指令是无法访问内核空间的，由于这个原因，内核给驱动编写人员提供了2个特殊 的步骤。他们是memcpy_tofs()和memcpy_fromfs()。内核里的做法是用memcpy_tofs() 拷贝内核数据到用户空间，类似的memcpy_fromfs()也是这样的，只是他拷贝用户数据到内核空间。这些程序步骤是由于调用verify_area()而被执行的，目的是确认数据访问不会违法。同样需要记住printk()的用法是打印调试信息，这个函数和printf()很 相像，但是它不能处理浮点数据。printf()函数在内核中是不能被使用的。由printk()产生的输出被转储到了一个目录./usr/adm /messages。

****

linux系统**ioctl**使用示例
These were writed and collected by kf701,
you can use and modify them but NO WARRANTY.
Contact with me : kf_701@21cn.com
程序1：检测接口的inet_addr, netmask, broad_addr
程序2：检查接口的物理连接是否正常
程序3：测试物理连接
程序4：调节音量

**********程序1**********

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <errno.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <sys/ioctl.h>
#include <net/if.h>

static void usage()
{
  printf("usage : ipconfig interface \n");
  exit(0);
}

int main(int argc,char **argv)
{
  struct sockaddr_in *addr;
  struct ifreq ifr;
  char *name,*address;
  int sockfd;

  if(argc != 2)  usage();
   else  name = argv[1];

  sockfd = socket(AF_INET,SOCK_DGRAM,0);
  strncpy(ifr.ifr_name,name,IFNAMSIZ-1);

  if(ioctl(sockfd,SIOCGIFADDR,&ifr) == -1)
    perror("ioctl error"),exit(1);

  addr = (struct sockaddr_in *)&(ifr.ifr_addr);
  address = inet_ntoa(addr->sin_addr);
  printf("inet addr: %s ",address);

  if(ioctl(sockfd,SIOCGIFBRDADDR,&ifr) == -1)
    perror("ioctl error"),exit(1);

  addr = (struct sockaddr_in *)&ifr.ifr_broadaddr;
  address = inet_ntoa(addr->sin_addr);
  printf("broad addr: %s ",address);

  if(ioctl(sockfd,SIOCGIFNETMASK,&ifr) == -1)
    perror("ioctl error"),exit(1);
  addr = (struct sockaddr_in *)&ifr.ifr_addr;
  address = inet_ntoa(addr->sin_addr);
  printf("inet mask: %s ",address);

  printf("\n");
  exit(0);
}
```

********** 程序2**********

```
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <fcntl.h>
#include <getopt.h>
```

```
#include <sys/socket.h>
#include <sys/ioctl.h>
#include <net/if.h>
#include <stdlib.h>
#include <unistd.h>
typedef unsigned short u16;
typedef unsigned int u32;
typedef unsigned char u8;
#include <linux/ethtool.h>
#include <linux/sockios.h>

int detect_mii(int skfd, char *ifname)
{
  struct ifreq ifr;
  u16 *data, mii_val;
  unsigned phy_id;

  /* Get the vitals from the interface. */
  strncpy(ifr.ifr_name, ifname, IFNAMSIZ);

  if (ioctl(skfd, SIOCGMIIPHY, &ifr) < 0)
    {
      fprintf(stderr, "SIOCGMIIPHY on %s failed: %s\n", ifname, strerror(errno));
      (void) close(skfd);
      return 2;
    }

  data = (u16 *)(&ifr.ifr_data);
  phy_id = data[0];
  data[1] = 1;

  if (ioctl(skfd, SIOCGMIIREG, &ifr) < 0)
   {
     fprintf(stderr, "SIOCGMIIREG on %s failed: %s\n", ifr.ifr_name, strerror(errno));
     return 2;
   }

  mii_val = data[3];
  return(((mii_val & 0x0016) == 0x0004) ? 0 : 1);
}

int detect_ethtool(int skfd, char *ifname)
{
  struct ifreq ifr;
  struct ethtool_value edata;
  memset(&ifr, 0, sizeof(ifr));
  edata.cmd = ETHTOOL_GLINK;

  strncpy(ifr.ifr_name, ifname, sizeof(ifr.ifr_name)-1);
  ifr.ifr_data = (char *) &edata;

  if (ioctl(skfd, SIOCETHTOOL, &ifr) == -1)
   {
     printf("ETHTOOL_GLINK failed: %s\n", strerror(errno));
     return 2;
   }

  return (edata.data ? 0 : 1);
}

int main(int argc, char **argv)
{
  int skfd = -1;
  char *ifname;
  int retval;

  if( argv[1] )  ifname = argv[1];
   else  ifname = "eth0";

  /* Open a socket. */
  if (( skfd = socket( AF_INET, SOCK_DGRAM, 0 ) ) < 0 )
    {
```

```
        printf("socket error\n");
        exit(-1);
      }

  retval = detect_ethtool(skfd, ifname);
  if (retval == 2)
    retval = detect_mii(skfd, ifname);

  close(skfd);

  if (retval == 2)
    printf("Could not determine status\n");
  if (retval == 1)
    printf("Link down\n");
  if (retval == 0)
    printf("Link up\n");

  return retval;
}
```

**********程序3**********

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <net/if.h>
#include <linux/sockios.h>
#include <sys/ioctl.h>

#define LINKTEST_GLINK 0x0000000a

struct linktest_value {
      unsigned int    cmd;
      unsigned int    data;
};

static void usage(const char * pname)
{
  fprintf(stderr, "usage: %s <device>\n", pname);
  fprintf(stderr, "returns: \n");
  fprintf(stderr, "\t 0: link detected\n");
  fprintf(stderr, "\t%d: %s\n", ENODEV, strerror(ENODEV));
  fprintf(stderr, "\t%d: %s\n", ENONET, strerror(ENONET));
  fprintf(stderr, "\t%d: %s\n", EOPNOTSUPP, strerror(EOPNOTSUPP));
  exit(EXIT_FAILURE);
}

static int linktest(const char * devname)
{
  struct ifreq ifr;
  struct linktest_value edata;
  int fd;

  /* setup our control structures. */
  memset(&ifr, 0, sizeof(ifr));
  strcpy(ifr.ifr_name, devname);

  /* open control socket. */
  fd=socket(AF_INET, SOCK_DGRAM, 0);
  if(fd < 0 )
    return -ECOMM;

  errno=0;
  edata.cmd = LINKTEST_GLINK;
  ifr.ifr_data = (caddr_t)&edata;

  if(!ioctl(fd, SIOCETHTOOL, &ifr))
    {
      if(edata.data)
        {
          fprintf(stdout, "link detected on %s\n", devname);
```

```
          return 0;
        } else
          {
           errno=ENONET;
          }
    }

  perror("linktest");
  return errno;
}

int main(int argc, char *argv[])
{
  if(argc != 2)
    {
      usage(argv[0]);
    }
  return linktest(argv[1]);
}
```

**********程序4**********

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/ioctl.h>
#include <sys/soundcard.h>
#include <stdio.h>
#include <unistd.h>
#include <math.h>
#include <string.h>
#include <stdlib.h>
#define  BASE_VALUE 257

int main(int argc,char *argv[])
{
  int mixer_fd=0;
  char *names[SOUND_MIXER_NRDEVICES]=SOUND_DEVICE_LABELS;
  int value,i;

  printf("\nusage:%s dev_no.[0..24] value[0..100]\n\n",argv[0]);
  printf("eg. %s 0 100\n",argv[0]);
  printf("will change the volume to MAX volume.\n\n");
  printf("The dev_no. are as below:\n");

  for (i=0;i<SOUND_MIXER_NRDEVICES;i++)
    {
      if (i%3==0) printf("\n");
      printf("%s:%d\t\t",names[i],i);
    }

  printf("\n\n");

  if (argc<3)  exit(1);

  if ((mixer_fd = open("/dev/mixer",O_RDWR)))
    {
       printf("Mixer opened successfully,working...\n");
       value=BASE_VALUE*atoi(argv[2]);

       if (ioctl(mixer_fd,MIXER_WRITE(atoi(argv[1])),&value)==0)
          printf("successfully.....");
       else
          printf("unsuccessfully.....");

       printf("done.\n");
    }
  else
     printf("can't open /dev/mixer error....\n");

exit(0);
}
```

上一篇：Linux内存映射: mmap
下一篇：在Linux控制台下显示JPEG图像

2

相关热门文章

linux 常见服务端口
python无法爬取阿里巴巴的数据...
【ROOTFS搭建】busybox的httpd...
linux-2.6.28 和linux-2.6.32....
xmanager 2.0 for linux配置
linux su - username -c 命...
什么是shell
我不得不在这里问一下网站使用...
linux socket的bug??
socket编程问题：不同局域网中...

给主人留下些什么吧！~~

zgj224 2014-03-06 19:53:42

好东西，解释的很详细。。。。

回复 | 举报

评论热议

请登录后评论。

登录 注册