af\_inet.c

<https://www.tenouk.com/Module41c.html>

struct inet\_protosw inetsw\_array[] = {

{

.type = SOCK\_DGRAM,

.protocol = IPPTOTO\_UDP,

.prot = &udp\_prot,

.ops = &inet\_dgram\_ops,

.flags = INET\_PROTOSW\_PERMANENT,

}

};

struct proto udp\_prot = {

.setsockopt = udp\_setsockopt,

};

udp\_setsockopt(sk,level,optname,optval,optlen)

{

if(level==SOL\_UDP || level==SOL\_UDPLITE)

return

return ip\_setsocktopt(sk,level,optname,optval,optlen);

}

ip\_setsocktopt(sk,level,optname,optval,optlen)

{

int err;

err = do\_ip\_setsockopt(sk,level,optname,optval,optlen);

}

struct ip\_mreqn {

struct in\_addr imr\_multiaddr;/\*ip multicast address of group\*/

struct in\_addr imr\_address;/\*local ip address of interface\*/

int imr\_ifindex;

};

do\_ip\_setsockopt(struct sock \*sk,int level,int optname,char \*optval,unsigned int optlen)

{

int val;

struct inet\_sock \*inet = inet\_sk(sk);

switch(optname) {

case IP\_MULTICAST\_TTL:

case IP\_MULTICAST\_ALL:

case IP\_MULTICAST\_LOOP:

if(optlen>=sizeof(int)) {

if(get\_user(val,(int \_\_user\*)optval))

return –EFAULT;

}

}

switch(optname) {

case IP\_MULTICAST\_TTL:

if(sk->sk\_type==SOCK\_STREAM)

goto e\_inval;

if(optlen<1)

goto e\_inval;

if(val==-1)

val =1;

inet->mc\_ttl = val;

break;

case IP\_MULTICAST\_LOOP:

inet->mc\_loop = !!val;

case IP\_MULTICAST\_IF:

struct ip\_mreqn mreq;

struct net\_device \*dev = NULL;

if(optlen>=sizeof(struct ip\_mreqn)) {

if(copy\_from\_user(&mreq,optval,sizeof(mreq)))

break;

}else {

memset(&mreq,0,sizeof(mreq));

if(optlen>=sizeof(struct in\_addr)) {

copy\_from\_user(&mreq.imr\_address,optval,

sizeof(struct in\_addr));

break;

}

}

if(!mreq.imr\_ifindex) {

if(mreq.imr\_address.s\_addr==htonl(INADDR\_ANY)) {

inet->mc\_index = 0;

inet->mc\_addr = 0;

break;

}

dev = in\_dev\_find(sock\_net(sk),mreq.imr\_address.s\_addr);

if(dev)

mreq.imr\_ifindex = dev->ifindex;

inet->mc\_index = mreq.imr\_ifindex;

inet->mc\_addr = mreq.imr\_address.s\_addr;

break;

}

case IP\_ADD\_MEMBERSHIP:

case IP\_DROP\_MEMBERSHIP:

{

struct ip\_mreqn mreq;

copy\_from\_user(&mreq,optval,sizeof(mreq));

if(optname == IP\_ADD\_MEMBERSHIP)

err = ip\_mc\_join\_group(sk,&mreq);

else

err =ip\_mc\_leave\_group(sk,&mreq);

break;

}

}

}

struct ip\_mc\_socklist {

struct ip\_mc\_socklist \_\_rcu \*next\_rcu;

struct ip\_mreqn multi;

unsigned int sfmode;

};

struct inet\_sock {

struct ip\_mc\_socklist \_\_rcu \*mc\_list;

};

ip\_mc\_join\_group(sk,&mreq)->

\_\_ip\_mc\_join\_group(sk,imr,MCAST\_EXCLUDE)

{

\_\_be32 addr = imr->imr\_multicast.s\_addr;

int ifindex;

struct inet\_sock \*inet=inet\_sk(sk);

struct ip\_mc\_socklist \*iml,\*i;

int count =0;

struct net\*net = sock\_net(sk);

struct in\_device \*in\_dev;

in\_dev = ip\_mc\_find\_dev(net,imr);

if(!ipv4\_is\_multicast(addr))

return –EINVAL;

in\_dev = ip\_mc\_find\_dev(net,imr);

ifindex = imr->imr\_ifindex;

for\_each\_pmc\_rtnl(inet,i) {

if(i->multi.imr\_mutiaddr.s\_addr==addr&&

i->multi.imr\_ifindex ==ifindex)

goto done;

count++;

}

iml = sock\_kmalloc(sk,sizeof(\*iml),GFP\_KERNEL);

memcpy(&iml->multi,imr,sizeof(\*imr));

iml->next\_rcu=inet->mc\_list;

iml->sflist =NULL;

iml->sfmode = mode;

rcu\_assign\_pointer(inet->mc\_list,iml);

\_\_\_\_ip\_mc\_inc\_group(in\_dev,addr,mode,GFP\_KERNEL);

}

\_\_\_\_ip\_mc\_inc\_group(in\_dev,addr,mode,GFP\_KERNEL)

{

struct ip\_mc\_list \*im;

for\_each\_pmc\_rtnl(in\_dev,im) {

if(im->multiaddr==addr) {

im->users++;

ip\_mc\_add\_src(in\_dev,&addr,mode,0,NULL,0);

goto out;

}

}

igmp\_group\_added(im);

}

igmp\_group\_added(im);

{

if(im->loaded==0) {

im->loaded =0;

ip\_mc\_filter\_add(in\_dev,im->multicast);

}

}

ip\_mc\_find\_dev(net,imr)

{

struct net\_device \*dev = NULL;

struct in\_device \*idev = NULL;

if(imr->imr\_ifindex) {

idev = inetdev\_by\_index(net,imr->imr\_ifindex);

}

}

struct net\_device {

struct in\_device \_\_rcu \*ip\_ptr;

};

inetdev\_by\_index(net,imr->imr\_ifindex)

{

struct net\_device \*dev;

dev = dev\_get\_by\_index\_rcu(net,imr->imr\_ifindex);

if(dev)

in\_dev = rcu\_dereference\_rtnl(dev->ip\_ptr);

return in\_dev;

}

struct ip\_mc\_list {

};

ip\_mc\_add\_src(in\_dev,&addr,mode,0,NULL,0)

{

struct ip\_mc\_list \*pmc;

int isexclude;

for\_each\_pmc\_rcu(in\_dev,pmc) {

if(\*pmca==pmc->multiaddr)

break;

}

}

## how switch senses

packet