TCP和UDP

TCP与UDP基本区别

1.基于连接与无连接
2.TCP要求系统资源较多，UDP较少；
3.UDP程序结构较简单
4.流模式（TCP）与数据报模式(UDP);
5.TCP保证数据正确性，UDP可能丢包
6.TCP保证数据顺序，UDP不保证

UDP应用场景：

1.面向数据报方式
2.网络数据大多为短消息
3.拥有大量Client
4.对数据安全性无特殊要求
5.网络负担非常重，但对响应速度要求高

TCP的应用场景：

应用层协议	应用	传输层协议
SMTP	电子邮件	TCP
TELNET	远程终端接入	
HTTP	万维网	
FTP	文件传输	
DNS	名字转换	UDP
TFTP	文件传输	
RIP	路由选择协议	
BOOTP，DHCP	IP 地址配置	
SNMP	网络管理	
NFS	远程文件服务器	
专用协议	IP 电话	
专用协议	流式多媒体通信	

其次的区别在于编程方面：

具体编程时的区别

1.socket()的参数不同
2.UDP Server不需要调用listen和accept
3.UDP收发数据用sendto/recvfrom函数
4.TCP：地址信息在connect/accept时确定
5.UDP：在sendto/recvfrom函数中每次均 需指定地址信息
6.UDP：shutdown函数无效.

TCP:

TCP编程的服务器端一般步骤是：

1、创建一个socket，用函数socket()；
2、设置socket属性，用函数setsockopt(); * 可选
3、绑定IP地址、端口等信息到socket上，用函数bind();
4、开启监听，用函数listen()；
5、接收客户端上来的连接，用函数accept()；
6、收发数据，用函数send()和recv()，或者read()和write();
7、关闭网络连接；
8、关闭监听；

TCP编程的客户端一般步骤是：

1、创建一个socket，用函数socket()；
2、设置socket属性，用函数setsockopt();* 可选
3、绑定IP地址、端口等信息到socket上，用函数bind();* 可选
4、设置要连接的对方的IP地址和端口等属性；
5、连接服务器，用函数connect()；
6、收发数据，用函数send()和recv()，或者read()和write();
7、关闭网络连接；

****

UDP:
与之对应的UDP编程步骤要简单许多，分别如下：

UDP编程的服务器端一般步骤是：

1、创建一个socket，用函数socket()；
2、设置socket属性，用函数setsockopt();* 可选
3、绑定IP地址、端口等信息到socket上，用函数bind();
4、循环接收数据，用函数recvfrom();
5、关闭网络连接；

UDP编程的客户端一般步骤是：

1、创建一个socket，用函数socket()；
2、设置socket属性，用函数setsockopt();* 可选
3、绑定IP地址、端口等信息到socket上，用函数bind();* 可选
4、设置对方的IP地址和端口等属性;
5、发送数据，用函数sendto();
6、关闭网络连接；

```
ssize_t sendto(int sockfd, const void *buf, size_t len, int flags,
               const struct sockaddr *dest_addr, socklen_t addrlen);
```

```
ssize_t recvfrom(int sockfd, void *buf, size_t len, int flags,
                 struct sockaddr *src_addr, socklen_t *addrlen);
```

recv是拥塞函数，没有数据就一直拥塞。

TCP与UDP区别总结：

1、TCP面向连接（如打电话要先拨号建立连接）;UDP是无连接的，即发送数据之前不需要建立连接
2、TCP提供可靠的服务。也就是说，通过TCP连接传送的数据，无差错，不丢失，不重复，且按序到达;UDP尽最大努力交付，即不保　证可靠交付
3、TCP面向字节流，实际上是TCP把数据看成一连串无结构的字节流;UDP是面向报文的
　UDP没有拥塞控制，因此网络出现拥塞不会使源主机的发送速率降低（对实时应用很有用，如IP电话，实时视频会议等）

4、每一条TCP连接只能是点到点的;UDP支持一对一，一对多，多对一和多对多的交互通信
5、TCP首部开销20字节;UDP的首部开销小，只有8个字节
6、TCP的逻辑通信信道是全双工的可靠信道，UDP则是不可靠信道

各自的优缺点;

1.tcp优缺点：

可靠，稳定，面向连接，有流量控制，窗口，重传，拥塞控制

tcp缺点：

慢，效率低，占用的资源多，易被攻击

2.udp的优点：

快，比tcp安全一些（但还是会别攻击 udp flood）

UDP没有拥塞控制，因此网络出现拥塞不会使源主机的发送速率降低

开销小。

udp缺点：不可靠，不稳定

相比于tcp没有那些可靠得协议支持，所以当信号不好时，很容易丢包，完全由上游控制其可靠性。

一般是udp为主，tcp为辅、

如果udp丢包了，怎么办？udp乱序了，怎么办？

udp丢包和乱序都是很正常的，因为它本来就是不安全的。

udp丢包是因为udp没有流量控制，所以发送速度可能要比接受速度快，这样一来，接收端还没来得及处理完数据，发送端就已经发完了，剩下没进入缓冲区的那一部分就丢掉。

解决方法：

1.降低发送的速度，可能网络性能导致的丢包

2.设置setsockopt将缓冲区放大一点。

3.增加一个响应报文，收到响应报文后，再进行继续发送，有点类似有tcp的ack

4，至于乱序问题，对数据报进行编号

这只是一些简单方面的，足够应对面试，但要问到更深层次。。。。

```
void handle_udp_msg(int fd)
{
    char buf[BUFF_LEN];  //接收缓冲区，1024字节
    socklen_t len;
   int count;
    struct sockaddr_in clent_addr;  //clent_addr用于记录发送方的地址信息
   while(1)
   {
18        memset(buf, 0, BUFF_LEN);
19        len = sizeof(clent_addr);
```

```
20        count = recvfrom(fd, buf, BUFF_LEN, 0, (struct sockaddr*)&clent_addr, &len);  //recvfrom是拥塞函数，没有数据就
21        if(count == -1)
22        {
23            printf("recieve data fail!\n");
24            return;
25        }
26        printf("client:%s\n",buf);  //打印client发过来的信息
27        memset(buf, 0, BUFF_LEN);
28        sprintf(buf, "I have recieved %d bytes data!\n", count);  //回复client
29        printf("server:%s\n",buf);  //打印自己发送的信息给
30        sendto(fd, buf, BUFF_LEN, 0, (struct sockaddr*)&clent_addr, len);  //发送信息给client，注意使用了clent_addr结构
31
32    }
33 }
34
36 /*
37    server:
38            socket-->bind-->recvfrom-->sendto-->close
39 */
41 int main(int argc, char* argv[])
42 {
43    int server_fd, ret;
44    struct sockaddr_in ser_addr;
45
46    server_fd = socket(AF_INET, SOCK_DGRAM, 0); //AF_INET:IPV4;SOCK_DGRAM:UDP
47    if(server_fd < 0)
48    {
49        printf("create socket fail!\n");
50        return -1;
51    }
52
53    memset(&ser_addr, 0, sizeof(ser_addr));
54    ser_addr.sin_family = AF_INET;
55    ser_addr.sin_addr.s_addr = htonl(INADDR_ANY); //IP地址，需要进行网络序转换，INADDR_ANY：本地地址
56    ser_addr.sin_port = htons(SERVER_PORT);  //端口号，需要网络序转换
57
58    ret = bind(server_fd, (struct sockaddr*)&ser_addr, sizeof(ser_addr));
59    if(ret < 0)
60    {
61        printf("socket bind fail!\n");
62        return -1;
63    }
64
65    handle_udp_msg(server_fd);   //处理接收到的数据
66
67    close(server_fd);
68    return 0;
69 }
```