Linux 网络协议栈分析及协议添加的实现

唐 续 刘心松 杨 峰
(电子科技大学8010研究室 成都610054)

Network Stack Analyzing and Protocol Add Technology in Linux

TANG Xu LIU Xin-Song YANG Feng
(8010 Division, University of Electronic Science and Technology, Chengdu 610054)

Abstract In order to improve the performance of Linux network, new protocols should be implemented and added in original protocol stack. For this demand, this paper has analyzed Linux network stack architecture and implement technology, then presented a method that appended new protocols in the network stack of Linux. The processes of protocol register, protocol operation, protocol header implement, packets receiving, user interface are involved in this method.

Keywords Network performance, Linux network protocol stack, Protocol adding

1 引言

随着 Linux 操作系统功能的日趋完善,其应用越来越广泛,其中包括一些使用 Linux 平台的专用系统,如分布式并行服务器、分布式计算环境[1,2]等。这类系统的性能在很大程度上取决于通信的速度,因此对通信协议的传输效率要求很高。而传统的 TCP/IP[3,4]协议由于其额外的软件开销通常不能满足这类特殊要求[5,6],因此要求开发者必须实现一些高效的新协议并添加到原有的通信协议栈中。

在 Linux 中添加新的网络协议,需要涉及 ISO 七层模型中的数据链路层、网络层和传输层,过程相当复杂。因此,本文首先对 Linux 内核中网络协议栈结构和相应的实现技术进行分析,然后详细描述在 Linux 网络协议栈各层添加新协议的实现方法,内容包括协议注册、协议相关操作、协议头实现、数据包接收、用户调用接口等各个方面。其目的是为改进 Linux 网络系统提供一种有效的途径。

2 Linux 网络系统分析

Linux 网络系统具有良好的开放性。本节详细描述其系统结构及实现技术,以说明在其中扩展新协议的可能性。

2.1 Linux 网络系统结构

Linux 网络子系统使用数据抽象技术,其支持大量的网络设备和网络协议。图1描述了网络子系统的结构,每个可能的网络设备都对应一个设备驱动程序模块;独立于设备的接口模块为所有网络设备提供了一个统一的视图,在子系统的高级别上无需关心所使用硬件的特定知识;网络协议模块负责实现每个可能的网络传输协议;独立于协议的接口模块提供一个独立于网络设备和网络协议的接口,其它内核子系统通过该接口模块来访问网络,无需依赖特定的协议或硬件。

可以看出,这种结构具有很强的可扩展性。一方面,新网络协议的开发只需面向抽象接口,就可专注于本协议专用操作,避免了与内核其它部分进行很多交互。另一方面,用户进程和其它内核子系统在访问网络时也只需了解协议的接口。这极大地方便了 Linux 中新网络协议的开发与测试。

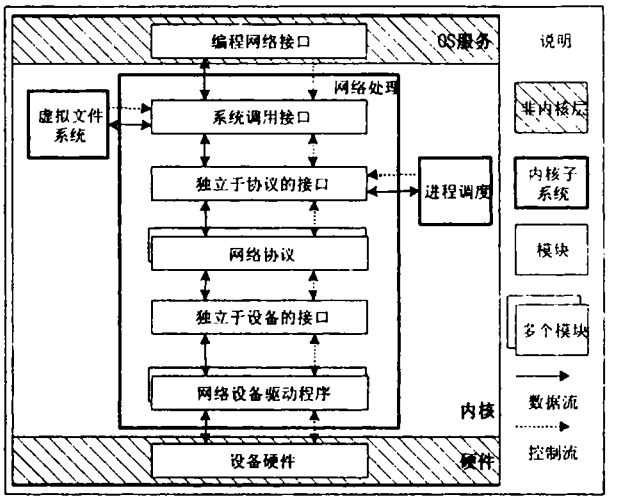

图1 LINUX 网络子系统层次结构图

2.2 Linux 协议栈实现技术

Linux 网络子系统支持 TCP/IP、X25、IPX 等通用的协议栈,并以高度抽象的接口屏蔽各协议栈模块的实现和操作细节。这些抽象接口体现为网络处理流程(如图2)中一些关键数据结构的实现,下面对照图1中的模块进行具体描述。

·socket 向上以标准的 BSD socket 系统调用接口为用户提供网络服务,并构成用户网络程序的主体。socket 也是网络与文件系统和进程管理的接口,Linux 中,它属于文件系统的一部分,网络通信可以被看作是对文件的读写和控制。

在一个 socket 结构上操作的确切意义依赖于其下层的地址族(后文也称为协议族)。Linux 支持多种常见的地址族,每种地址族又可以支持一种或多种套接字服务类型[7]。

建立在内核的各地址族在初始化时向 BSD socket 接口注册。将本族提供的服务入口 xxx_family_ops 添加到内核数

唐 续 硕士研究生,研究方向为三网合一、分布式并行计算机等。刘心松 教授,博士生导师,研究方向为计算机网络、CATV 和电话宽带综合网络、分布式计算机、并行处理、计算机网络与通信。杨 峰 博士研究生,研究方向为分布式并行数据存储和宽带综合网络。

据结构 net_families []中。稍后，应用程序创建和使用 BSD socket 时，调用通用创建例程：

socket(地址族，套接字类型，协议类型号)，将引发调用地址族参数相关的 socket 创建例程。此例程中再根据套接字类型参数，确定此次 socket 通信使用的操作函数集 xxx_xxx_ops。数据结构 proto_ops 抽象了 socket 所提供的通用网络服务，而具体的数据变量 xxx_xxx_ops 则定义了当前协议族能支持的服务操作，它从最高层次规划了一类此协议族提供的网络服务。如 INET 族定义的 inet_stream_ops 和 inet_dgram_ops，分别规划了流类型和数据报类型两类网络服务。

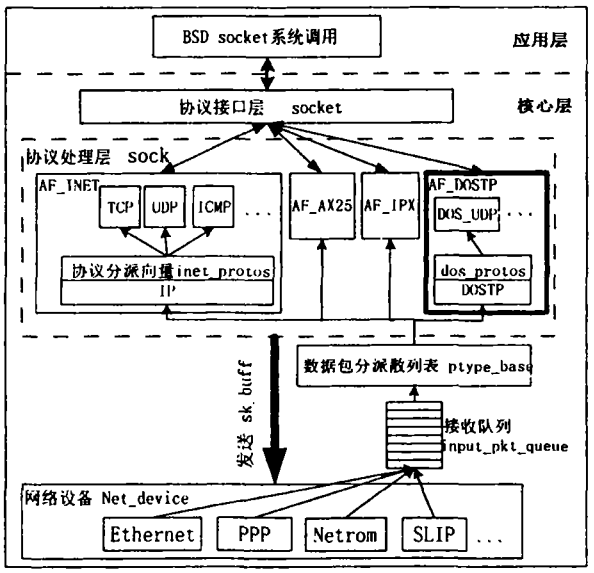

图2 LINUX 网络处理流程图及协议族 DOSTP

·sock 处于 socket 的下层，抽象了独立于协议的接口。相同或不同协议的通信实例在内核中并发执行，各对应一个 sock 实例。特定 sock 实例的操作代表了通过 BSD socket 接口调用发起的具体的协议处理。

Sock 数据结构中定义了协议通用处理的数据结构成员，如链表、等待队列、定时器、锁和引用记数等；为了也能表征各种不同的协议，sock 结构中还提供回调(callback)函数形式的操作集和联合(union)形式的各协议专用的数据结构项，其以标准的操作界面提供对协议相关处理的灵活支持。

·sk_buff 用于管理不同协议层之间的数据传递，对 Linux 网络系统性能至关重要。sk_buff 组织成双向链表形式，每个 sk_buff 包括一些控制方法和一块数据缓冲区。控制方法按功能分为两种类型：一种是控制整个 buffer 链的方法；另一种是控制数据缓冲区的方法。对 buffer 链表的操作主要是删除链表头的元素和添加到链表尾。对数据缓冲区的操作主要是增删协议头和管理缓冲区内存块。sk_buff 的控制方法都很短小以尽量减少系统负荷。

与 sock 结构定义的策略相同，sk_buff 结构中以联合的形式实现协议层头部定义，灵活支持不同层次的协议操作。该结构也设置协议相关控制块 cb[48]，但不同于 sock 中的控制块 protinfo 针对于整次通信，cb 用于自定义在通信中每一数据包都需要的私有数据。如 INET 族中定义它实现 IP 协议的选项操作[3]和 TCP 协议[4]的每数据包控制信息封装等。

数据发送时，sk_buff 在用户数据通过 socket 接口向下进入协议处理层时分配，处理各协议层对数据的封装后，通过标准调用 dev_queue_xmit 进入网络驱动程序，并在网络设备完成此数据包发送后释放；数据接收时，sk_buff 由网络驱动程序的接收例程产生，封装接收到的数据包并送到输入队列 input_pkt_queue[8]，通过系统软中断机制——网络的底半层 bottom half[7]进入协议处理层，最后把数据递交到 socket 接口层。

·net_device 网络设备驱动程序在内核启动网络初始化时向 Linux 登记其控制的设备。每个网络设备各用一 net_device 数据结构表示。如图2，net_device 结构抽象了不同的网络设备，为协议处理层提供了独立于设备的接口，为 sk_buff 中设备相关的操作提供了统一界面。net_device 数据结构包括此设备的信息和其所支持的网络协议使用此设备服务的函数指针。其也是网络设备驱动程序的核心数据结构。对网络设备驱动程序设计的讨论可参阅文[9]。

·ptype_base 网络驱动程序接收的数据包从统一入口——输入队列 input_pkt_queue 进入协议处理层，要求被正确地递交到相应的协议，这一功能由 ptype_base 完成。协议处理层底部的每一个协议在初始化时向内核登记：通过调用标准例程 dev_add_pack 在 ptype_all 链表或 ptype_base 散列表中增加一个 packet_type 结构实例。packet_type 包括协议标识符，一个网络设备的指针，一个协议的接收处理例程的指针和一个指向此链表或散列表中下一 packet_type 数据项的指针。ptype_all 链表中各项的协议标识符都是 ETH_P_ALL，特别用于探测(snoop)从任意网络设备上接收到的所有的数据报文，通常不使用。ptype_base 使用协议标识符为散列。网络的底半层 bottom half 把到来的 sk_buff 按其协议标识符和表中对应散列项内的 packet_type 条目逐条匹配。协议可能会匹配一个或多个条目，此时 sk_buff 可以被克隆，被传递到所有匹配协议的处理例程。

3 新协议添加

对 Linux 网络子系统的改造，一方面体现为在不修改现有网络协议的前提下，截获感兴趣的网络协议包，并可能插入自定义处理，这是常用的网络入侵及相对的网络防火墙的实现手段之一[10]；另一方面体现为在特殊应用环境实现专用的新型网络协议，如嵌入式环境使用的轻型协议实现、分布式环境使用的高速协议实现等。由于截获网络协议包实现相对简单，其底层操作主要通过向 ptype_base 或 ptype_all 添加自定义接收函数来实现，因此这里不作详述。下面主要阐述新协议添加的实现。

3.1 协议添加实现

从上节对 Linux 网络系统的分析可以看出，合理利用网络系统结构中的抽象接口和数据结构定义中提供的灵活的自定义结构项，能在不同协议层修改或增加网络协议。本节以向 Linux 添加自行开发的专用通信协议栈 DOSTP 为例，较全面地反映在不同通信协议层所做的操作。

此协议栈实现为单独一类可扩展的协议族 AF_DOSTP，其结构如图2中所示：协议栈分两层，上层 DOS_UDP 等协议提供了不同套接字类型的高层服务，下层 DOSTP 主要为上层各协议提供通用支持。为了使本协议族功能融入 Linux 网络系统中，需要进行以下各项工作。

3.1.1 *协议族的注册* 此项工作的目的在于向 Linux 网络系统添加新协议族提供的顶层服务。其包括以下操作：取地址族号 AF_DOSTP，为系统定义未占用的地址族编号；声明 proto_ops 结构的变量 dostp_proto_ops 以规划此协议族的服务操作集；定义本协议族的服务入口 dostp_family_ops；在协议初始化时调用以下代码：

```
struct net_proto_family dostp_family_ops = {
```

```
PF_DOSTP, //本协议族号,等于 AF_DOSTP
dostp_create //本协议族相关的 socket 创建例程
}
sock_register(&dostp_family_ops);//地址族注册函数
```

3.1.2 协议相关操作的实现 这部分工作实现具体协议功能的添加,主要体现在对 sock 结构的操作,下面以 dos_udp 协议为例说明。首先需要定义此协议在 sock 结构上的具体操作集 struct proto dosudp_prot,实现协议族服务操作集到具体功能函数集的映射;然后是针对此协议特殊处理的数据结构的实现:在充分利用 sock 结构原有数据项的前提下,以协议相关控制块 dostp_cb 结构实现。此结构被加入到抽象数据联合 protoinfo 中(sock 结构定义的修改代码如下),并在 sock 结构生成时以指针形式相互挂接。

```
struct sock { ……
   struct proto *prot; //用于指向具体协议操作集 dosudp_prot
   union {
   void *destruct_hook;
   struct unix_opt af_unix;
   ……
# ifdef CONFIG_DOSTP_MODULE
  strcut dostp_cb *dos_cb;
#endif
} protinfo; //协议相关控制块
   ……
}
```

至此,本协议功能已处于协议栈向下调用流程中。最后是函数集具体算法的实现,不在此赘述。

3.1.3 协议头的实现 此项工作在于管理通信数据缓存,处理各层协议间的数据传递。本例中实现为由 sk_buff 结构管理通信数据,以自定义协议头支持完成各协议层操作。协议头定义集成于 sk_buff 结构中的相关协议头联合:

```
struct sk_buff { ……
  union{ struct tcphdr *th;
         struct udphdr *uh;
         ……
   # ifdef CONFIG_DOSTP_MODULE
          struct dos_udphdr *dos_uh; //DOS_UDP 协议头
   # endif
          unsigned char *raw;
  } h; //传输层协议头
  union{ struct iphdr *iph;
         struct ipxhdr *ipxh;
         ……
   #ifdef CONFIG_DOSTP_MODULE
         struct dostphdr *dostph; // DOSTP 协议头
   # endif
          unsigned char *raw;
  } nh; //网络层协议头
    ……
}
```

这一实现利用 sk_buff 的高效管理手段及其与网络驱动程序的方便接口,能较大程度上减少通信开销;同时,因避免了直接操纵内存块,也降低了协议实现难度。

3.1.4 协议数据包的接收 目的在于实现此协议栈中数据包的接收流程。首先要求底层协议的数据包类型及其接收函数注册:在初始化时调用如下代码,将 dostp_packet_type 添加到数据包分派散列表 ptype_base 中。

```
static struct packet_type dostp_packet_type ={
   ETH_P_ DOSTP, /*本协议自定义包类型标识符*/
   0, /*设备*/
   dostp_recv, /*本通信协议平台的接收函数*/
   NULL, /*私有数据*/
   NULL, /*指向类型链表的下一个*/
   };
dev_add_pack(&dostp_packet_type); //协议包类型注册
```

注:Linux 实现源码[6]中要求自定义包类型标识号>=1536

然后是高层协议如 DOS_UDP 在协议族内的注册。和 INET 协议族中设置 inet_protos 支持 TCP、UDP、ICMP 等协议类型(如图2)类似,DOSTP 族以协议链表 dos_protos 组织高层协议,当中保存在初始化时加入的协议相关接收函数。

接收流程如下:当此协议族的数据包从网络驱动程序进入协议处理层时,匹配包头与 ptype_base 中注册的类型标识符,从而被 DOSTP 接收,并在其处理函数中比较此包网络层协议头 dostph 中的协议类型号,然后根据匹配值将数据包递交给 DOS_UDP 等高层协议所注册的接收函数。限于篇幅,具体实现不再详细描述。

3.1.5 协议的用户调用接口 新协议作为内核的系统功能,一般以新系统调用的方式向用户提供[11]。而提供简单易用的用户编程接口是软件可移植性的要求,也是新协议能否被用户接受的主要因素之一。因此本协议完全采用 BSD 套接字接口来进行封装,通过头文件 dostp.h 向用户提供协议族标识号 AF_DOSTP 和协议地址结构 sockaddr_dostp 等套接字信息。编程用户只需简单替换地址族和协议地址结构,修改调用参数,就能将符合套接字调用标准的程序,方便地移植到当前的通信平台中。

3.2 协议的编译和加载

Linux 是单内核操作系统,直接增删系统功能很不方便。其提供一种内核模块机制[7],为新功能代码加入系统内核提供了捷径。

本协议栈实现为一个内核模块。由于对 Linux 内核源代码中的一些数据结构作了改动,为了保持原内核的清洁,对修改的部分都加以条件编译指令包括,并在模块编译时加上-DCONFIG_DOSTP_MODULE 参数使修改代码生效。编译成功的新协议模块一般在系统网络初始化时,判断指定的网络设备已正常启动后,由控制台命令 insmod(或写成执行脚本)加载,从而把新功能融入到网络系统中。当不再使用此协议栈时,该模块也允许用 rmmod 命令动态卸载,以精简内核运行规模。

结束语 本文对 Linux 网络协议栈结构和实现技术进行了详细分析,并介绍了协议注册、协议相关操作、协议头实现、数据包接收、用户调用接口等协议添加实现过程。文中所给代码都已调试通过,并在自行开发的分布式并行服务器系统中正常运行。

参 考 文 献

1 Zhang W, et al. Linux virtual server project. http://www.LinuxVirtualServer.org/,1998-now
2 Mandek R. Design and Implementation of PVM Version3. Masterthesis University of Tennessee,1994
3 Postel J. Internet Protocol. RFC791. Sep-01-1981
4 Postel J. Transmission Control Protocol. RFC793. Sep-01-1981
5 潘建平,等. 区域网络 TCP/IP 协议栈的性能分析. 计算机学报,1999,22(5):513~518
6 龙翔,等. SNOW 系统通信协议——Homer. 计算机科学,1999,27(1)
7 Rusling D A,et al. Linux Programming White Papers. [M]北京:机械工业出版社,2000
8 LINUX Kernel v2.4.2-2 Source Code
9 Rubini A. Linux Device Drivers. [M]北京:中国电力出版社,2000.361~403
10 王宏健,邵佩英,张籍. 基于 Linux 内核防火墙 Netfilter 的安全应用的设计方法. 小型微型计算机系统,2001,22(12):1516~1518
11 王姝阳,庞丽萍,李胜利. LINUX 系统调用实现技术. 小型微型计算机系统,1999,20(12):924~926