**多播服务器/客户端**

[第1章 前言 2](#_Toc1306198044)

[第2章 工作流程 2](#_Toc819162587)

[2.1 多播服务器的工作流程 2](#_Toc872009006)

[2.2 多播客户端的工作流程 3](#_Toc2071077126)

[第3章 主要的函数及功能 3](#_Toc1278251208)

[3.1 多播服务器3.1.1 main函数 3](#_Toc203405707)

[3.1.2 mcast\_join函数 5](#_Toc307424787)

[3.2 多播客户端 8](#_Toc485425529)

[3.2.1 main函数 8](#_Toc1286613895)

[3.2.2 dg\_cli函数 9](#_Toc2061537782)

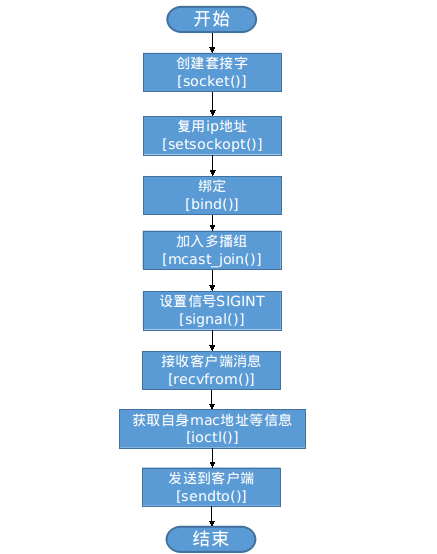
[第4章 总结 11](#_Toc1762142483)

# 第1章 前言

此程序的功能是客户端获取局域网内运行了服务器程序的主机的mac地址、ip地址等信息。权当学习使用，实用价值并不高，若要更实用的话，在程序内使用arp协议应该是不错的选择。此程序是在学习《unix网络编程（卷1）》的时候从书中程序分离出来的，本文主要介绍一下此程序的主要的函数和工作流程。

# 第2章 工作流程

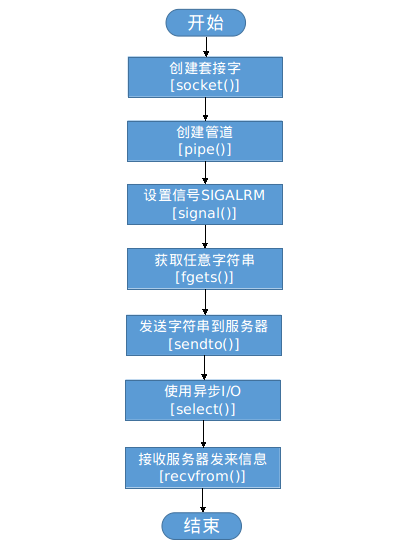
## 2.1 多播服务器的工作流程



使用地址复用是为了在一台主机开启多个服务器程序。

设置信号量的作用是：计算接收到的信息的数量。

## 2.2 多播客户端的工作流程



管道的作用是：配合信号函数使程序退出死循环，结束程序。

设置信号量的作用是：定时，时间到的时候写管道，管道读取到数据则跳出无限循环。

# 第3章 主要的函数及功能

## 3.1 多播服务器3.1.1 main函数

主函数显示了服务器程序的整体流程。首先创建套接字，填充结构，然后调用setsockopt函数设置地址复用，接下来把套接字和地址绑定，调用mcast\_join函数加入多播组，调用dg\_echo()函数收发信息。

#define MCAST\_IP "224.22.22.22"

int main(void)

{

int sockfd = 0;

struct sockaddr\_in servaddr; //服务器结构

struct sockaddr\_in grpaddr; //多播组结构

struct sockaddr\_in cliaddr; //客户端结构

const int on = 1; //用于开启选项

sockfd = my\_socket(AF\_INET, SOCK\_DGRAM, 0);

//初始化结构体并填充服务器信息

bzero(&servaddr, sizeof(servaddr));

servaddr.sin\_family = AF\_INET;

servaddr.sin\_addr.s\_addr = htonl(INADDR\_ANY); //主机转网络

servaddr.sin\_port = htons(SERV\_PORT); //主机转网络

//SO\_REUSEADDR地址复用;

//如果不使用这句, 则同一台主机中,则无法开启两个服务器

//bind的时候会失败:address already in use

//另一个方案"随机分配端口号"应该也可以

my\_setsockopt(sockfd, SOL\_SOCKET, SO\_REUSEADDR, &on, sizeof(on));

//把一个本地协议的地址赋予一个套接字

my\_bind(sockfd, (SA \*) &servaddr, sizeof(servaddr));

//初始化结构体并填充多播组信息

bzero(&grpaddr, sizeof(servaddr));

grpaddr.sin\_family = AF\_INET;

grpaddr.sin\_addr.s\_addr = inet\_addr(MCAST\_IP);

//加入多播组

mcast\_join(sockfd, (SA \*)&grpaddr, sizeof(grpaddr), NULL, 0);

//回射发来的信息, 以及时间, mac地址

dg\_echo(sockfd, (SA \*)&cliaddr, sizeof(cliaddr));

return 0;

}

### 3.1.2 mcast\_join函数

加入多播组需要本机的接口索引/名称和接口地址。

第一部分：ip无关套接字

/第一部分:加入一个多播组:IP无关套接字//

int mcast\_join(int sockfd, const SA \*grp, socklen\_t grplen,

const char \*ifname, u\_int ifindex)

{

#ifdef MCAST\_JOIN\_GROUP //编译的时候手工指定?

struct group\_req req;

if (ifindex > 0)

{

req.gr\_interface = ifindex;

}

else if (ifname != NULL)

{

if ( (req.gr\_interface = if\_nametoindex(ifname)) == 0)

{

errno = ENXIO; //i/f name not found

DEBUG;

printf("#errno = ENXIO!\n"); //+

return(-1);

}

}

else

{

req.gr\_interface = 0;

}

//为了防止下面复制的时候溢出, 先检查大小

if (grplen > sizeof(req.gr\_group))

{

errno = EINVAL; //22, invalid argument

DEBUG;

printf("#errno = EINVAL!\n");

return (-1);

}

memcpy(&req.gr\_group, grp, grplen);

//setsockopt执行加入操作

//my\_family\_to\_level用来确定协议类型

return (setsockopt(sockfd, my\_family\_to\_level(grp->sa\_family),

MCAST\_JOIN\_GROUP, &req, sizeof(req)));

//第一部分:加入一个多播组:IP无关套接字//

定义的req结构是ip无关套接字的，通过填充相关信息后作为setsockopt函数的参数，以加入多播组。

第二部分：ipv4套接字

结构ip\_mreq是加入多播组的时候使用的，结构ifreq是获取接口信息的时候使用的。

首先填充mreq结构的多播地址字段，接下来判断是否知道接口索引，如果知道则通过接口索引获取接口名字，再通过接口名字获取接口地址，注意doioctl标志，用的很巧妙。

#else //默认用了这个

//第二部分:加入一个多播组:IPv4套接字//

switch (grp->sa\_family)

{

case AF\_INET:

struct ip\_mreq mreq; //多播组地址+ifreq结构

struct ifreq ifreq; //接口名称+union(地址, 目的地址...)

memcpy(&mreq.imr\_multiaddr,

&((const SA\_IN \*) grp)->sin\_addr,

sizeof(struct in\_addr));

if (ifindex > 0)

{

if (if\_indextoname(ifindex, ;ifreq.ifr\_name) == NULL)

{

errno = ENXIO; //6, no such device or address

printf("errno = ENXIO!\n");

return(-1);

}

goto doioctl; //转成名字了, 跳下去执行

}

else if (ifname != NULL)

{

strncpy(ifreq.ifr\_name, ifname, IFNAMSIZ);

doioctl:

if (ioctl(sockfd, SIOCGIFADDR, &ifreq) < 0)

{

printf("ioctl error!\n"); //+

return (-1);

}

memcpy(&mreq.imr\_interface,

&((SA\_IN \*) &ifreq.ifr\_addr)->sin\_addr,

sizeof(struct in\_addr));

}

else

{

mreq.imr\_interface.s\_addr = htonl(INADDR\_ANY);

}

return (setsockopt(sockfd, IPPROTO\_IP,

IP\_ADDR\_MEMBERSHIP,

&mreq, sizeof(mreq)));

//第二部分:加入一个多播组:IPv4套接字//

第三部分：IPv6套接字

情况和IPv4的类似，不同的是使用不同的结构。

//第三部分:加入一个多播组:IPv6套接字//

#ifdef IPV6

case AF\_INET6:

struct ipv6\_mreq mreq6;

memcpy(&mreq6, ipv6mr\_multiaddr,

&((const SA\_IN6 \*) grp)->sin6\_addr,

sizeof(struct in6\_addr));

if (ifindex > 0)

{

mreq6.ipv6mr\_interface = ifindex;

}

else if (ifname != NULL)

{

if ((mreq6.ipv6mr\_interface

= if\_nametoindex(ifname)) == 0)

{

errno = ENXIO;

DEBUG;

printf("#errno = ENXIO!\n"); //+

return (-1);

}

}

else

{

mreq6.ipv6mr\_interface = 0;

}

return(setsockopt(sockfd, IPPROTO\_IPV6,

IPV6\_JOIN\_GROUP,

&mreq6, sizeof(mreq60)));

#endif

default:

errno = EAFNOSUPPORT; //97: address family not supported by protocol

DEBUG;

printf("errno = EAFNOSUPPORT!\n");

return (-1);

}

#endif

}

//第三部分: 加入一个多播组:IPv6套接字//

## 3.2 多播客户端

## 3.2.1 main函数

主函数主要是创建了套接字，然后调用了dg\_cli()函数

#define MCAST\_IP "224.22.22.22"

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

{

int sockfd = 0;

struct sockaddr\_in servaddr;

if (argc < 2)

{

err\_quit("usage: ./a.out <IPaddress>");

}

bzero(&servaddr, sizeof(servaddr));

servaddr.sin\_family = AF\_INET;

servaddr.sin\_port = htons(SERV\_PORT); //将一个无符号短整型的主机数值转换为网络字节顺序

my\_inet\_pton(AF\_INET, MCAST\_IP, &servaddr.sin\_addr); //十进制IP转二进制

//创建一个套接字

sockfd = my\_socket(AF\_INET, SOCK\_DGRAM, 0);

dg\_cli(stdin, sockfd, (SA \*) &servaddr, sizeof(servaddr));

exit(0);

}

## 3.2.2 dg\_cli函数

发送任意信息到多播地址，接收多播服务器返回的信息。定时3秒是为了应付服务器多的情况。防止客户端还没接收完服务器发来的信息就又再次获取。

void dg\_cli(FILE \*fp, int sockfd, const SA \*pservaddr, socklen\_t servlen)

{

int n = 0;

int maxfdp1 = 0;

const int on = 1;

char sendline[MAXLINE] = {0};

//+64是因为对端还发送mac地址,和时间过来

char recvline[MAXLINE + 64] = {0};

fd\_set rset;

socklen\_t len;

struct sockaddr \*preply\_addr = NULL;

preply\_addr = my\_malloc(servlen);

//设置套接字选项;

//SO\_BROADCAST 允许广播数据报

//SOL\_SOCKET是级别(level)[套接字级别?]

//可以不要

// my\_setsockopt(sockfd, SOL\_SOCKET, SO\_BROADCAST, &on, sizeof(on));

//创建管道

my\_pipe(pipefd);

// printf("pipefd[0] = %d\n", pipefd[0]); //+

maxfdp1 = max(sockfd, pipefd[0]) + 1; //max是一个宏, 应该改大写的.

FD\_ZERO(&rset); //清空文件描述符集;套接字也是文件描述符的一种

my\_signal(SIGALRM, recvfrom\_alarm); //设置时间信号规则

while (my\_fgets(sendline, MAXLINE, fp) != NULL ) //输入任意字符

{

my\_sendto(sockfd, sendline, strlen(sendline),

0, pservaddr, servlen); //发送给服务器

alarm(3); //设置定时3秒

while (1)

{

FD\_SET(sockfd, &rset); //设置描述符集

FD\_SET(pipefd[0], &rset);

//I/O复用

if ( (n = select(maxfdp1, &rset, NULL, NULL, NULL)) < 0 )

{

if (errno == EINTR)

{

DEBUG; //一个宏, 测试用

printf("select errno = EINTR\n"); //select失败的errno值

continue; //忽略错误, 继续循环

}

else

{

err\_sys("select error");

}

}

// printf("kankan!\n");

if ( FD\_ISSET(sockfd, &rset) ) //套接字可读, 表示有消息到来

{

len = servlen;

n = my\_recvfrom(sockfd, recvline, MAXLINE,

0, preply\_addr, &len); //接收信息

recvline[n] = '\0';

printf("\nfrom %s: %s",

my\_sock\_ntop(preply\_addr, len), //二进制转10进制

recvline); //打印服务器IP和返回的信息

}

if ( FD\_ISSET(pipefd[0], &rset) )

{

// printf("read 上面\n");

my\_read(pipefd[0], &n, 1); //timer expired

// printf("read 下面!\n");

break; //退出循环

}

}

}

free(preply\_addr);

}

# 第4章 总结

当时是有人给我一个类似功能的程序让我看，然后重现出来，无奈定义太多结构以及知识点太多不懂，无法完成。一气之下把《unix网络编程卷1》前20多章又过了一遍，从书中源码分离出这个程序，从书中、程序中，都学到很多东西。个人感觉这是个很不错的程序，值得学习，不得不佩服stevens大神。学习写一些移植性高的程序的感觉真的不错。