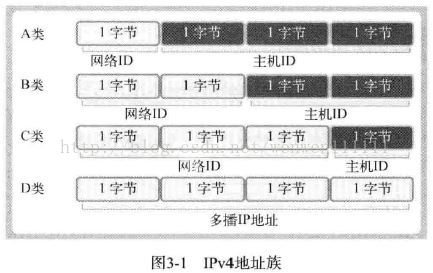
IP地址族

Ipv4 4字节地址族

Ipv6 16字节地址族

Ipv4标准的4字节ip地址分为网络地址和主机地址，，分成abcde等类型



A类地址的首字节范围：0-127

B类地址的首字节范围：128-191

C类地址的首字节范围：192-223

另一种表达方法：

A类地址的首位以0开始

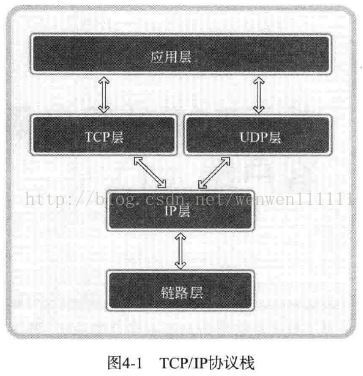
B类地址的前2位以10开始

C类地址的前3位以110开始

1. 理解TCP和UDP

根据数据传输方式的不同，基于网络协议的套接字一般分成TCP套接字和UDP套接字。

TCP套接字是面向连接的，又称为基于流Stream的套接字，TCP（Transmission Control Protocol 传输控制协议）的简写，书中将TCP/IP协议栈分成4层架构，掌握这4层协议栈就足够了。另外不同于OSI 7层架构：物理层-->数据链路层-->网络层-->传输层-->会话层-->表示层-->应用层。



1. 链路层：是物理链接领域标准化的结果，也是最基本的领域，专门定义LAN，WAN，MAN等网络标准。

2. IP层：复杂网络中，负责路径的选择。IP本身是面向消息的，不可靠的协议。

3. TCP/UDP层：已IP层提供的路劲信息为基础完成实际的数据传输，因此也称为传输层。TCP比UDP复杂，TCP可以保证可靠的数据传输。

4. 应用层：根据程序特点决定服务器端和客户端之间数据传输规则。

1. 实现基于TCP的服务器端和客户端



1. #include <sys/socket.h>
3. **int** socket(**int** domain, **int** type, **int** protocol);
5. 成功时返回文件描述符，失败时返回-1.
7. domain 套接字中使用的协议簇信息；
8. type 套接字数据传输类型信息；
9. protocol 计算机间通信使用的协议信息。
10. #include <sys/socket.h>
12. **int** bind(**int** sockfd, **struct** sockaddr \*myaddr, socklen\_t addrlen);
14. 成功时返回0，失败时返回-1
16. sockfd 要分配地址信息的套接字文件描述符
17. myaddr 存有地址信息的结构体变量地址值
18. addrlen 第二个结构体变量的长度
19. #include <sys/socket.h>
21. **int** listen(**int** sock, **int** backlog);
23. 成功时返回0，失败时返回-1
25. sock 希望进入等待连接请求状态的套接字文件描述符，传递的描述符套接字参数成为服务器端套接字
26. backlog 连接请求等待队列长度，若为5，则队列长度为5，表示最多5个连接请求进入队列。
27. #include <sys/socket.h>
29. **int** accept(**int** sock, **struct** sockaddr \*addr, socklen\_t \*addrlen);
31. 成功时返回创建的套接字文件描述符，失败时返回-1
33. sock 服务器套接字的文件描述符
34. addr保存发起连接请求的客户端地址信息的变量地址值，函数调用后传递来的地址变量参数填充客户端地址信息
35. addrlen 第二个参数addr结构体的长度
36. #include <sys/socket.h>
38. **int** connect (**int** sock, **struct** sockaddr \*servaddr, socklen\_t addrlen);
40. 成功时返回0，失败时返回-1
42. sock 客户端套接字文件描述符
43. servaddr 保存目标服务器端地址信息的变量地址值
44. addrlen 以字节为单位传递已传递给第二个结构体参数servaddr的地址变量长度
45. 迭代回声服务器端、客户端

服务器端在同一时刻只与一个客户端相连，并提供回声服务；

服务器端依次向5个客户端提供服务并退出

客户端接收用户输入的字符串并发送给服务器

服务器端将接收到的字符串传回客户端

服务器与客户端之间的字符串回声一直到执行到客户端输入Q为止。

Server端代码：

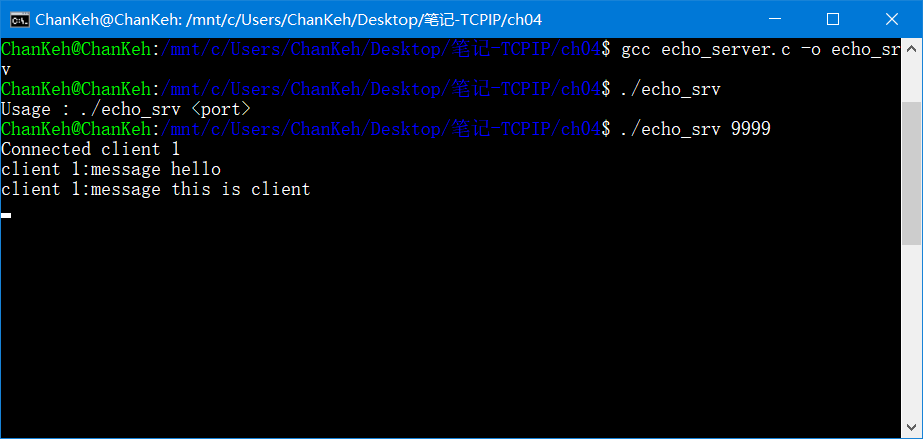
1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <string.h>
4. #include <unistd.h>
5. #include <arpa/inet.h>
6. #include <sys/socket.h>
8. #define BUF\_SIZE    1024
9. **void** error\_handling(**char** \*message);
11. **int** main(**int** argc, **char** \*argv[]){
12. **int** serv\_sock;
13. **int** clnt\_sock;
15. **char** message[BUF\_SIZE];
16. **int** str\_len,i;
17. **struct** sockaddr\_in serv\_addr;
18. **struct** sockaddr\_in clnt\_addr;
19. socklen\_t clnt\_addr\_size;
21. **if**(argc != 2){
22. printf("Usage : %s <port>\n",argv[0]);
23. exit(1);
24. }
25. serv\_sock = socket(PF\_INET,SOCK\_STREAM,0);
26. **if**(serv\_sock == -1){
27. error\_handling("socket() error");
28. }
30. memset(&serv\_addr,0,**sizeof**(serv\_addr));
31. serv\_addr.sin\_family=AF\_INET;
32. serv\_addr.sin\_addr.s\_addr=htonl(INADDR\_ANY);
33. serv\_addr.sin\_port=htons(atoi(argv[1]));
35. **if**(bind(serv\_sock,(**struct** sockaddr\*)&serv\_addr,**sizeof**(serv\_addr)) == -1){
36. error\_handling("bind() error");
37. }
39. **if**(listen(serv\_sock,5) == -1){
40. error\_handling("listen() error");
41. }
43. clnt\_addr\_size = **sizeof**(clnt\_addr);
44. **for**(i=0;i<5;i++){
45. clnt\_sock = accept(serv\_sock,(**struct** sockaddr \*)&clnt\_addr,&clnt\_addr\_size);
46. **if**(clnt\_sock == -1){
47. error\_handling("accept() error");
48. }**else**{
49. printf("Connected client %d\n",i+1);
50. }
51. **while**((str\_len = read(clnt\_sock,message,BUF\_SIZE)) != 0)
52. {
53. write(clnt\_sock,message,str\_len);
54. printf("client %d:message %s",i+1,message);
55. }
56. close(clnt\_sock);
57. }
58. close(serv\_sock);
60. **return** 0;
62. }
64. **void** error\_handling(**char** \*message){
66. fputs(message,stderr);
67. fputs("\n",stderr);
68. exit(1);
69. }

Client端代码：

1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <string.h>
4. #include <unistd.h>
5. #include <arpa/inet.h>
6. #include <sys/socket.h>
8. #define BUF\_SIZE    1024
9. **void** error\_handling(**char** \*message);
11. **int** main(**int** argc,**char** \*argv[]){
12. **int** sock;
13. **struct** sockaddr\_in serv\_addr;
14. **char** message[BUF\_SIZE];
15. **int** str\_len;
17. **if**(argc != 3){
18. printf("Usage : %s <IP> <port>\n",argv[0]);
19. exit(1);
20. }
22. sock = socket(PF\_INET,SOCK\_STREAM,0);
23. **if**(sock == -1){
24. error\_handling("socket() error");
25. }
27. memset(&serv\_addr,0,**sizeof**(serv\_addr));
28. serv\_addr.sin\_family = AF\_INET;
29. serv\_addr.sin\_addr.s\_addr = inet\_addr(argv[1]);
30. serv\_addr.sin\_port = htons(atoi(argv[2]));
32. **if**(connect(sock,(**struct** sockaddr \*)&serv\_addr,**sizeof**(serv\_addr)) == -1){
33. error\_handling("connect() error\r\n");
34. }**else**{
35. printf("Connected....");
36. }
38. **while**(1){
39. printf("Input message(Q to quit):");
40. fgets(message,BUF\_SIZE,stdin);
42. **if**(!strcmp(message,"q\n") || !strcmp(message,"Q\n"))
43. **break**;
44. write(sock,message,strlen(message));
45. str\_len = read(sock,message,BUF\_SIZE-1);
46. message[str\_len] = 0;
48. printf("Message from server : %s \n",message);
49. }
50. close(sock);
51. **return** 0;
52. }
54. **void** error\_handling(**char** \*message){
55. fputs(message,stderr);
56. fputs("\n",stderr);
57. exit(1);
58. }

执行结果：

服务端



客户端

