第3章　网络编程

本章主要是设计和实现数据链路层、网络层、传输层和应用层的常用协议，加深理解网络协议的工作原理，掌握网络编程的基本流程和基本方法，培养编程开发能力。

本章的主要内容包括以太网 ARP 请求发送程序、太网 ARP 响应分析程序、ping 程序、多播客户/服务器程序、UDP 报文发送程序、TCP SYN 报文发送程序、UDP 客户/服务器程序、TCP 客户/服务器程序、服务器接口监测程序、邮件客户端程序和聊天客户/服务器程序等。其中前 8 个程序采用 C 语言实现，后 3 个程序采用 Java 语言实现。

3.1　以太网 ARP 请求发送程序

3.1.1　课程设计目的

ARP 协议为 IP 地址到对应的硬件地址之间提供动态映射。本课程设计的主要目的是通过构造与发送以太网 ARP 请求，掌握 ARP 协议的工作原理，理解以太网帧和 ARP 请求的格式。

3.1.2　课程设计内容

本课程设计要求设计与实现 Linux 下以太网 ARP 请求发送程序，具体内容如下：

(1) 该程序能够实现以下功能：能够获取指定接口的 MAC 地址、IP 地址和接口索引；构造以太网帧头部和 ARP 请求，并向本地网络广播这个 ARP 请求；

(2) 设计和实现以太网 ARP 请求发送程序；

(3) 能够用 Wireshark 分析发送的 ARP 请求。

3.1.3　相关知识

本课程设计需要的 ARP 协议知识，请参考 2.3.3 节。

本课程设计需要的 Linux 网络编程的知识，请参考附录 B。

3.1.4　课程设计分析

1. 详细设计

以太网 ARP 请求发送程序的流程图如图 3-1 所示。

构造 ARP 请求的流程图如图 3-2 所示。

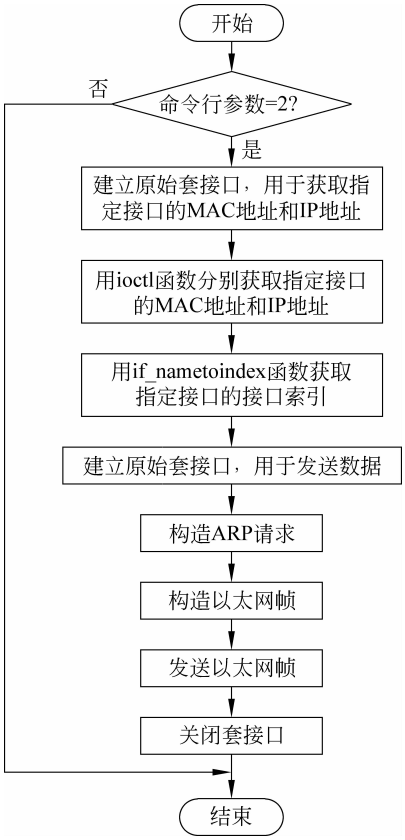

图 3-1　以太网 ARP 请求发送程序的流程图

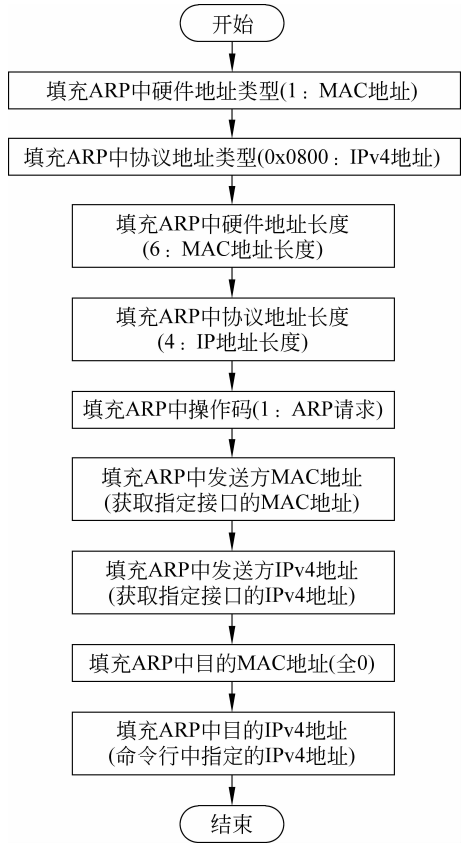

图 3-2　构造 ARP 请求流程图

构造以太网帧的流程图如图 3-3 所示。

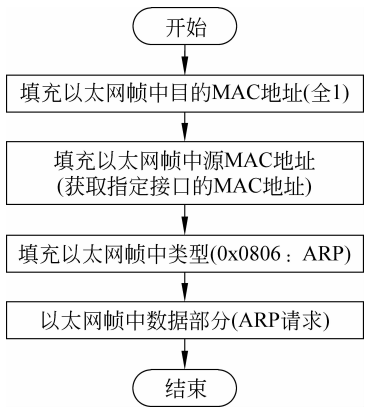

图 3-3　构造以太网帧的流程图

2. 核心代码

(1) ARP 结构定义。ARP 结构(ether_arp 结构)在<netinet/if_ether.h>中定义,其中 arphdr 在<linux/if_arp.h>中定义。

ether_arp 结构:

```
struct  ether_arp {
  struct arphdr ea_hdr;                    //见"struct arphdr"
  u_int8_t arp_sha[ETH_ALEN];              //发送方 MAC 地址
  u_int8_t arp_spa[4];                     //发送方 IP 地址
  u_int8_t arp_tha[ETH_ALEN];              //目的 MAC 地址
  u_int8_t arp_tpa[4];                     //目的 IP 地址
};
#define arp_hrd ea_hdr.ar_hrd
#define arp_pro ea_hdr.ar_pro
#define arp_hln ea_hdr.ar_hln
#define arp_pln ea_hdr.ar_pln
#define arp_op ea_hdr.ar_op
```

arphdr 结构:

```
struct arphdr
{
  __be16        ar_hrd;                    //硬件类型。值为 1 表示以太网地址
  __be16        ar_pro;                    //协议类型。值为 0x0800 表示 IP 地址
  unsigned char ar_hln;                    //硬件地址长度。值为 6 表示 MAC 地址长度
  unsigned char ar_pln;                    //协议地址长度。值为 4IP 地址长度
  __be16        ar_op;            //操作码。值为 1 表示 ARP 请求,值为 2 表示 ARP 响应

#if 0
    //
      * Ethernet looks like this : This bit is variable sized however...

    unsigned char        ar_sha[ETH_ALEN];  //sender hardware address
    unsigned char        ar_sip[4];         //sender IP address
    unsigned char        ar_tha[ETH_ALEN];  //target hardware address
    unsigned char        ar_tip[4];         //target IP address
#endif

};
```

(2) 构造以太网头部和 ARP 请求,发送 ARP 请求。

```
//说明的变量
char * if_name, * tgt_ip, * src_ip;
uint8_t * src_mac, * tgt_mac, * ether_frame;
```

```
struct ether_arp arp_hdr;                    //ARP 头部
struct sockaddr_ll phy_layer_addr;
struct ifreq ifr;
…;
//建立一个原始套接口,用于获取指定接口的 MAC 地址和 IPv4 地址
sd=socket(AF_INET, SOCK_RAW, IPPROTO_RAW);
//用 ioctl()函数获取名为 if_name 接口的 MAC 地址
memset(&ifr, 0, sizeof(ifr));
snprintf(ifr.ifr_name, sizeof(ifr.ifr_name), "%s", if_name);
ioctl(sd, SIOCGIFHWADDR, &ifr);
//用 ioctl()函数获取名为 if_name 接口的 IPv4 地址
ioctl(sd, SIOCGIFADDR, &ifr);
…;
//建立一个原始套接口,用于发送以太网帧
sd=socket(PF_PACKET, SOCK_RAW, htons(ETH_P_ALL));
//以下是构造 ARP
//填 ARP 中硬件地址类型。它的值为 1 表示以太网
arp_hdr.arp_hrd=htons(1);
//ARP 中协议地址类型。它的值为 0x0800 表示 IP
arp_hdr.arp_pro=htons(ETH_P_IP);
//ARP 中硬件地址长度。它的值为 6 表示 MAC 地址长度
arp_hdr.arp_hln=6;
//ARP 中的协议地址长度。它的值为 4 表示 IPv4 地址
arp_hdr.arp_pln=4;
//ARP 中操作码。它的值为 1 表示 ARP 请求
arp_hdr.arp_op=htons(ARPOP_REQUEST);
//ARP 中发送方 MAC 地址
memcpy(&arp_hdr.arp_sha, src_mac, 6 * sizeof(uint8_t));
//ARP 中发送方 IPv4 地址
status=inet_pton(AF_INET, src_ip, &arp_hdr.arp_spa);
//ARP 中目的 MAC 地址。将它设为 0
memset(&arp_hdr.arp_tha, 0, 6 * sizeof(uint8_t));
//ARP 中目的 IPv4 地址
status=inet_pton(AF_INET, tgt_ip, &arp_hdr.arp_tpa);
//以下是构造以太网帧
//以太网帧中目的 MAC 地址
memcpy(ether_frame, tgt_mac, 6 * sizeof(uint8_t));
//以太网帧中的源 MAC 地址
memcpy(ether_frame+6, src_mac, 6 * sizeof(uint8_t));
//以太网帧中类型。对于 ARP 请求或响应来说,该字段的值为 0x0806
ether_frame[12]=ETH_P_ARP / 256;
ether_frame[13]=ETH_P_ARP %256;
//向套接口 sd 中发送构造的以太网帧
sendto(sd, ether_frame, frame_length, 0,(struct sockaddr *)&phy_layer_addr,
```

```
sizeof(phy_layer_addr));
…;
```

3.1.5 进一步的扩展

通过对目标主机或网关发送伪造的 ARP 请求报文，欺骗目标的 ARP 高速缓存，使目标主机的数据发送到错误的地方，达到使对方无法正常上网的效果。

(1) 定时向默认网关发送伪造的 ARP 请求，伪装成目标主机，修改网关的 ARP 高速缓存，使网关发送到目标主机的信息都发送到本机，实现目标主机的断网效果。向默认网关发送伪造的 ARP 请求的主要字段见表 3-1。

表 3-1 向默认网关发送伪造的 ARP 请求的主要字段

ARP 请求的字段	填 入 内 容	ARP 请求的字段	填 入 内 容
目的以太网地址	网关 MAC 地址	发送方 IP 地址	目的主机 IP 地址
源以太网地址	本机 MAC 地址	目的 MAC 地址	网关 MAC 地址
操作码	1	目的 IP 地址	网关 IP 地址
发送方 MAC 地址	本机 MAC 地址		

(2) 定时向目标主机发送伪造的 ARP 请求，伪装成网关，修改目标主机的 ARP 高速缓存，使目标主机发送到网关的信息都发送到本机，实现目标主机的断网效果。向目标主机发送伪造的 ARP 请求的主要字段见表 3-2。

表 3-2 向目标主机发送伪造的 ARP 请求的主要字段

ARP 请求的字段	填 入 内 容	ARP 请求的字段	填 入 内 容
目的以太网地址	目标主机 MAC 地址	发送方 IP 地址	网关 IP 地址
源以太网地址	本机 MAC 地址	目的 MAC 地址	目标主机 MAC 地址
操作值	1	目的 IP 地址	目标主机 IP 地址
发送方 MAC 地址	本机 MAC 地址		

3.2 以太网 ARP 响应分析程序

3.2.1 课程设计目的

本课程设计的主要目的是通过接收与解析以太网 ARP 响应，掌握 ARP 协议的工作原理，理解以太网帧和 ARP 请求的格式。

3.2.2 课程设计内容

本课程设计要求设计与实现 Linux 下以太网 ARP 响应分析程序，具体内容如下：

(1) 该程序能够实现以下功能：从网络中读取数据包，若 ARP 响应，则解析这个响应的各字段并输出；

(2) 设计和实现以太网 ARP 响应分析程序；

(3) 能够用 Wireshark 分析收到的 ARP 响应。

3.2.3 相关知识

本课程设计需要的 ARP 协议知识，请参考 2.3 节。

本课程设计需要的 Linux 网络编程知识，请参考附录 B。

3.2.4 课程设计分析

1. 详细设计

以太网 ARP 响应分析程序的流程图如图 3-4 所示。

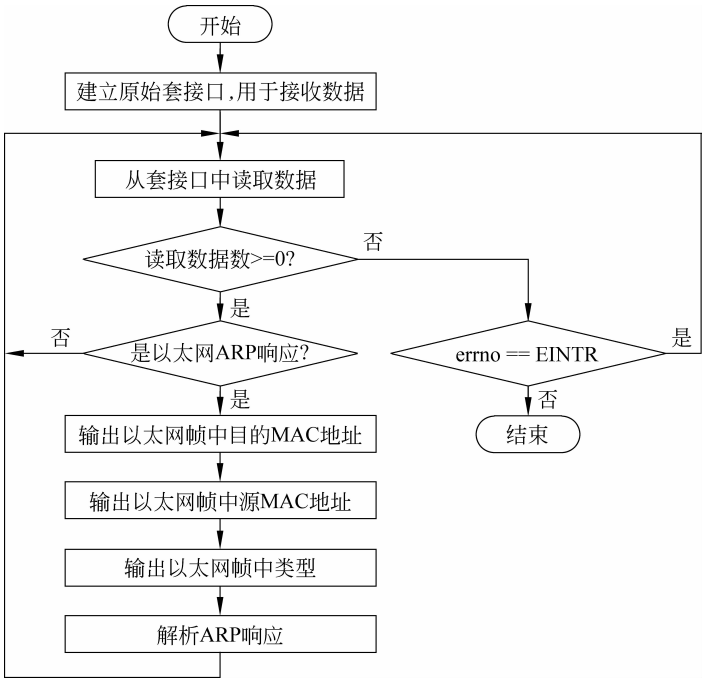

图 3-4 以太网 ARP 响应分析程序的流程图

解析 ARP 响应的流程图如图 3-5 所示。

图 3-5　解析 ARP 响应流程图

2. 核心代码

(1) ARP 结构定义。请参考太网 ARP 请求发送程序。

(2) 以太网 ARP 接收与解析。

```
//说明变量
uint8_t *ether_frame;
struct ether_arp *arp_hdr;
…;
//建立一个原始套接口,用于接收数据
sd=socket(PF_PACKET, SOCK_RAW, htons(ETH_P_ALL));
…;
//从套接口 sd 读取以太网帧
if((status=recv(sd, ether_frame, IP_MAXPACKET, 0))<0){
  if(errno==EINTR){
    memset(ether_frame, 0, IP_MAXPACKET * sizeof(uint8_t));
    continue;
  } else { …;}
}
//如果它是 ARP 响应,则输出这个 ARP 以太网帧
```

```
if(((((ether_frame[12])<<8)+ether_frame[13])==ETH_P_ARP)&&
    (ntohs(arp_hdr->arp_op)==ARPOP_REPLY)){
  //输出以太网帧中的头部
  printf("%02x:%02x:%02x:%02x:%02x:%02x\n", ether_frame[0], ether_frame[1],
  ether_frame[2], ether_frame[3], ether_frame[4], ether_frame[5]);
  printf("%02x:%02x:%02x:%02x:%02x:%02x\n", ether_frame[6], ether_frame[7],
  ether_frame[8], ether_frame[9], ether_frame[10], ether_frame[11]);
  printf("Ethernet type code(0x0806 for ARP): 0x%02x%02x\n", ether_frame[12],
  ether_frame[13]);
  //输出以太网帧中的数据(ARP响应头部)
  printf("\nEthernet data(ARP reply header):\n");
  printf("Hardware address type(1 for ethernet): %u\n", ntohs(arp_hdr->arp_
  hrd));
  printf("Protocol address type(0x0800 for IPv4 addresses): 0x%04x\n", ntohs(arp
  _hdr->arp_pro));
  printf("Hardware(MAC)address length: %u bytes\n", arp_hdr->arp_hln);
  …;
}
```

3.2.5 进一步的扩展

通过对目标主机或网关发送伪造的ARP响应,欺骗目标的ARP高速缓存,使目标主机的数据发送到错误的地方,达到使对方无法正常上网的效果。

(1) 定时向默认网关发送伪造的ARP响应,伪装成目标主机,修改网关的MAC高速缓存,使网关发送到目标主机的信息都发送到本机,实现目标主机的断网效果。向默认网关发送伪造的ARP响应的主要字段见表3-3。

表3-3 向默认网关发送伪造的ARP响应的主要字段

ARP响应字段	填入内容	ARP响应字段	填入内容
目的以太网地址	网关MAC地址	发送方IP地址	目的主机IP地址
源以太网地址	本机MAC地址	目的MAC地址	网关MAC地址
操作值	2(响应)	目的IP地址	网关IP地址
发送方MAC地址	本机MAC地址		

(2) 定时向目标主机发送伪造的ARP响应报文,伪装成网关,修改目标主机的MAC表,使目标主机发送到网关的信息都发送到本机,实现目标主机的断网效果。向目标主机发送伪造的ARP响应的主要字段见表3-4。

表 3-4 向目标主机发送伪造的 ARP 响应的主要字段

ARP 响应字段	填 入 内 容	ARP 响应字段	填 入 内 容
目的以太网地址	目标主机 MAC 地址	发送方 IP 地址	网关 IP 地址
源以太网地址	本机 MAC 地址	目的 MAC 地址	目标主机 MAC 地址
操作值	2(响应)	目的 IP 地址	目标主机 IP 地址
发送方 MAC 地址	本机 MAC 地址		

3.3 ping 程序

3.3.1 课程设计目的

本课程设计主要目的是通过构造和发送 ICMP 回射请求报文,接收和解析 ICMP 响应报文,使学生掌握 ICMP 协议的工作原理,理解 ICMP 回射报文格式。

3.3.2 课程设计内容

本课程设计要求设计与实现 Linux 下 ping 程序,具体内容如下:

(1) 该程序能够实现以下功能:能够构造 ICMP 回射请求报文,通过信号量定时发送回射请求报文,接收和解析 ICMP 回射响应报文;

(2) 设计和实现 ping 程序;

(3) 能够用 Wireshark 分析的 ICMP 报文。

3.3.3 相关知识

ping 程序发送一份 ICMP 回射请求(echo request)报文给指定主机,并等待返回 ICMP 回射响应(echo reply);该程序会按时间和反应成功的次数,估计往返时延和丢包率。

ICMP 回射报文的格式如图 3-6 所示。

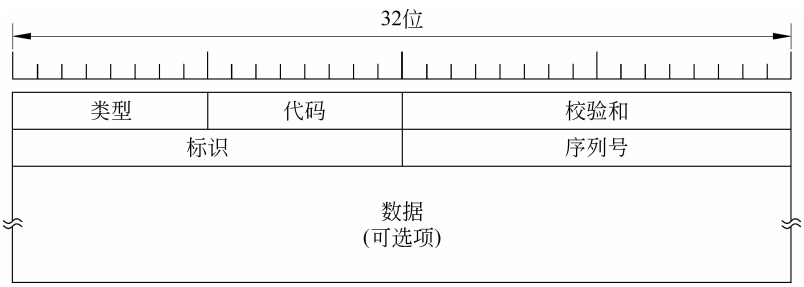

图 3-6 ICMP 回射报文的格式

类型的值为 8 表示回射请求，类型的值为 1 表示回射响应。
代码为 0。
校验和为从 TYPE 开始到 IP 包结束的校验和。
如果代码的值为 0，标识用来匹配回射请求和回射响应报文。
ICMP 回射报文封装成一个 IP 数据包的格式如图 3-7 所示。

图 3-7　ICMP 回射报文封装

本课程设计需要的 Linux 网络编程知识，请参考附录 B。

3.3.4　课程设计分析

1. 详细设计

main 函数如图 3-8 所示。
readloop()函数流程图如图 3-9 所示。

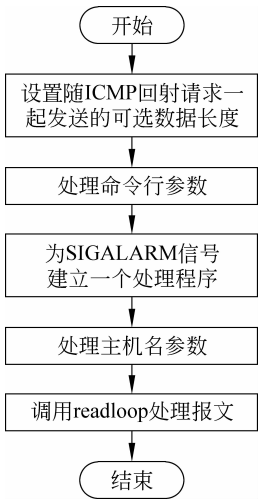

图 3-8　main 函数流程图

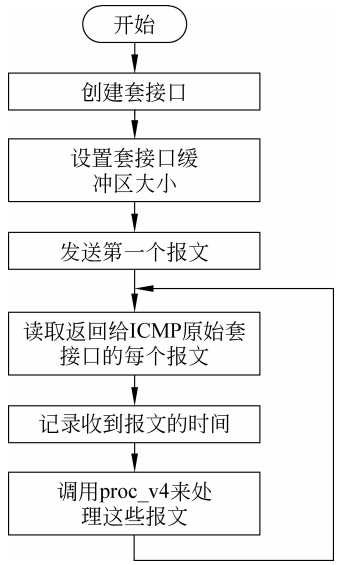

图 3-9　readloop()函数流程图

proc_v4()函数流程图如图 3-10 所示。
send_v4()函数流程图如图 3-11 所示。

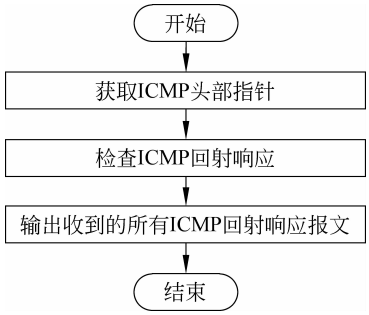

图 3-10　proc_v4()函数流程图

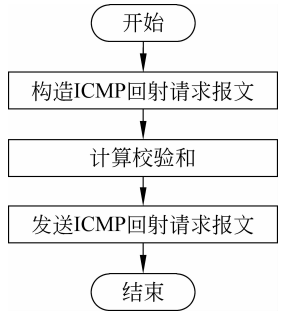

图 3-11　send_v4()函数流程图

2. 核心代码

(1) IPv4 头结构在< netinet/ip. h >中定义。IPv4 头结构如下所示：

```
struct ip
{
#if __BYTE_ORDER==__LITTLE_ENDIAN       //Intel x86 机器采用 little-endian 字节序
    unsigned int ip_hl:4;               //头部长度
    unsigned int ip_v:4;                //版本
#endif
#if __BYTE_ORDER==__BIG_ENDIAN
    unsigned int ip_v:4;
    unsigned int ip_hl:4;
#endif
    u_int8_t ip_tos;                    //tos
    u_short ip_len;                     //总长度
    u_short ip_id;                      //标识
    u_short ip_off;                     //标志和分段偏移
#define    IP_RF 0x8000                 //保留分段标志位
#define    IP_DF 0x4000                 //不允许分段标志位
#define    IP_MF 0x2000                 //更多分段标志位
#define    IP_OFFMASK 0x1fff            //分段位移的掩码
    u_int8_t ip_ttl;                    //TTL
    u_int8_t ip_p;                      //协议
    u_short ip_sum;                     //检验和
    struct in_addr ip_src, ip_dst;      //源地址和目的地址
};
```

ICMP 头结构在<netinet/ip_imcp. h>中定义。ICMP 头结构如下所示：

```
struct icmp
{
  u_int8_t  icmp_type;                  //类型
  u_int8_t  icmp_code;                  //代码
```

```
  u_int16_t icmp_cksum;                                    //校验和
  union
  {
    u_char ih_pptr;
    struct in_addr ih_gwaddr;
    struct ih_idseq
    {
      u_int16_t icd_id;
      u_int16_t icd_seq;
    } ih_idseq;
    u_int32_t ih_void;

    //ICMP_UNREACH_NEEDFRAG --Path MTU Discovery(RFC1191)
    struct ih_pmtu
    {
      u_int16_t ipm_void;
      u_int16_t ipm_nextmtu;
    } ih_pmtu;

    struct ih_rtradv
    {
      u_int8_t irt_num_addrs;
      u_int8_t irt_wpa;
      u_int16_t irt_lifetime;
    } ih_rtradv;
  } icmp_hun;
#define icmp_pptr    icmp_hun.ih_pptr
#define icmp_gwaddr  icmp_hun.ih_gwaddr
#define icmp_id      icmp_hun.ih_idseq.icd_id          //标识
#define icmp_seq     icmp_hun.ih_idseq.icd_seq         //序列号
#define icmp_void    icmp_hun.ih_void
#define icmp_pmvoid  icmp_hun.ih_pmtu.ipm_void
#define icmp_nextmtu      icmp_hun.ih_pmtu.ipm_nextmtu
#define icmp_num_addrs    icmp_hun.ih_rtradv.irt_num_addrs
#define icmp_wpa    icmp_hun.ih_rtradv.irt_wpa
#define icmp_lifetime     icmp_hun.ih_rtradv.irt_lifetime
  union
  {
    struct
    {
      u_int32_t its_otime;
      u_int32_t its_rtime;
      u_int32_t its_ttime;
    } id_ts;
```

```
    struct
    {
      struct ip idi_ip;
      //options and then 64 bits of data
    } id_ip;
    struct icmp_ra_addr id_radv;
    u_int32_t   id_mask;
    u_int8_t    id_data[1];
  } icmp_dun;
#define icmp_otime  icmp_dun.id_ts.its_otime
#define icmp_rtime  icmp_dun.id_ts.its_rtime
#define icmp_ttime  icmp_dun.id_ts.its_ttime
#define icmp_ip     icmp_dun.id_ip.idi_ip
#define icmp_radv   icmp_dun.id_radv
#define icmp_mask   icmp_dun.id_mask
#define icmp_data   icmp_dun.id_data          //ICMP 数据
};
```

(2) 函数的部分实现。

readloop()函数的部分实现：

```
//建立一个 ICMP 原始套接口
sockfd=socket(sasend->sa_family, SOCK_RAW, icmpproto);
//为安全性考虑,将进程的有效用户 ID 设成进程的实际用户 ID
setuid(getuid());
//将套接口接收缓冲区设为 61440 字节,主要为了减少接收缓冲区溢出的可能性
size=60 * 1024;
setsockopt(sockfd, SOL_SOCKET, SO_RCVBUF, &size, sizeof(size));
//调用 SIGALRM 信号处理程序来发送一个数据包,并调度下一次 SIGALRM 为一秒之后
sig_alrm(SIGALRM);
for(; ;){
  len=salen;
  n=recvfrom(sockfd, recvbuf, sizeof(recvbuf), 0, sarecv, &len);
  //记录数据包收到的时间
  gettimeofday(&tval, NULL);
  //处理 ICMP
  proc_v4(recvbuf, n, &tval);
}
```

sig_alrm(int signo)函数的实现：

```
//SIGALRM 信号处理函数
sig_alrm(int signo)
{
  send_v4();
  //设置信号 SIGALRM 在经过 1 秒数后发送给目前的进程
  alarm(1);
```

```
  return;
}
```

proc_v4 函数的部分实现：

```
//IP 头部指针
ip=(struct ip *)ptr;                                    //start of IP header
//IP 头部长度。它是以 4 字节为一个单位计数的,将它乘以 4 才是以字节为单位的头部长度
hlen1=ip->ip_hl <<2;
//ICMP 头部指针
icmp=(struct icmp *)(ptr+hlen1);
//如果是 ICMP 回射响应,那么检查标识符字段,这个响应是不是这个进程发出的请求的响应
if(icmp->icmp_type==ICMP_ECHOREPLY){
  if(icmp->icmp_id !=pid)
    return;
  if(icmplen<16)
    err_quit("icmplen(%d)<16", icmplen);
  //将请求发出时间(包含在 ICMP 响应的可选数据部分内)与当前时间相减,可计算出 RTT(以毫
    秒为单位)
  tvsend=(struct timeval *)icmp->icmp_data;
  tv_sub(tvrecv, tvsend);
  rtt=tvrecv->tv_sec * 1000.0+tvrecv->tv_usec / 1000.0;
  …;
}
send_v4()函数的部分实现:
//指向 ICMP 头
icmp=(struct icmp *)sendbuf;
//ICMP 中类型
icmp->icmp_type=ICMP_ECHO;
…;
//ICMP 中数据。它的值为当前时间
gettimeofday((struct timeval *)icmp->icmp_data, NULL);
…;
//发送 ICMP。由于没使用 IP_HDRINCL 选项,内核将构造 IPv4 头部并将其放在缓冲区前面
sendto(sockfd, sendbuf, len, 0, sasend, salen);
```

3.3.5 进一步的扩展

ping 命令实现的扩充。在给定的 ping 程序的基础上做如下功能扩充：

(1) -h——显示帮助信息。

(2) -b——允许 ping 一个广播地址,只用于 IPv4。

(3) -t——设置 ttl 值,只用于 IPv4。

(4) -q——安静模式。不显示每个收到的包的分析结果,只在结束时显示汇总结果。

3.4 多播客户/服务器程序

3.4.1 课程设计目的

本课程设计主要目的是通过服务器向指定多播地址发送多播报文和客户加入多播组后接收多播报文，使学生掌握多播的工作原理，理解多播组地址到以太网地址的转换，在局域网中加入和离开多播组过程。

3.4.2 课程设计内容

本课程设计要求设计与实现 Linux 下多播客户/服务器程序，具体内容如下：

(1) 该程序能够实现以下功能：多播服务器能每 2 秒向指定多播地址发送多播报文，本局域网中的多播客户能够加入指定多播组，并能接收 8 个多播报文后离开多播组；

(2) 设计和实现多播客户/服务器程序；

(3) 能够用 Wireshark 分析的多播报文。

3.4.3 相关知识

IP 多播是一种允许一台或多台主机(多播源)发送单一数据包到多台主机的 TCP/IP 网络技术。多播作为一点对多点的通信，是节省网络带宽的有效方法之一。多播能使一个或多个多播源只把数据包发送给特定的多播组，而只有加入该多播组的主机才能接收到数据包。目前 IP 多播技术被广泛应用在直播体育比赛、向复制服务器池更新程序、网络视频会议、多媒体远程教育等方面。

1. IP 多播地址

范围从 224.0.0.0 到 239.255.255.255 的 IPv4 地址被划分为局部链接多播地址、预留多播地址和管理权限多播地址三类。其中局部链接多播地址范围在 224.0.0.0～224.0.0.255，这是为路由协议和其他用途保留的地址，路由器并不转发属于这个范围的 IP 包；预留多播地址为 224.0.1.0～238.255.255.255，可用于全球范围(如 Internet)或网络协议；管理权限多播地址为 239.0.0.0～239.255.255.255，可供组织内部使用，类似于私有 IP 地址，不能用于 Internet，可限制多播范围。

2. 多播组

使用同一个 IP 多播地址接收多播数据包的所有主机构成了一个主机组，也称为多播组。一个多播组的成员是随时变动的，一台主机可以随时加入或离开多播组，多播组成员的数目和所在的地理位置也不受限制，一台主机也可以属于几个多播组。同时不属于某一个多播组的主机也可以向该多播组发送数据包。一些多播组地址被 IANA 确定为知

名地址，它们被当作永久主机组。如，224.0.0.1 代表“子网内的所有支持多播的主机组”，224.0.0.2 代表“子网内的所有支持多播的路由器组”，224.0.1.1 用作网络时间协议 NTP。

3. 多播组地址到以太网地址的转换

IP 多播数据报传送依赖于低层物理网络，应用最广泛的以太网数据链路层支持多播功能。以太网的物理地址的第 1 个字节的最低位表示单地址或组地址：0 为单地址，1 为组地址，如多播地址 01:00:5E:00:00:01。IANA 拥有高位 24bit 为 00:00:5e 的以太网地址块，其中用于多播的以太网地址范围从 01:00:5e:00:00:00 到 01:00:5e:7f:ff:ff。当 IP 多播数据报交到底层以太网进行传送时，多播地址到以太网地址的映射关系如图 3-12 所示。

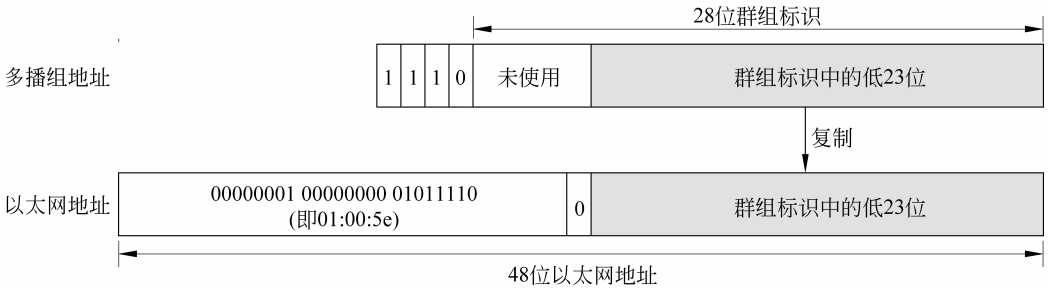

图 3-12　多播组地址到以太网地址映射

多播组号中的 5 bit 在映射过程中被忽略，地址映射是不唯一的，设备驱动程序或 IP 层就必须对数据报进行过滤。

4. 多播的工作过程

在局域网中，主机的网络接口将到目的主机的数据包发送到高层，这些数据包中的目的地址是物理接口地址或广播地址。如果主机已经加入到一个多播组中，主机的网络接口就会识别出发送到该组成员的数据包。因此，如果主机接口的物理地址为 8C:70:5A:46:F8:6C，其加入的多播组为 224.1.2.3，则发送给主机的数据包中的目的地址必是下面三种类型之一：接口地址：8C:70:5A:46:F8:6C，广播地址：FF:FF:FF:FF:FF:FF:FF:FF，多播地址：01:00:5E:01:10:11。

广域网中，路由器必须支持多播路由。当主机中运行的进程加入到某个多播组中时，主机向子网中的所有多播路由器发送 IGMP(Internet 组管理协议)报文，告诉路由器凡是发送到这个多播组的多播报文都必须发送到本地的子网中，这样主机的进程就可以接收到报文了。子网中的路由器再通知其他的路由器，这些路由器就知道该将多播报文转发到哪些子网中去。

子网中的路由器也向 224.0.0.1 发送一个 IGMP 报文，要求组中的主机提供组的相关信息。组中的主机收到这个报文后，经过一段随机时间，再向路由器发送响应。这样可防止了组中所有的主机同时向路由器发送响应，造成网络拥塞。主机向多播地址发送一

个报文作为对路由器的响应，组中的其他主机 一旦看到这个响应报文，就不再发送响应报文了，因为组中的主机向路由器提供的都是相同的信息。

如果组中的主机都退出了，路由器就收不到响应，路由器认为该组目前没有主机加入，于是停止到该子网报文的路由。IGMPv2 的解决方案是组中的主机在退出时向 224.0.0.2 发送报文通知多播路由器。

本课程设计的多播客户和服务器在同一局域网中。

本课程设计需要的 Linux 网络编程知识，请参考附录 B。

3.4.4 课程设计分析

1. 详细设计

多播客户程序流程图如图 3-13 所示。

多播服务器程序流程图如图 3-14 所示。

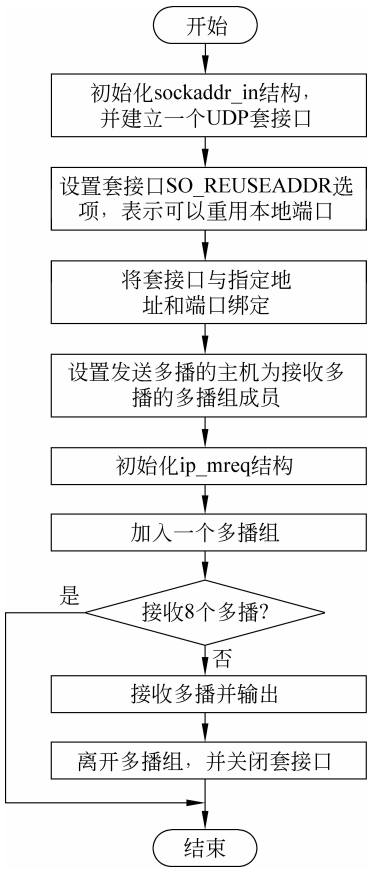

图 3-13 多播客户程序流程图

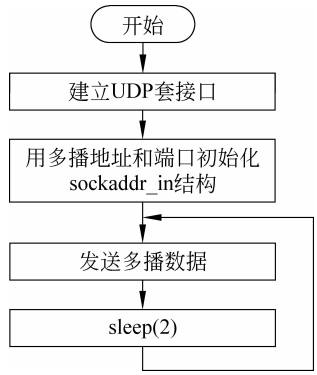

图 3-14 多播服务器程序流程图

2. 核心代码

(1) ip_mreq 结构在< >中定义。ip_mreq 结构如下所示。

ip_mreq 结构在<netinet/in.h>定义：

```
#if defined __USE_MISC || defined __USE_GNU
//IPv4 多播请求
struct ip_mreq
{
  struct in_addr imr_multiaddr;                    //IP 多播地址
  struct in_addr imr_interface;                    //本地接口的 IP 地址
};
```

(2) 多播客户和服务器程序的部分实现。

多播客户程序的部分实现：

```
//建立一个 UDP 套接口
sd=socket(PF_INET, SOCK_DGRAM, 0));
//设定 SO_REUSEADDR,允许多个应用绑定同一个本地端口接收数据包
loop=1;
setsockopt(sd, SOL_SOCKET, SO_REUSEADDR, &loop, sizeof(loop));
//用 bind 绑定本地端口,IP 为 INADDR_ANY,从而能接收多播播数据包
bind(sd,(struct sockaddr *)&sin, sizeof(sin));
//设置发送多播的主机为接收多播的多播组成员
loop=1;
setsockopt(sd, IPPROTO_IP, IP_MULTICAST_LOOP, &loop, sizeof(loop));
//将所加入的多播组的 IP 多播地址复制到一个 ip_mreq 结构中
command.imr_multiaddr.s_addr=inet_addr("224.1.2.3");
//所要加入多播组的那个本地接口的 IP 地址复制到一个 ip_mreq 结构中
command.imr_interface.s_addr=htonl(INADDR_ANY);
//加入一个多播组。进一步告诉 Linux 内核,特定的套接口即将接收多播数据
setsockopt(sd, IPPROTO_IP, IP_ADD_MEMBERSHIP, &command, sizeof(command))
//接收 8 个多播
while(iter++<8){
  sin_len=sizeof(sin);
  recvfrom(sd, message, 256, 0,(struct sockaddr *)&sin, &sin_len);
}
//退出多播组
setsockopt(sd, IPPROTO_IP, IP_DROP_MEMBERSHIP, &command, sizeof(command));
```

多播服务器程序的部分实现：

```
//建立 UDP 套接口
socket_descriptor=socket(AF_INET, SOCK_DGRAM, 0);
//用多播地址和端口初始化 sockaddr_in 结构
memset(&address, 0, sizeof(address));
address.sin_family=AF_INET;
```

```
address.sin_addr.s_addr=inet_addr("224.1.2.3");
address.sin_port=htons(port);
//开始 IP 多播
while(1){
  sendto(socket_descriptor, "test from multicast",
        sizeof("test from multicast"), 0,(struct sockaddr * )&address,
        sizeof(address));
  sleep(2);
}
```

3.5 UDP 报文发送程序

3.5.1 课程设计目的

UDP 协议是一个无连接的传输协议。本课程设计的主要目的是通过构造与发送 UDP 报文,掌握 UDP 协议的工作原理,理解 UDP 报文格式。

3.5.2 课程设计内容

本课程设计要求设计与实现 Linux 下 UDP 报文发送程序,具体内容如下:

(1) 该程序能够实现以下功能:能够获取指定接口的索引;能够构造 IP 头部和 UDP 报文,并向指定地址发送这个 UDP 报文;

(2) 设计和实现 UDP 报文发送程序;

(3) 能够用 Wireshark 分析发送的 UDP 报文。

3.5.3 相关知识

本课程设计需要的 IP 协议知识,请参考 2.4 节。

本课程设计需要的 UDP 协议知识,请参考 2.6 节。

本课程设计需要的 Linux 网络编程知识,请参考附录 B。

3.5.4 课程设计分析

1. 详细设计

UDP 报文发送程序的流程图如图 3-15 所示。

构造 UDP 头的流程图如图 3-16 所示。

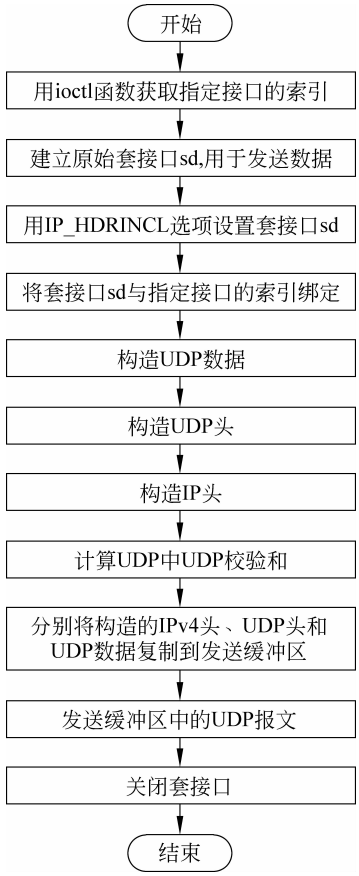

图 3-15　UDP 报文发送程序流程图

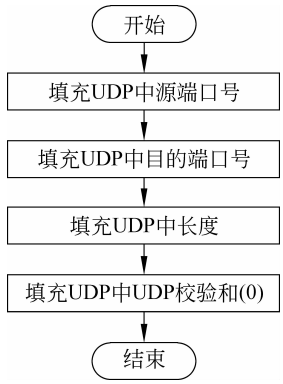

图 3-16　构造 UDP 头流程图

2. 核心代码

(1) IPv4 头结构在< netinet/ip. h >中定义。UDP 头结构在<netinet/udp. h>中定义。IPv4 结构如下所示：

```
//UDP header as specified by RFC 768, August 1980.
#ifdef __FAVOR_BSD
struct udphdr
{
  u_int16_t uh_sport;                    //源端口号
  u_int16_t uh_dport;                    //目的端口号
  u_int16_t uh_ulen;                     //UDP 长度
  u_int16_t uh_sum;                      //UDP 检验和
};
#else
struct udphdr
{
```