PF\_INET socket

1.       parameters of function socket(int domain, int type,  int protocol)

a.       domain：A  communication domain  such as PF\_INET, PF\_PACKET

structure  of the domain

  struct net\_proto\_family {

      int             family;

      int             (\*create)(struct net \*net, struct socket \*sock,  int protocol, int kern);

      struct module         \*owner;

};

PF\_INET

static const struct net\_proto\_family inet\_family\_ops = {

      .family = PF\_INET,

      .create = inet\_create,

      .owner    = THIS\_MODULE,

};

b.            type           &         protocol

SOCK\_STREAM         IPPROTO\_TCP

SOCK\_DGRAM          IPPROTO\_UDP

SOCK\_DGRAM          IPPROTO\_ ICMP

SOCK\_RAW              IPPROTO\_IP

1)      Ops of type

struct proto\_ops :operation of a type . There are three kinds of ops,  inet\_dgram\_ops, inet\_sockraw\_ops, inet\_stream\_ops.

struct proto\_ops {

      int             family;

      struct module         \*owner;

      int             (\*release)   (struct socket \*sock);

      int             (\*bind)         (struct socket \*sock,

                                        struct sockaddr \*myaddr,

                                        int sockaddr\_len);

      int             (\*accept)    (struct socket \*sock,

                                        struct socket \*newsock, int flags);]

      int             (\*ioctl)     (struct socket \*sock, unsigned int cmd,

                                        unsigned long arg);

      .....

      }

2)      Ops of protocol

struct proto : operation structure of a protocol

struct proto tcp\_prot = {

      .name                        = "TCP",

      .obj\_size                   = sizeof(struct tcp\_sock),

      .close                          = tcp\_close,

      .connect          = tcp\_v4\_connect,

      .disconnect              = tcp\_disconnect,

.recvmsg           = tcp\_recvmsg,

      .sendmsg                  = tcp\_sendmsg,

      .....

};

struct proto udp\_prot = {

      .name                  = "UDP",

      .obj\_size                   = sizeof(struct udp\_sock),

      .close                   = udp\_lib\_close,

      .ioctl                     = udp\_ioctl,

      .sendmsg           = udp\_sendmsg,

      .recvmsg             = udp\_recvmsg,

      .....

};

3)      Structure of  type&protocol

struct inet\_protosw {

      struct list\_head list;

        /\* These two fields form the lookup key.  \*/

      unsigned short       type;         /\* This is the 2nd argument to socket(2). \*/

      unsigned short       protocol; /\* This is the L4 protocol number.  \*/

      struct proto    \*prot;

      const struct proto\_ops \*ops;

      char             no\_check;   /\* checksum on rcv/xmit/none? \*/

      unsigned char         flags;      /\* See INET\_PROTOSW\_\* below.  \*/

};

static struct inet\_protosw inetsw\_array[] =

{

      {

                .type =       SOCK\_STREAM,

                .protocol =   IPPROTO\_TCP,

                .prot =       &tcp\_prot,

                .ops =        &inet\_stream\_ops,

                .no\_check =   0,

                .flags =      INET\_PROTOSW\_PERMANENT |

                               INET\_PROTOSW\_ICSK,

      },

      {

                .type =       SOCK\_DGRAM,

                .protocol =   IPPROTO\_UDP,

                .prot =       &udp\_prot,

                .ops =        &inet\_dgram\_ops,

                .no\_check =   UDP\_CSUM\_DEFAULT,

                .flags =      INET\_PROTOSW\_PERMANENT,

       },

       {

             .type =       SOCK\_RAW,

             .protocol =   IPPROTO\_IP,   /\* wild card \*/

             .prot =       &raw\_prot,

             .ops =        &inet\_sockraw\_ops,

             .no\_check =   UDP\_CSUM\_DEFAULT,

             .flags =      INET\_PROTOSW\_REUSE,

       }

};

The initialization of  PF\_INET

int inet\_init()

{

1)      Init the domain of  PF\_INET

(void)sock\_register(&inet\_family\_ops);

2)      Resiger operation of all protocols ops

rc = proto\_register(&tcp\_prot, 1);

rc = proto\_register(&udp\_prot, 1);

rc = proto\_register(&raw\_prot, 1);

3)      Resiger  type&protocol

for (q = inetsw\_array; q < &inetsw\_array[INETSW\_ARRAY\_LEN]; ++q)

                inet\_register\_protosw(q);

}

2.       Start a socket: socket(x, x, x);

socket->sys\_socket ->socket\_creat->inet\_family\_ops->create

int inet\_create()

{

1)      Find the type&protocol through protocol and type parameters

struct inet\_protosw \*answer;

list\_for\_each\_entry\_rcu(answer, &inetsw[sock->type], list)

2)      Store the socket type ops

sock->ops = answer->ops;

3)      Get protocol ops and allocate sk

struct proto \*answer\_prot;

answer\_prot = answer->prot;

sk = sk\_alloc(net, PF\_INET, GFP\_KERNEL, answer\_prot);

/\* sk->sk\_prot = answer\_prot, the sizeof sk is decided by answer\_prot->obj\_size, for udp the structure is also udp\_sock, tcp is also tcp\_sock\*/

}

3.       Ops of the socket

xxx: bind, close, connect, sendmsg …

 /\*first find the type ops then find the protocol ops\*/

xxx:sock->ops->xxx

{

     struct sock \*sk = sock->sk;

     if (sk->sk\_prot->xxx) {

           err = sk->sk\_prot->xxx(sk, uaddr, addr\_len);

           goto out;

  }

}

1. Ioctl

socket\_file\_ops->sock\_ioctl

{

case SIOCXXXBR:

/\*bridge\*/

.....

case SIOCXXVLAN:

/\*vlan\*/

.....

case SIOCXXDLCI:

/\*dlci\*/

.....

case xxxxxx:

/\*net device attribute\*/

dev\_ioctl(net, cmd, argp); /\*for netdevice attribute, it maybe needs dev->ops sometimes \*/

if (sock->ops-ioctl)

sock->ops->ioctl(sock, cmd, arg);

/\*inet\_stream\_ops->inet\_ioctl(sock, cmd, arg);\*/

}

int inet\_ioctl(struct socket \*sock, unsigned int cmd, unsigned long arg)

{

case SIOXXXRT:

/\*route\*/

ip\_rt\_ioctl

case SIOXXXARP:

/\*route\*/

arp\_ioctl

case SIOXXXADRR:

case SIOXXXNETMASK:

/\*net address\*/

devinet\_ioctl

if (sk->sk\_prot->ioctl)

err = sk->sk\_prot->ioctl(sk, cmd, arg);

/\*tcp\_prot->ioctl or udp\_prot->ioctl\*/

}