一、sockaddr

sockaddr在头文件#include <sys/socket.h>中定义，sockaddr的缺陷是：sa\_data把目标地址和端口信息混在一起了，如下：

struct sockaddr {

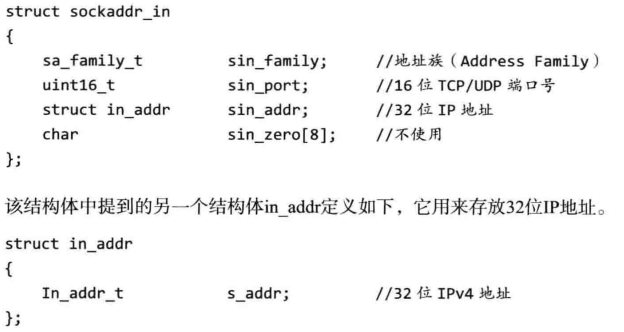
sa\_family\_t sin\_family;//地址族

　　 char sa\_data[14]; //14字节，包含套接字中的目标地址和端口信息

　　 };

二、sockaddr\_in

sockaddr\_in在头文件#include<netinet/in.h>或#include <arpa/inet.h>中定义，该结构体解决了sockaddr的缺陷，把port和addr 分开储存在两个变量中，如下：



总结：

二者长度一样，都是16个字节，即占用的内存大小是一致的，因此可以互相转化。二者是并列结构，指向sockaddr\_in结构的指针也可以指向sockaddr。

sockaddr常用于bind、connect、recvfrom、sendto等函数的参数，指明地址信息，是一种通用的套接字地址。

sockaddr\_in 是internet环境下套接字的地址形式。所以在网络编程中我们会对sockaddr\_in结构体进行操作，使用sockaddr\_in来建立所需的信息，最后使用类型转化就可以了。一般先把sockaddr\_in变量赋值后，强制类型转换后传入用sockaddr做参数的函数：sockaddr\_in用于socket定义和赋值；sockaddr用于函数参数。

例子：

#include <stdio.h>

#include <stdlib.h>

#include <sys/socket.h>

#include <netinet/in.h>

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

{

int sockfd;

struct sockaddr\_in mysock;

sockfd = socket(AF\_INET,SOCK\_STREAM,0); //获得fd

bzero(&mysock,sizeof(mysock)); //初始化结构体

mysock.sin\_family = AF\_INET; //设置地址家族

mysock.sin\_port = htons(800); //设置端口

mysock.sin\_addr.s\_addr = inet\_addr("192.168.1.0"); //设置地址

bind(sockfd,(struct sockaddr \*)&mysock,sizeof(struct sockaddr); /\* bind的时候进行转化 \*/

... ...

return 0;

}



TEST:

N-TSED TBP：

client\_socket(ip,port) //启动Procy-P,内容包括socket套接字启动

IF SOCKET\_FAIL //检查设备是否在线，如果启动失败

THEN StartException(SOCKET\_FAIL) //打印失败原因

timestamp = time(0) //获取时间，防止超时

random\_num = Random() //首次启动生成随机数，防止消息重放

SEND:

sys\_info = CollectInfo() //包括硬件序列号，操作系统版本，启动分区的哈希值、 // proxy-P的哈希值

enc\_sys\_info = Enc(sys\_info,pub\_key) //用服务器公钥加密系统信息

sig\_hash = Hash(enc\_sys\_info|random\_num|timestamp) //对需要发送的消息进行哈希

sig = Sign(sig\_hash,pri\_key) //用自己的私钥对哈希值进行签名

data = { {"sysInfo",enc\_sys\_info}, //data

{"timestamp",timestamp},

{"random",random\_num},

{"sign",sig}

{"ID",DEVICE\_ID}}

command = TBP\_STARTUP //区分哪个协议返回的服务器消息

msg = {{"command",command}, {"data",data}} //要发送的数据

send(client\_socket,msg)

...

rep\_msg = recv(client\_socket)

rep\_command = rep\_msg["command"]

SWITCH rep\_command

CASE TBP\_RESULT:

IF rep\_msg["data"] == TBP\_RESULT\_TIMEOUT ||

rep\_msg["data"] == TBP\_RESULT\_REPLAY

THEN

Renew(random\_num,timestamp)

goto SEND

IF rep\_msg["data"] == TBP\_RESULT\_FAILED

StartException(VERIFY\_FAIL)

IF rep\_msg["data"] == TBP\_RESULT\_SUCCEED

THEN LoadOS()

CASE RVP\_RESULT:

...

CASE NAP\_RESULT:

...

DEFAULT COMMAND\_MISMATCH:

THEN Result\_Exception(COMMAND\_MISMATCH)

...

TSED TBP：

server\_socket(port) // 启动Procy-V,内容包括socket套接字启动

WHILE TRUE

client\_msg = recv(server\_socket) //接收客户端的消息

command = client\_msg["command"] //区分哪个协议发来的客户端请求

data = client\_msg["data"]

SWITCH command:

CASE TBP\_STARTUP:

ID = data["ID"]

dev = DeviceList.find(ID)

old\_ip = dev.getIP();

IF old\_ip != ip //更新设备IP

dev.setIP(ip)

IF random\_num[ID] == NULL //判断是否为重放消息

THEN randon\_num[ID] = data["random"]

ELSE

IF CheckRandom(random\_num,data["random"]) == FALSE

THEN

IF CheckReplay(replay\_cnt[ID]) == TRUE

THEN

msg = {{"command",TBP\_RESULT},

{"data",TBP\_RESULT\_REPLAY}}

send(server\_socket,msg)

IF CheckTime(data["timestamp"]) == FLASE //时间戳判断是否超时

THEN

msg = {{"command",TBP\_RESULT},

{"data",TBP\_RESULT\_TIMEOUT}}

send(server\_socket,msg)

msg\_hash = dec(pub\_key,data["sign"]) //解密获得哈希值

sig\_hash = Hash(data["sys\_info"]|data["random"]|data["timestamp"])

IF msg\_hash != sig\_hash

THEN

msg = {{"command",TBP\_RESULT},

{"data",TBP\_RESULT\_FAILED}}

send(server\_socket,msg)

sys\_info = dec(data["sys\_info"],pri\_key)

PCR\_Reset() //PCR寄存器置0

PCR\_Extend(sys\_info)

extend\_pcr = PCR\_Read() //读取扩展之后的PCR

nv\_verify\_pcr = NV\_Read()

verify\_pcr = Unseal(nv\_verify\_pcr) //得到预先存储在nv中的pcr扩展值

IF extend\_pcr == verify\_pcr

THEN

msg = {{"command",TBP\_RESULT},

{"data",TBP\_RESULT\_SUCCEED}}

send(server\_socket,msg)

ELSE

THEN

msg = {{"command",TBP\_RESULT},

{"data",TBP\_RESULT\_FAILED}}

send(server\_socket,msg)