服务器

[TCP/IP 协议族 3](#_Toc32855718)

[TCP协议 3](#_Toc32855719)

[建立连接 4](#_Toc32855720)

[关闭连接 4](#_Toc32855721)

[半关闭 4](#_Toc32855722)

[TCP 状态转换图 5](#_Toc32855723)

[TCP 连接分组交换 6](#_Toc32855724)

[TCP 超时重传 7](#_Toc32855725)

[TCP 拥塞控制 7](#_Toc32855726)

[IP 协议 7](#_Toc32855727)

[套接字选项 8](#_Toc32855728)

[网络API 8](#_Toc32855729)

[大小端转换 8](#_Toc32855730)

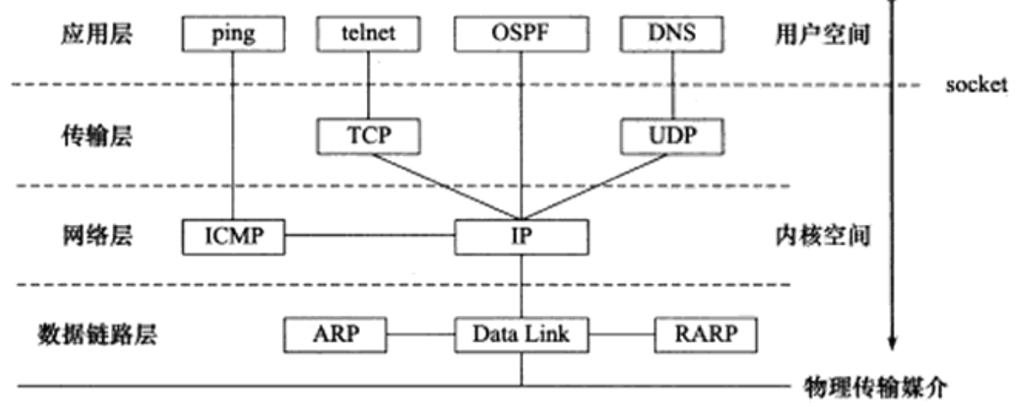
[socket 地址结构 8](#_Toc32855731)

[IP 地址转换函数 9](#_Toc32855732)

[通信建立函数 9](#_Toc32855733)

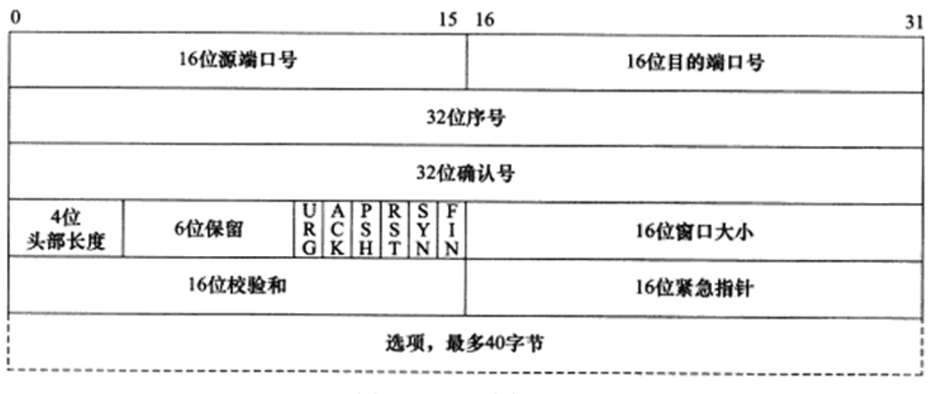
[信号机制 10](#_Toc32855734)

### TCP/IP 协议族



#### TCP协议

面向连接、字节流、可靠传输。



URG 紧急报文

用于发送带外数据（OOB），OOB比普通数据有更高优先级。

ACK 同步报文

PSH 数据及时读取提示

Server每发4个TCP报文，就会发送一个PSH标志给client，通知其尽快读取数据。

RST 复位报文

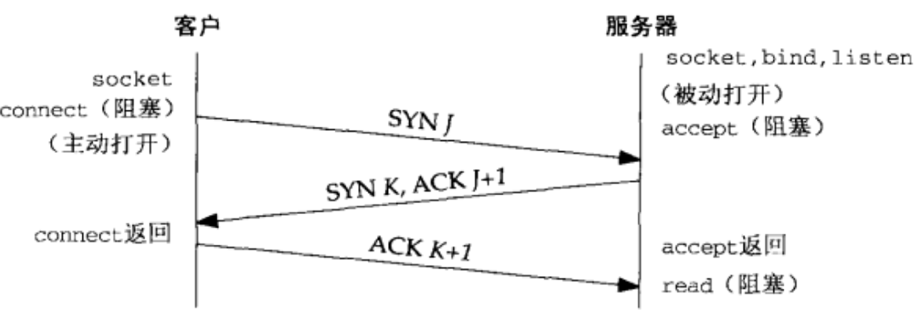
访问不存才的端口，client 访问不存在的端口，或者端口占用的连接处于TIME\_WAIT状态，server端将发送RST报文，

异常终止连接，通过发送RST终止一个连接。

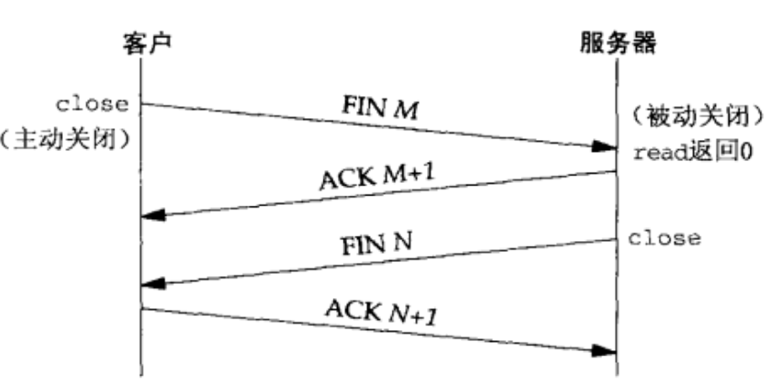
处理半打开连接。

FIN 终止报文

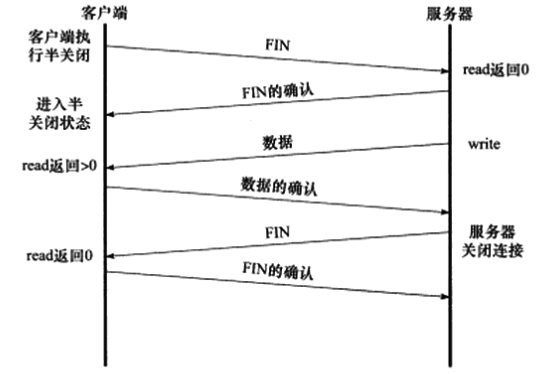
##### 建立连接



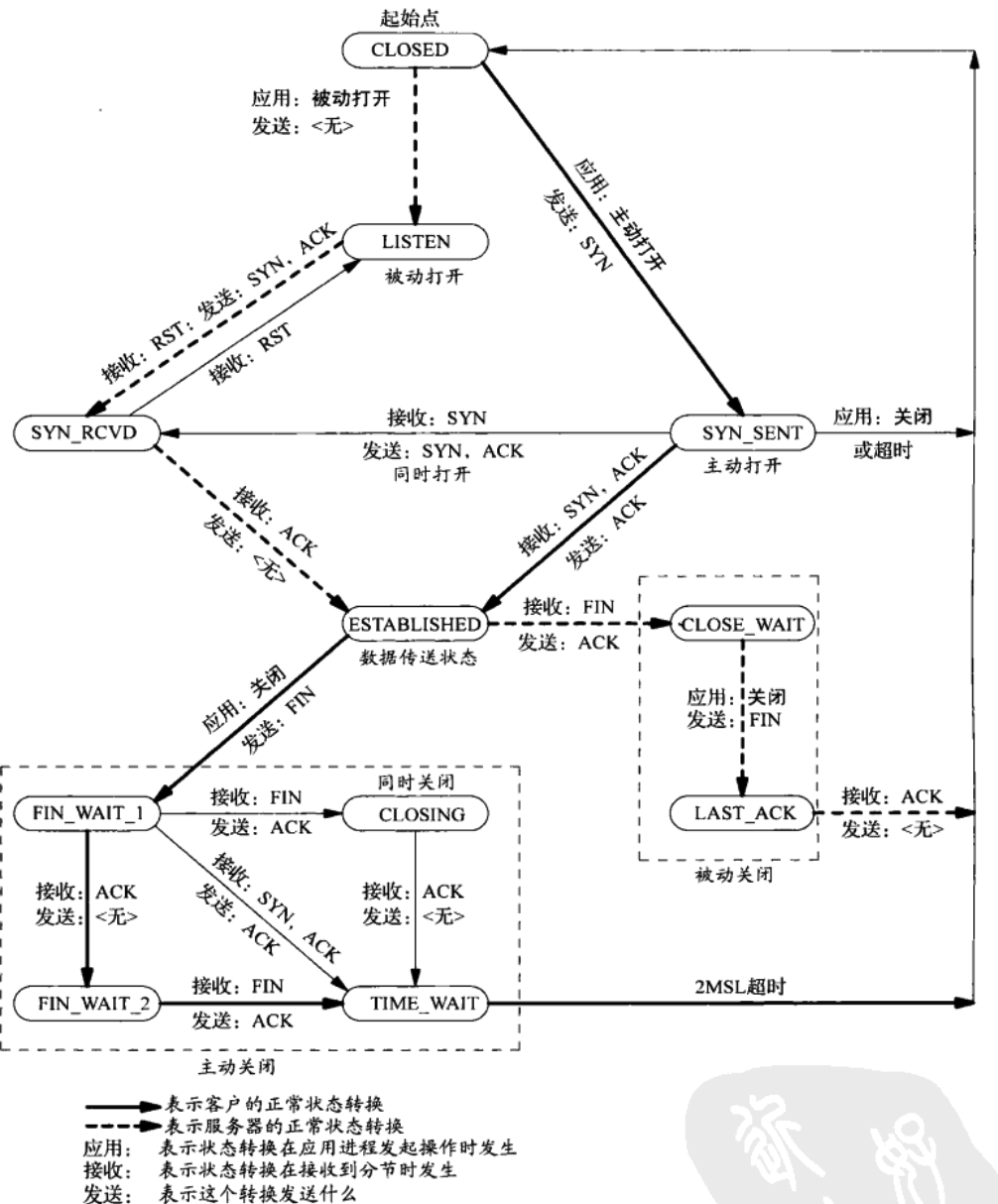
##### 关闭连接



##### 半关闭



##### TCP 状态转换图

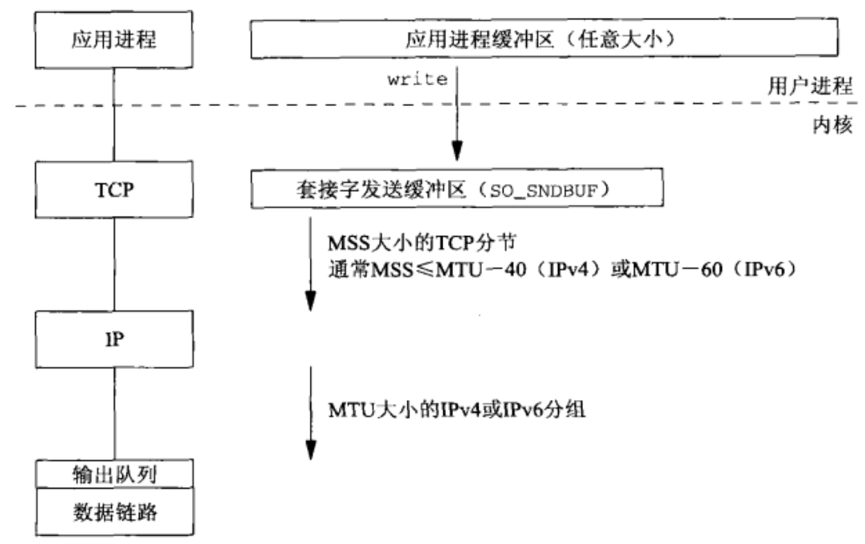


##### TCP 连接分组交换

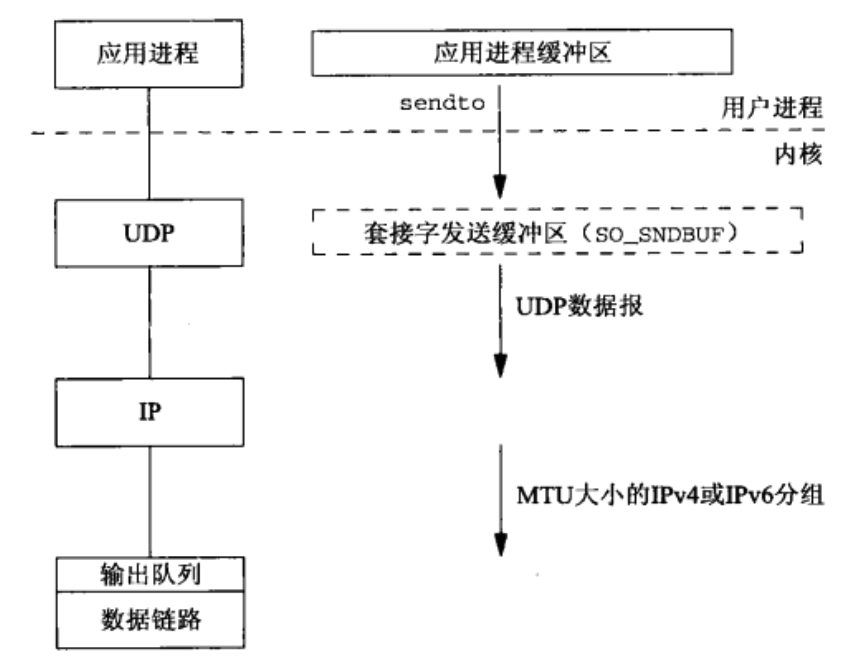


连接停留在FIN\_WAIT2的状态，可能发生在客户端执行半关闭的时候，未等服务器关闭连接就强制退出了。此时客户端连接由内核来接管。涉及内核参数/proc/sys/net/ipv4/tcp\_max\_orphan（连接数），/proc/sys/net/ipv4/tcp\_fin\_timeout（存活时间）。

应用进程写TCP套接字



应用进程写UDP 套接字



##### TCP 超时重传

/proc/sys/net/ipv4/tcp\_retries1 // 最少重试次数

/proc/sys/net/ipv4/tcp\_retries2 // 连接断开前最多重试次数

##### TCP 拥塞控制

/proc/sys/net/ipv4/tcp\_congestion\_control 控制算法。

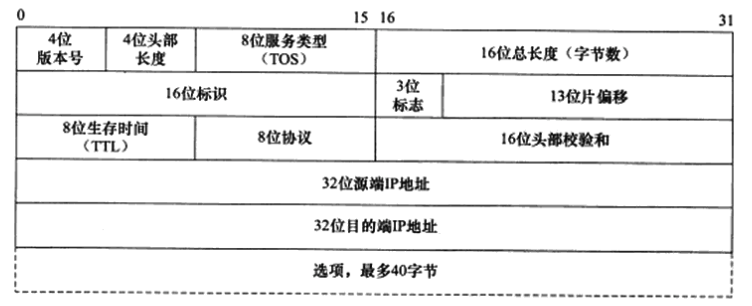
传输超时使用 慢启动和拥塞避免；

接收到重复的确认报文（连续收到3个重复报文）使用快速重传

和快速恢复

#### IP 协议

提供无连接、无状态、不可靠的服务。

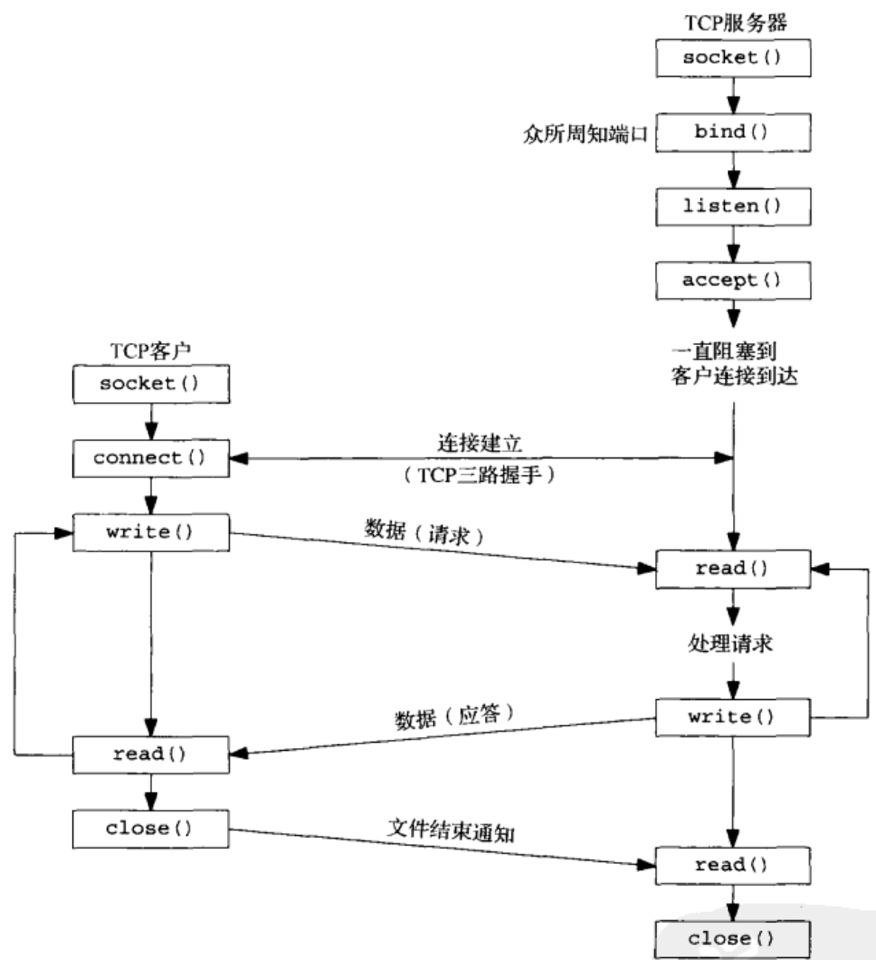


TOS（Type Of Service），包括三位优先权和1位保留位（置为0），4位TOS字段可为最小延时、最大吞吐量、最高可靠性、最小费用。

#### 套接字选项

SO\_REUSEADDR自己对于其理解就是，服务器在断开连接后，允许在time\_wait存在的情况下，给服务器分配一个其他的本机IP，而不是使用原有IP，再绑定相同的端口，实现TCP的成功连接。

### 网络API

****

**基本client/server 套接字函数**

#### 大小端转换

大端称为网络字节序，小段称为主机字节序。

Linux 提供4个函数来完成网络字节序和主机字节序的转换。

1. #include<netinet/in.h>
2. unsigned **long** **int** htonl(unsigned **long** **int** hostlong);
3. unsigned **short** **int** htons(unsigned **short** **int** hostshort);
4. unsigned **long** **int** ntohl(unsigned **long** **int** netlong);
5. unsigned **short** **int** ntohs(unsgined **short** **int** netshort);

#### socket 地址结构

1. #include<bits/socket.h>
2. **struct** sockaddr
3. {
4. sa\_family\_t sa\_family;
5. **char** sa\_data[14];
6. };
8. **struct** sockaddr\_in
9. {
10. sa\_family\_t sin\_family;
11. u\_int16\_t sin\_port;
12. **struct** in\_addr sin\_addr;  //ipv4 地址结构
13. };
15. **struct** in\_addr
16. {
17. u\_int32\_t s\_addr; //网络字节序表示
18. };

在实际开发中，sockadd\_in 必须转换为sockaddr。

#### IP 地址转换函数

1. #include<arpa/inet.h>
2. in\_addr\_t inet\_addr(**const** **char** \* strptr);
3. //点十分进制字符串转为网络字节序，失败返回INADDR\_NONE
4. **int** inet\_aton(**const** **char** \*  cp,**struct** in\_addr \* inp);
5. //将转化结果存储于 inp,成功返回1，失败返回0a
6. **char** \* inet\_ntoa(**struct** in\_addr in);
7. // 该函数不可重入
8. int inet\_pton(int af, const char \* src, void \* dest);
9. //成功返回1，失败返回0, 支持ipv6
10. const char \* inet\_ntop(int af,const void \*src, char \* dest,socklen\_t cn);
11. //成功返回目标存储单元地址，失败返回null

#### 通信建立函数

1. #include<sys/socket.h>
2. **int** socket(nt domain, **int** type, **int** protocol);
3. //成功返回可用的socket 描述符，失败返回-1。
4. **int** bind(**int** sockfd, **const** **struct** sockaddr \* my\_addr, socklen\_t addrlen);
5. //成功返回0，失败返回-1，并设置errno。常见的errno是EACCES，EADDRINUSE。
6. **int** listen(**int** sockfd, **int** backlog);
7. //backlog 处于完全连接(ESTABLISED)状态的socket  上限。成功返回0，失败返回-1，并设置errno。
8. **int** connect(**int** sockfd, **const** **struct** aockaddr \* addr, socklen\_t addrlen);
9. //成功返回0，失败返回-1，并设置errno，若失败则该套接字不再可用，必须关闭。
10. //常见的errno是ETIMEOUT；
11. //EINPROGRESS，以非阻塞方式执行，代表连接进行中，可使用poll/select 判断是否可写，可写表示连接完成；
12. //ECONNREFUSED，收到RST报文，表示服务器端无等待连接进程。
13. **int** accept(**int** sockfd, **struct** sockaddr \* addr, socklen\_t \* addrlen);
14. //用于从已完成连接队列的对头返回新连接的socketfd，失败返回-1,，并设置errno。
15. //常见的errno是ECONNABORTED，accept 返回前终止连接，一般由内核重新调用accept；
16. //EAGAIN、EWOULDBLOCK，等待请求到来时，非阻塞模式下返回-1，并置为上述标志，阻塞模式下可使用poll/select 判断是否可读。

### 信号机制

Sleep 实现

sleep使进程一直挂起，直到满足一下两个条件之一：

1、睡眠时间到时；

2、调用进程捕捉到一个信号，并从该信号处理函数返回；

1. #include <signal.h>
3. **static** **void**
4. sig\_alrm(**int** signo)
5. {
7. }
9. unsigned **int**
10. sleep(unsigned **int** nsecs)
11. {
12. **struct** sigaction newact, oldact;
13. sigset\_t newmask, oldmask, suspmask;
14. unsigned **int** unslpt;
16. newact.sa\_handler = sig\_alrm;
17. sigemptyset(&newact.sa\_mask);
18. newact.sa\_flags = 0;
20. sigaction(SIGALRM, &act, &oldact);
22. sigemptyset(&newmask);
23. sigaddset(&newmask, SIGALRM);
25. **if**(sigprocmask(SIG\_BLOCK, &newmask, &oldmask) < 0)
26. err\_sys("sigprocmask(SIG\_BLOCK)");
28. alarm(nsecs);
29. suspmask = oldmask;
30. sigdelset(&suspmask, SIGALRM);
31. sigsuspend(&suspmask);
32. unslept = alrm(0);
33. sigaction(SIGALRM, &oldact, NULL);
34. sigprocmask(SIG\_SETMASK, &oldmask, NULL);
35. **return** unslept;
36. }