# 应用编程与网络编程

硬件部分：网卡

操作系统底层：socket接口

应用层：低级（直接基于socket接口编程）

应用层: 高级（基于网络通信应用框架库）

应用层：更高级（http、网络控件（GUI））

路由器对外的接口是WAN口，对内的接口是LAN口；

所以拿到路由器先设置WAN口的拨号设置。

路由器有两个网卡，对内作网关，对外作节点。

NAT穿透技术：让两个在局域网内部的电脑通过一个曾经都连过的服务器牵线，使二者都知道对方所在局域网路由器的地址。然后进行p2p通信。

socket编程接口连接

socket #include<sys/socket.h> int socket(int domain,int type,int protocol) ,类似于open函数，用来打开一个网络连接，如果成功，就返回一个网络文件描述符。之后操作网络连接都通过这个网络描述符。

bind

listen

connect

发送和接受

send和write

recv和read

辅助性函数

inet\_aton 、 inet\_addr 、 inet\_ntoa 只支持ipv4

inet\_ntop 、 int\_pton ipv4和ipv6都支持 32位二进制转换为点分十进制ntop（network to p（string 点分十进制））

表示IP地址相关数据结构

都定义在netinet/in.h cat /usr/include/netinet/in.h

struct sockaddr :这个结构体是网络编程接口中用来表示一个IP地址的，这个IP地址是不区分IPv4和IPv6的。

typedef uint32\_t in\_addr\_t 网络内部用于表示IP地址的类型

struct in\_addr{

in\_addr\_t s\_addr;

};

struct sockaddr\_in

{

\_\_SOCKADDR\_COMMON (sin\_);

in\_port\_t sin\_port; /\* Port number. \*/

struct in\_addr sin\_addr; /\* Internet address. \*/

/\* Pad to size of `struct sockaddr'. \*/

unsigned char sin\_zero[sizeof (struct sockaddr) -

\_\_SOCKADDR\_COMMON\_SIZE -

sizeof (in\_port\_t) -

sizeof (struct in\_addr)];

};

struct sockaddr 这个结构体式Linux网络编程接口中用来表示IP地址的标准结构体，bind、connect等函数体中都需要这个结构体，这个结构体是兼容IPv4和IPv6的，在实际编程中这个结构体会被一个struct sockaddr\_in 或者一个struct\_sockaddr\_in6所填充

#include<stdio.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#define IPADDR "192.168.1.102"

//0x66 01 a8 c0

//102 1 168 192

//网络字节序，其实就是大端模式

int main(void){

int\_addr\_t addr =0; //32位二进制

addr =inet\_addr(IPADDR); inet\_addr 将点分十进制转换为二进制

printf("addr=0x%x.\n",addr); //0x6601a8c0

return 0;

}

#include <arpa/inet.h>

#define IPADDR "192.168.1.102"

int main(void){

//使用inet\_pton来转换

int ret=0;

struct in\_addr addr ={0};

ret=inet\_pton(AF\_INET,IPADDR,&addr);

if(ret!=1){

printf("inet\_pton.\n");

return -1;

}

printf("add=0x%x.\n",addr.s\_addr);

return 0;

}

#include <arpa/inet.h>

int main(void){

//使用inet\_ntop来转换

//char \*ret=NULL;

struct in\_addr addr ={0};

char buf[50]={0};

addr.s\_addr=0x6601a8c0;

inet\_ntop(AF\_INET,&addr,buf,sizeof(buf));

//if(ret==NULL){

//printf("ip address error.\n");

//return -1;

//}

printf("ip address is %s.\n",buf);

return 0;

}

socket 实践编程

服务器端程序编写

socket

bind

listen

accept 返回值是一个fd，正确返回就表示已经和前来连接我的客户端建立了一个TCP连接。以后我们就要通过这个链接来和客户端进行读写操作，读写操作需要一个fd，这个fd就需要accept来返回。

客户端程序编写

socket

connect

### server.c

#include<stdio.h>

#include<sys/socket.h>

#include<sys/types.h>

#include<arpa/inet.h>

#define SERPORT 6003

#define BACKLOG 100//负载队列

#define SERADDR "140.82.22.121" //ifconfig 查看自己的ip地址

int main(){

//第一步先socket打开文件描述符

int sockfd=-1,ret=-1;

struct sockaddr\_in seraddr={0};

struct sockaddr\_in cliaddr={0};

socklen\_t len=0;

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

if(-1==sockfd){

perror("socket");

return -1;

}

printf("socketfd=%d.\n",sockfd);

//第二步 bind绑定sockfd和当前电脑的ip地址和端口号

seraddr.sin\_family=AF\_INET; //设置地址族为ipv4

seraddr.sin\_port=htons(SERPORT); //转换字节序为大端模式 man 3 byteorder

seraddr.sin\_addr.s\_addr=inet\_addr(SERADDR);//设置ip地址

bind(sockfd,(const struct sockaddr \*)&seraddr,sizeof(seraddr));

if(ret<0){

perror("bind");

return -1;

}

printf("bind sucess.\n");

//第三步，listen监听端口

ret=listen(sockfd,BACKLOG);

if(ret<0){

perror("listen");

return -1;

}

//第四部，accept阻塞等待

ret=accept(sockfd,(struct sockaddr \*)&cliaddr,&len);

printf("hahaha ret=%d.\n",ret);

return 0;

}

### client.c

#include<stdio.h>

#include<sys/socket.h>

#include<sys/types.h>

#include<arpa/inet.h>

#define SERPORT 6003

#define SERADDR "140.82.22.121" //ifconfig 查看服务器ip地址

int main(){

//第一步先socket打开文件描述符

int sockfd=-1,ret=-1;

struct sockaddr\_in seraddr={0};

socklen\_t len=0;

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

if(-1==sockfd){

perror("socket");

return -1;

}

printf("socketfd=%d.\n",sockfd);

//第二步 通过connect连接服务器

seraddr.sin\_family=AF\_INET; //设置地址族为ipv4

seraddr.sin\_port=htons(SERPORT ); //转换字节序为大端模式 man 3 byteorder

seraddr.sin\_addr.s\_addr=inet\_addr(SERADDR);//设置ip地址

connect(sockfd,(const struct sockaddr\*) &seraddr,sizeof(seraddr));

if(ret<0){

perror("connect");

return -1;

}

printf("connect sucess.ret=%d.\n",ret);

return 0;

}

## 使用gdb调试：

gcc -g server.c -o server.o

gdb ./server.o

start

next 简写 n 单步执行

step s step into

list l 显示附近代码

list 6,21 显示行号范围内的代码

list file.c:函数名 指定函数显示

print expr p 打印expr值

display

quit q 退出gdb

run 提示是否需要重新运行程序，输入y重新开始运行程序

break file.c:行号 b file.c:18 如果是单文件则可以直接 b 行号

break file.c:行号 if expr 条件断点

info b 显示当前断点信息，尤其是第一个num 可以用来删除断点

delete breakpoints num 删除断点

continue c 继续执行到下一个断点

### server.c

#include<stdio.h>

#include<sys/socket.h>

#include<sys/types.h>

#include<arpa/inet.h>

#include<string.h>

#define SERPORT 6003

#define BACKLOG 100//负载队列

#define SERADDR "140.82.22.121" //ifconfig 查看自己的ip地址

char recvbuf[100];

int main(){

//第一步先socket打开文件描述符

int sockfd=-1,ret=-1;

struct sockaddr\_in seraddr={0};

struct sockaddr\_in cliaddr={0};

socklen\_t len=0;

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

if(-1==sockfd){

perror("socket");

return -1;

}

printf("socketfd=%d.\n",sockfd);

//第二步 bind绑定sockfd和当前电脑的ip地址和端口号

seraddr.sin\_family=AF\_INET; //设置地址族为ipv4

seraddr.sin\_port=htons(SERPORT); //转换字节序为大端模式 man 3 byteorder

seraddr.sin\_addr.s\_addr=inet\_addr(SERADDR);//设置ip地址

bind(sockfd,(const struct sockaddr \*)&seraddr,sizeof(seraddr));

if(ret<0){

perror("bind");

return -1;

}

printf("bind sucess.\n");

//第三步，listen监听端口

ret=listen(sockfd,BACKLOG);

if(ret<0){

perror("listen");

return -1;

}

//第四部，accept阻塞等待

chifd=accept(sockfd,(struct sockaddr \*)&cliaddr,&len);

printf(连接已经建立，client fd=%d.\n",chifd);

/\*

//建立连接以后就可以通信了

//服务器收

ret=recv(clifd,recvbuf,sizeof(recvbuf),0);

printf("成功接受了%d个字符.\n",ret);

printf("client 发送过来的内容是：%s",recvbuf);

\*/

/\*

//服务器反复收

while(1){

ret=recv(clifd,recvbuf,sizeof(recvbuf),0);

printf("成功接受了%d个字符.\n",ret);

printf("client 发送过来的内容是：%s\n",recvbuf);

memset(recvbuf,0,sizeof(recvbuf));

}

\*/

//服务器发

printf("请输入要发送的内容\n");

scanf("%s",recvbuf);

printf("刚才输入的是%s.\n",recvbuf);

ret=send(clifd,recvbuf,strlen(recvbuf),0);

return 0;

}

### client.c

#include<stdio.h>

#include<sys/socket.h>

#include<sys/types.h>

#include<arpa/inet.h>

#include<string.h>

#define SERPORT 6003

#define SERADDR "140.82.22.121" //ifconfig 查看服务器ip地址

char sendbuf[100];

int main(){

//第一步先socket打开文件描述符

int sockfd=-1,ret=-1;

struct sockaddr\_in seraddr={0};

socklen\_t len=0;

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

if(-1==sockfd){

perror("socket");

return -1;

}

printf("socketfd=%d.\n",sockfd);

//第二步 通过connect连接服务器

seraddr.sin\_family=AF\_INET; //设置地址族为ipv4

seraddr.sin\_port=htons(SERPORT ); //转换字节序为大端模式 man 3 byteorder

seraddr.sin\_addr.s\_addr=inet\_addr(SERADDR);//设置ip地址

connect(sockfd,(const struct sockaddr\*) &seraddr,sizeof(seraddr));

if(ret<0){

perror("connect");

return -1;

}

printf("connect sucess.ret=%d.\n",ret);

/\*

//建立连接之后就可以开始通信了

//客户端发

strcpy(sendbuf,"hello world.");

ret=send(sockfd,sendbuf,strlen(sendbuf),0);

printf("发送了%d个字符\n",ret);

\*/

/\*

//客户端反复发

while(1){

printf("请输入要发送的内容\n");

scanf("%s",sendbuf);

printf("刚才输入的是%s.\n",sendbuf);

ret=send(sockfd,sendbuf,strlen(sendbuf),0);

}

\*/

//客户端收

ret=recv(sockfd,sendbuf,sizeof(sendbuf),0);

printf("成功接受了%d个字符.\n",ret);

printf("client 发送过来的内容是：%s\n",sendbuf);

memset(sendbuf,0,sizeof(sendbuf));

return 0;

}

客户端发送&服务器接收

服务器发送&客户端接收

探讨：如何让服务器和客户端好好沟通

1. 客户端和服务器原则上都是可以任意的发和收，但是实际上就是双方必须配合：client发的时候server就收，而server发的时候client就收。
2. 必须了解到一点：client和server之间的通信是异步的。这就是问题的根源
3. 解决方案：依靠应用层协议来解决。说白了就是client和server之间事先做好一系列的通信约定。

socket编程实践4

自定义应用层协议第一步：规定发送和接收方法

1. 规定连接建立后客户端主动向服务器发送一个数据包，然后服务器收到数据包后回复客户端一个数据包，这就是一个通信回合。
2. 整个连接的通信就是由N多个会和组成的

自定义应用层协议第二步：定义数据包格式

常用应用层协议：http、ftp.......

### server.c

#include<stdio.h>

#include<sys/socket.h>

#include<sys/types.h>

#include<arpa/inet.h>

#include<string.h>

#define SERPORT 6003

#define BACKLOG 100//负载队列

#define SERADDR "140.82.22.121" //ifconfig 查看自己的ip地址

#define CMD\_REGISTER 1001 //注册学生信息

#define CMD\_CHK 1002 //检测学生信息

#define CMD\_GETINFO 1003 //获取学生信息

#define STAT\_OK 30 //回复ok

#define STAT\_ERR 31 //回复出错了

typedef struct commu{

char name[20]; //学生姓名

int age; //学生年龄

int cmd; //命令码

int stat; //状态信息，用来回复

}info;

char recvbuf[100];

int main(){

//第一步先socket打开文件描述符

int sockfd=-1,ret=-1,clifd=-1;

struct sockaddr\_in seraddr={0};

struct sockaddr\_in cliaddr={0};

socklen\_t len=0;

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

if(-1==sockfd){

perror("socket");

return -1;

}

printf("socketfd=%d.\n",sockfd);

//第二步 bind绑定sockfd和当前电脑的ip地址和端口号

seraddr.sin\_family=AF\_INET; //设置地址族为ipv4

seraddr.sin\_port=htons(SERPORT); //转换字节序为大端模式 man 3 byteorder

seraddr.sin\_addr.s\_addr=inet\_addr(SERADDR);//设置ip地址

ret=bind(sockfd,(const struct sockaddr \*)&seraddr,sizeof(seraddr));

if(ret<0){

perror("bind");

return -1;

}

printf("bind sucess.\n");

//第三步，listen监听端口

ret=listen(sockfd,BACKLOG);

if(ret<0){

perror("listen");

return -1;

}

//第四部，accept阻塞等待

clifd=accept(sockfd,(struct sockaddr \*)&cliaddr,&len);

printf("连接已经建立，client fd=%d.\n",clifd);

while(1){

info stu;

//回合第一步：服务器收

memset(&stu,0,sizeof(info));

ret=recv(clifd,&stu,sizeof(info),0);

/\*

//printf("成功接受了%d个字符.\n",ret);

printf("client 发送过来的内容是：%s\n",recvbuf);

memset(recvbuf,0,sizeof(recvbuf));\*/

//回合中第二步：服务器解析客户端数据包，然后干活

if(stu.cmd==CMD\_REGISTER){

printf("用户要注册学生信息.\n");

printf("学生姓名：%s,学生年龄：%d\n.",stu.name,stu.age);

//回合中第三步：恢复客户端

stu.stat=STAT\_OK;

ret=send(clifd,&stu,sizeof(info),0);

}

if(stu.cmd==CMD\_CHK){

}

if(stu.cmd==CMD\_GETINFO){

}

}

return 0;

}

### client.c

#include<stdio.h>

#include<sys/socket.h>

#include<sys/types.h>

#include<arpa/inet.h>

#include<string.h>

#define SERPORT 6003

#define SERADDR "140.82.22.121" //ifconfig 查看服务器ip地址

char sendbuf[100];

char recvbuf[100];

#define CMD\_REGISTER 1001 //注册学生信息

#define CMD\_CHK 1002 //检测学生信息

#define CMD\_GETINFO 1003 //获取学生信息

#define STAT\_OK 30 //回复ok

#define STAT\_ERR 31 //回复出错了

typedef struct commu{

char name[20]; //学生姓名

int age; //学生年龄

int cmd; //命令码

int stat; //状态信息，用来回复

}info;

int main(){

//第一步先socket打开文件描述符

int sockfd=-1,ret=-1;

struct sockaddr\_in seraddr={0};

socklen\_t len=0;

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

if(-1==sockfd){

perror("socket");

return -1;

}

printf("socketfd=%d.\n",sockfd);

//第二步 通过connect连接服务器

seraddr.sin\_family=AF\_INET; //设置地址族为ipv4

seraddr.sin\_port=htons(SERPORT ); //转换字节序为大端模式 man 3 byteorder

seraddr.sin\_addr.s\_addr=inet\_addr(SERADDR);//设置ip地址

ret=connect(sockfd,(const struct sockaddr\*) &seraddr,sizeof(seraddr));

if(ret<0){

perror("connect");

return -1;

}

printf("connect sucess.ret=%d.\n",ret);

while(1){

//回合中第一步：客户端给服务器发送信息

info stu1;

printf("请输入学生姓名\n");

scanf("%s",&stu1.name);

printf("请输入学生年龄.\n");

scanf("%d",&stu1.age);

stu1.cmd=CMD\_REGISTER;

ret=send(sockfd,&stu1,sizeof(info),0);

printf("发送了1条学生信息\n");

//回合中第二步：客户端接收服务器的回复

memset(&stu1,0,sizeof(info));

ret=recv(sockfd,&stu1,sizeof(info),0);

if(stu1.stat==STAT\_OK){

printf("注册学生信息成功.\n");

}else{

printf("注册学生信息失败.\n");

}

//回合中第三步：客户端解析服务器的回复，再做下一步定夺。

}

return 0;

}