基于 netlink机制内核空间与用户空间通信的分析

董 昱, 马 鑫
(兰州交通大学 自动化与电气工程学院, 甘肃 兰州 730070)

摘要: 介绍了 Linux操作系统的内核空间和用户空间的概念, 给出了基于 netlink通信的相关流程图和简要程序说明。基于 netlink的内核空间与用户空间的双向通信, 可以有效地解决内核空间与用户态进程间无阻塞通信的问题。
关键词: Linux; netlink; 内核空间; 用户空间
中图分类号: TP316 文献标识码: A 文章编号: 1000- 8829(2007)09- 0057- 02

Analysis of Communications Between Kernel-Space and User-Space Based on Netlink Mechanism

DONG Yu, MA Xin
(School of Automatization and Electrical Engineering, Lanzhou Jiaotong University, Lanzhou 730070, China)

Abstract The definition of kernel-space and user-space of Linux operating system is introduced, it brings forth the flow charts and programme instruction based on netlink communications. Bidirectional communications between kernel-space and user-space based on netlink mechanism can solve the problems of nonblocking communications between kernel-space and user process effectively.
Key words Linux; netlink; kernel-space; user-space

随着对 Linux操作系统的应用日趋广泛, 以 Linux为平台开发的应用软件也越来越多。如何将 Linux操作系统的内核部分与应用程序更好的协调一致的工作, 即在某些操作中, 需要在内核态与用户空间实行双向交互通信。例如, 把在内核态采集到的数据传送到用户态的一个或若干个进程中进行处理。在 Linux2.4以后版本的内核中, 几乎全部的中断过程与用户态进程的通信都是使用 netlink套接字实现的, 因而, 这是一个需要分析和解决的问题。

1 Linux内存管理模式

对于 Linux而言, 用户进程可访问 4GB的虚拟线性内存空间。其中 0~3GB的虚拟内存地址是用户空间, 用户进程可以直接对其进行访问。3~4GB虚拟内存地址是内核空间, 存放仅用于内核访问的代码和数据, 用户对它不能进行操作[2]。

对于一个任务, 定义了 4种执行特权级别, 用来限制对任务中断进行访问[1]。其中, 在 0级运行的是操作系统的内核程序, 它所在的内存空间叫内核空间; 在 3级运行的是应用程序, 它所在的内存空间是用户空间。Linux通过系统调用或硬件中断完成从用户到内核空间的转换[4]。内核空间与用户空间示意图参见图 1。(↓表示用户级线程, L表示 LPW (lightweighted process)轻权级进程)。

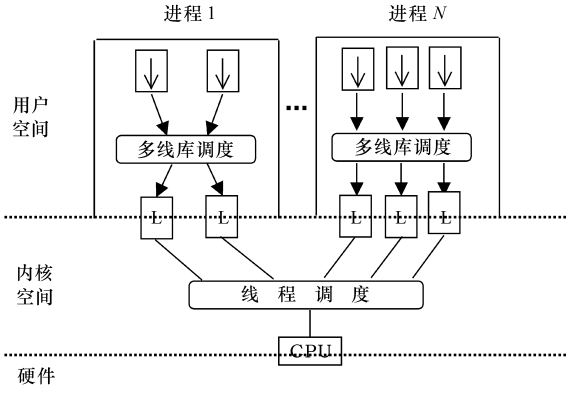

图 1 内核空间与用户空间示意图

2 内核空间与用户空间的 netlink通信

内核为了方便向用户空间发消息及调用用户程序, 封装 netlink机制提供了简单的消息广播函数向用户空间广播消息, 内核设计了建立在仿真 netlink socket上的内核 - 用户通信协议, 提供了多个内核 - 用户双向通信连接, 是一套完整的内核消息队列的无阻塞支持。

在 include/linux/netlink.h中有类似 netlink的协议簇的定义, 支持各种形式的用途的通信。其中, 0~15已被内核模块使用, 16~31由用户指定专用。内核态与用户态发送接收流程, 如图 2所示[1]。

2.1 用户空间的程序设计

(1)创建套接字。

函数 netlink_kernel_create创建 netlink类型的 socket与输入句柄连接, 并将 sock插入 nl_table链表中, 函数简要说明如

收稿日期: 2006-10-10
基金项目: 甘肃省自然科学基金资助项目 (3ZS062-B25-004)
作者简介: 董昱 (1962—), 男, 甘肃兰州市人, 教授, 主要研究方向为智能控制与并行计算。

下:

```
struct sock*  netlink_kernel_create( int unit, void(*  input)
( struct sock *  sk, int len))
```

/* 参数 unit表示 netlink协议类型,参数 input则为内核模块定义的 netlink消息处理函数,当有消息到达这个 netlink socket时,该 input函数指针就会被引用。函数指针 input的参数 sk实际上就是函数 netlink_kernel_create返回的 struct sock指针,sock实际是 socket的一个内核表示数据结构,用户空间应用创建的 socket在内核中也会有一个 struct sock结构来表示。* /

```
{ ……
  if ( sock_create_lite( PF_NETLINK, SOCK_DGRAM, unit,
& sock) )  return NULL;
  /创建 socket结构类型节点 sock
  if ( netlink_create( sock, unit) < 0)  {
     sock_release( sock);
     return NULL;
  }
  sk= sock-> sk;  /给 sock加上 netlink_ops,分配并初始化
                  sk
  sk-> sk_data_ready= netlink_data_ready;  /函数 netlink_
                  data_ready接收数据,唤醒接收等待队列
  if( input)  nlk_sk( sk) -> data_ready= input;  /赋值输入
                                          数据函数句柄
  netlink_insert( sk, 0);  //sk加入 nl_table链表中
  return sk;
  }
```

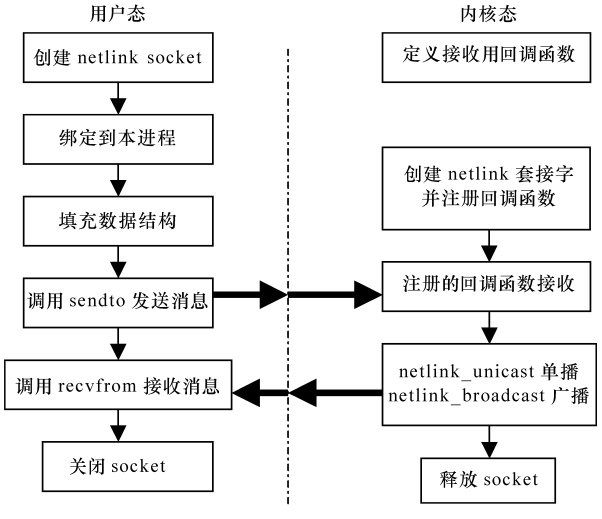

图 2　内核态与用户态发送接收流程图

(2)程序设计。

用户空间的 main()简要说明如下。

```
intmain( void)
{ ……
skfd= socket ( PF_NETLINK, SOCK_RAW, NL_MP2);
```

/* netlink对应的协议簇是 AF_NETLINK,第 2个参数必须是 SOCK_RAW或 SOCK_DGRAM,第 3个参数指定 netlink协议类型,它可以是一个自定义的类型,也可以使用内核预定义的类型。* /

```
if( bind ( skfd,  ( struct sockaddr*  )& local, sizeof( local)) !
= 0)  return- 1;
```

/* bind函数需要绑定协议地址, netlink的 socket地址使用 struct sockaddr_nl结构描述* /

```
signal( SIGINT, sig_int);    //设置 SIGINT信号为 sig_int函
数
……
sendto( skfd, &message, message. hdr. nlmsg_len, 0, ( struct
sockaddr*  )& kpeer, sizeof( kpeer));
```

/通过 socket向内核模块发消息 MP2_U_PID,下面循环是从内核接收消息

```
while( 1)
{
  kpeerlen= sizeof( struct sockaddr_nl);
  recvlen= recvfrom ( skfd, & info, sizeof( struct u_packet_info),
  0, ( struct sockaddr*  )& kpeer, & kpeerlen);    /从 socket
                                                   中接收消息
  ……
  }
return 0;
}
```

2.2　内核空间程序设计

(1)接收用户空间数据程序。

数据接收程序,用户进程所发送的实际数据保存在 sk指向的 sock中。

```
static void kernel_receive( struct sock *  sk,  int len)  /接收
用户空间数据,运行在软件中断环境。
{
  do
  {……
          while (( skb = skb_dequeue(& sk- > receive_
  queue) )!  = NULL)    //从队列取数据
          {……
            kfree_skb( skb);
          }
       } while( nlfd&& nlfd- > receive_queue. qlen);
}
```

(2)用户进程发送数据。

```
static int send_to_user( strcut packet_info*  info)  /* 发送数
据到用户空间* /
{……
  /开辟一个新的套接字缓存
  skb= alloc_skb( size, GFP_ATOMIC);
  old_tail= skb-> tail;
  /填写数据报相关信息
  nlh= NLMSG_PUT ( skb, 0, 0,  MP2_K_MSG, size- sizeof
  (*  nlh));
  packet= NLMSG_DATA( nlh);    /* 跳过消息首部,指向
                                          数据区* /
  memset ( packet, 0, sizeof( struct packet_info));    /* 初始
                                          化数据区* /
  /传输到用户空间的数据
  packet-> src= info-> src;
  packet-> dest= info-> dest;
  /* 计算经过字节对其后的数据实际长度* /
  nlh-> nlmsg_len= skb-> tail- old_tail;  (下转第 60页)
```

结构如图 3所示。

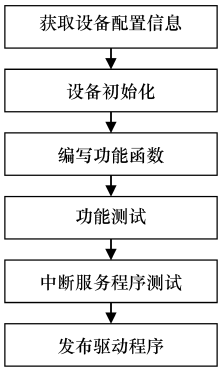

图 2　驱动程序开发流程图

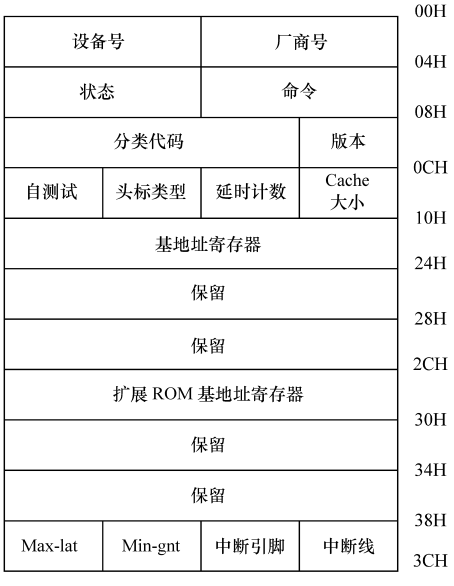

图 3　PCI设备配置头

PCI配置空间的访问机制为: 首先向 I/O地址为 0x0CF8处写入要访问寄存器的地址, 然后从 0x0CFC地址处读出 /写入数据。向 0x0CF8中写入的寄存器地址必须满足图 4所示的格式, 因此, 如果要获取一个 PCI设备的配置空间中某个寄存器的值, 必须知道设备的 Bus(总线)号、Device(设备)号、Fun(功能)号以及该寄存器在 PCI配置空间中的偏移地址。PCI协议规定, 一个系统中最多有 256条 PCI总线, 每条 PCI总线上最多可以挂接 32个 PCI设备, 每个设备最多有 8个功能。设备在上电时由 BIOS分配这 3个参数。

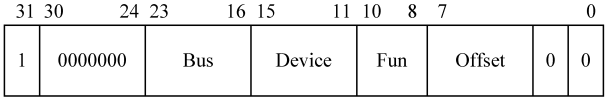

图 4　配置空间地址格式

3　VxWorks系统中的设备初始化

驱动程序对设备的初始化是指在系统中搜索指定的设备并按照设备配置空间中的要求为设备分配资源和填充设备信息。

从前面可以知道, 要确定系统的一个 PCI设备, 必须知道该设备所在的总线号、设备号和功能号。这 3个参数可以这样获得: 实际上, 所有 PCI设备的 256 B的配置数据结构都会依次出现在操作系统的一个数组中, 而设备的总线号、设备号和功能号就是该数组的索引值。从前文可知, PCI设备配置头的开始 2个字为设备的厂商号和设备标识, 这 2个字就标识了一种设备。所以只要对数组进行遍历, 并依次把每一个设备的厂商号和设备标识与要搜索设备的厂商号和设备标识进行比较, 就可以得到该设备所在的总线号、设备号和功能号。VxWorks系统中设备的搜索程序如下, 其中 pciConfigInWord函数实现以字(16位)的方式从指定的设备上读取 PCI配置寄存器的值, 函数的前 3个参数指明要访问的 PCI设备, 第 4个参数是要读取的寄存器在配置空间中的偏移地址, 最后一个参数用来保存读到的字。

(上接第 58页)

```
    NETLINK_CB(skb).dst_groups= 0;    /* 设置控制字段
                                      * /
    read_lock_bh(&user_proc.lock);
    //利用 socket向用户空间广播消息
    ret= netlink_unicast(nlfd, skb, user_proc.pid, MSG_DONT-
    WAIT);    //发送数据
    read_unlock_bh(&user_proc.lock);
    return ret;
    //若发送失败,则取消套接字缓存
    NLMsg_failure:
    if(skb) kfree_skb(skb);
    return -1;
}
```

3　结束语

在 Linux系统中, 操作系统的内核部分和应用程序被分别放置在不同的存储空间, 以利于实现对内存的保护。尽管内核具有比用户空间更高的保护级, 内核可以直接调用函数 execve来运行用户空间的应用程序, 但存在许多问题。比如, 内核不能一直等待应用程序运行完而不做其他的事情等。为了妥善解决内核与用户空间的通信问题, Linux系统使用了 netlink机制, 来实现内核与用户空间的双向通信。

在开发的基于 Linux的计算机联锁软件中, 内核经常处于软中断状态, 因而在设计中采用了 netlink通信机制, 来实现内核与用户空间的双向通信。经过在现场一年多的使用, 程序稳定可靠, 证明 netlink通信机制是解决内核与用户空间的双向通信的最佳方案。

参考文献:

[1]倪继利. Linux内核分析及编程[M]. 北京: 电子工业出版社, 2005.
[2]Gray Nutt. Kernel Projects for Linux[M]. Pearson Education, 2001.
[3]周明德. UNIX/Linux核心[M]. 北京: 清华大学出版社, 2004.
[4]夏卫民. 并行操作系统原理与技术[M]. 北京: 国防工业出版社, 2002.
[5]杜飞, 刘心松. netlink套接字在系统通信中的应用研究[J]. 微计算机信息, 2006, (9).

□