C++研发高频笔/面试题目  
**笔试：**  
笔试这一块想不到特别好说的，很多时候只是公司筛选人的第一步，分数高低代表意义不大（尤其是大公司，现身说法，我的腾讯笔试真的是呵呵哒），当然笔试是一个门槛，不过后面很可能就不用谈了。我笔试经验也不足，感觉笔试考察最重点的就是算法的熟悉程度，代码能力，我笔试成绩通常不太高，原因是我编程题速度比较慢，往往都做不完。

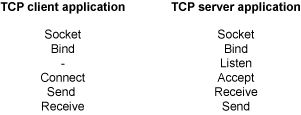
2计算机网络（TCP/IP）  
（1） 建立TCP服务器的各个系统调用：

计算机网络基础，socket编程，其中子网划分，七层协议（例如交换机属于数据链路层一类题目），五层协议，协议名称及其作用，常用端口号，https1.0 1.1特性与区别，三次握手四次挥手（可能出现的服务器攻击），加密（对称，非对称），IO复用等

典型的 TCP 客户机和服务器应用程序通过发布 TCP 系统调用序列来获取某些函数。这些系统调用包括socket ()、bind ()、listen ()、accept ()、

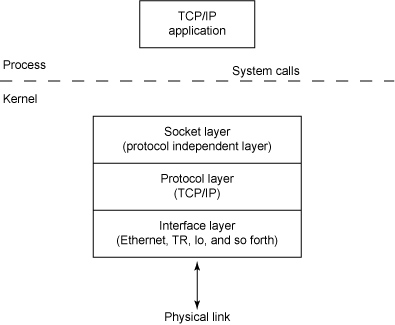
send () 和 receive()。本文介绍在应用程序发布 TCP 系统调用时在较低级别中发生的情况，如下图 所示。

图 1. TCP 应用程序进行的普通调用序列



[图 2](http://www.ibm.com/developerworks/cn/aix/library/au-tcpsystemcalls/#fig2) 显示了 TCP 系统调用在物理链路上发出之前进行传播的各个层。

图 2. TCP 系统调用的各个层



套接字层接收进行的任何 TCP 系统调用。套接字层验证 TCP 应用程序传递的参数的正确性。这是一个独立于协议 的层，因为尚未将协议连接到调用中。

套接字层下面是协议层，该层包含协议的实际实现（本例中为 TCP）。当套接字层对协议层进行调用时，将确保对两个层之间共享的数据结构具有独占访问权限。这样做是为了避免任何数据结构损坏。

各种网络设备驱动程序在接口层运行，该层从物理链路接收数据，并向物理链路传输数据。

每个套接字具有一个套接字队列，并且每个接口具有一个用于数据通信的接口队列。不过，对于整个协议层，只有一个称为 IP 输入队列的协议队列。接口层通过此 IP 输入队列将数据输入到协议层。协议层使用相应的接口队列将数据输出到接口。

在本文中，将学习以下系统调用：

* [Socket](http://www.ibm.com/developerworks/cn/aix/library/au-tcpsystemcalls/#socket)
* [Bind](http://www.ibm.com/developerworks/cn/aix/library/au-tcpsystemcalls/#bind)
* [Listen](http://www.ibm.com/developerworks/cn/aix/library/au-tcpsystemcalls/#listen)
* [Accept](http://www.ibm.com/developerworks/cn/aix/library/au-tcpsystemcalls/#accept)
* [Connect](http://www.ibm.com/developerworks/cn/aix/library/au-tcpsystemcalls/#connect)
* [Shutdown](http://www.ibm.com/developerworks/cn/aix/library/au-tcpsystemcalls/#shutdown)
* [Close](http://www.ibm.com/developerworks/cn/aix/library/au-tcpsystemcalls/#close)
* [Send](http://www.ibm.com/developerworks/cn/aix/library/au-tcpsystemcalls/#send)
* [Receive](http://www.ibm.com/developerworks/cn/aix/library/au-tcpsystemcalls/#receive)

Socket

socket (struct proc \*p, struct socket\_args \*uap, int retval)

struct sock\_args

{

int domain,

int type,

int protocol;

};

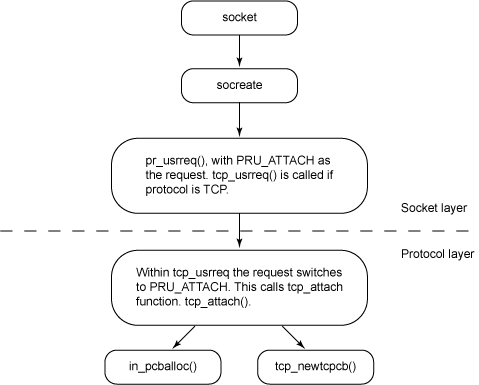
在 socket 系统调用中：

* p 是一个指针，指向进行 socket 调用的进程的 proc 结构。
* uap 是一个指向 socket\_args 结构的指针，该结构包含传递到 socket 系统调用中的进程的参数。
* retval 是系统调用的返回值。

socket 系统调用通过分配新的描述符创建新的套接字。将新的描述符返回到调用进程。任何后续的系统调用都使用创建的套接字标识。socket 系统调用还向创建的套接字描述符分配协议。

domain、type 和 protocol 参数值指定系列、类型和协议，以分配给创建的套接字。[图 3](http://www.ibm.com/developerworks/cn/aix/library/au-tcpsystemcalls/#fig3) 显示了调用序列。

图 3. 用于 socket 系统调用的调用序列



从进程检索参数后，socket 函数调用 socreate 函数。socreate 函数根据进程指定的参数发现指向协议切换 protsw 结构的指针。socreate 函数然后分配新的套接字结构。然后进行协议特定的调用 pr\_usrreq，进而切换到与套接字描述符关联的相应协议特定的请求。pr\_usrreq 函数的原型为：

int pr\_usrreq(struct socket \*so , int req, struct mbuf \*m0 , \*m1 , \*m2);

在 pr\_usrreq 函数中：

* so 是指向套接字结构的指针。
* req 的功能是标识请求。本例中为 PRU\_ATTACH。
* m0、m1 和 m2 是指向 mbuf 结构的指针。值因请求而异。

pr\_usrreq 函数为大约 16 个请求提供服务。

tcp\_usrreq() 函数调用 tcp\_attach( )，以处理 PRU\_ATTACH 请求。要分配 Internet 协议控制块，可调用 in\_pcballoc()。在in\_pcballoc 中，调用了内核的内存分配器函数，该函数将内存分配给 Internet 控制块。完成所有必要的 Internet 控制块结构指针初始化之后，该控制返回到 tcp\_attach()。

分配新的 TCP 控制块，并在 tcp\_newtcpcb() 中初始化。它还初始化所有的 TCP 定时器变量，并且控制返回到 tcp\_attach()。现在套接字状态初始化为 CLOSED。在返回到 tcp\_usrreq 函数时，创建套接字描述符，以指向套接字的 TCP 控制块。

Internet 控制块是双向链接的循环链表，其指针指向套接字结构，同时套接字结构的 so\_pcb 部分指向 Internet 控制块结构。Internet 控制块还具有指向 TCP 控制块的指针。

Bind

bind (struct proc \*p, struct bind\_args \*uap, int \*retval)

struct bind\_args

{ int s;

caddr\_t name;

int namelen;

};

在 bind 系统调用函数中：

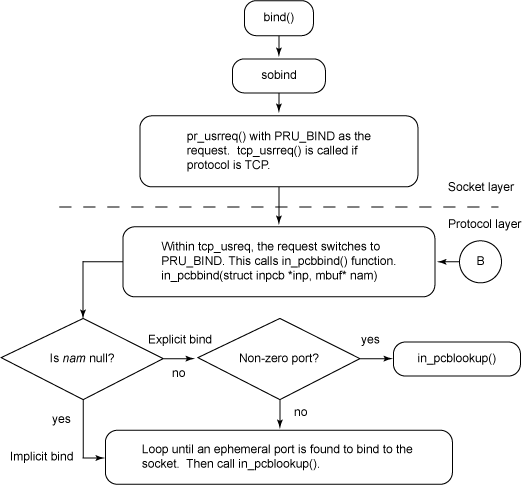
* s 是套接字描述符。
* name 是指向包含网络传输地址的缓冲区的指针。
* namelen 是缓冲区的大小。

bind 系统调用将本地网络传输地址与套接字关联。对于客户端进程，发布 bind 调用不是强制的。当客户端进程发布 [connect](http://www.ibm.com/developerworks/cn/aix/library/au-tcpsystemcalls/#connect) 系统调用时，内核负责执行隐式绑定。服务器进程接受连接或启动与客户端的通信之前，发布显式绑定请求通常是必需的。

bind 调用将进程指定的本地地址复制到 mbuf，并调用 sobind，后者则根据请求使用 PRU\_BIND 调用 tcp\_usrreq()。tcp\_usrreq() 中的切换实例调用 in\_pcbbind()，后者将本地地址和端口号绑定到套接字。in\_pcbbind 函数首先执行一些完整性检查，以确保不绑定套接字两次，并且至少一个接口分配了 IP 地址。in\_pcbbind 负责隐式和显式绑定。

如果对 in\_pcbbind()（指向 sockaddr\_in 结构的指针）的调用中的第二个参数为非空，则发生显式绑定。其他情况下，则发生隐式绑定。对于显式绑定，在绑定的 IP 地址上执行检查，并相应设置套接字选项。

图 4. 用于 bind 系统调用的调用序列



如果指定的本地端口是一个非零值，则对超级用户特权进行检查，以确定绑定是否位于保留的端口（例如，根据 Berkley 约定，端口号 < 1024）。然后调用 in\_pcblookup()，以便查找具有提到的本地 IP 地址和本地端口号的控制块。in\_pcblookup() 验证本地地址和端口对是否仍未使用。如果 in\_pcbbind() 中的第二个参数是 NULL，或本地端口是零，则控制失败，并检查临时端口（例如，根据 Berkley 约定，1024 < 端口号 < 5000）。然后调用 in\_pcblookup()，以验证发现的端口是否未使用。

Listen

listen (struct proc \*p, struct listen\_args \*uap, int \*retval)

struct listen\_args

{ int s;

int backlog;

};

在 listen 系统调用中：

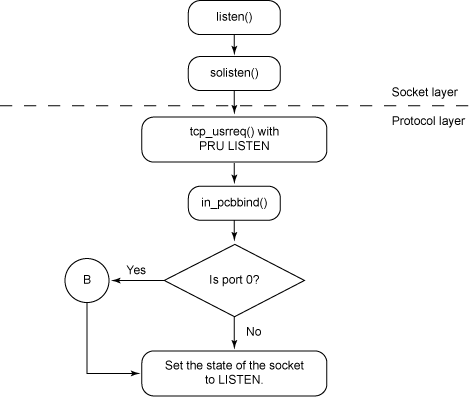
* s 是套接字描述符。
* backlog 是套接字上的连接数的队列限制。

listen 调用指示协议，服务器进程准备接受套接字上任何新传入的连接。存在一个可以排列的连接数限制，在该连接数之后，忽略任何进一步的连接请求。

listen 系统调用使用套接字描述符和 listen 调用中指定的backlog 值调用 solisten。solisten 仅使用 PRU\_LISTEN 作为请求调用tcp\_usrreq 函数。在 tcp\_usrreq() 函数的切换语句中，PRU\_LISTEN 的实例检查套接字是否绑定到端口。如果端口为零，则调用in\_pcbbind()，将套接字绑定到一个端口（按照 [Bind](http://www.ibm.com/developerworks/cn/aix/library/au-tcpsystemcalls/#bind) 部分中的描述）。

如果端口上已存在侦听的套接字，则将套接字的状态更改为 LISTEN。通常，所有的服务器进程都侦听众所周知的端口号。很少调用in\_pcbbind 来执行服务器进程的隐式绑定。[图 5](http://www.ibm.com/developerworks/cn/aix/library/au-tcpsystemcalls/#fig5) 显示了侦听的调用序列。

图 5. 用于 listen 系统调用的调用序列



Accept

accept(struct proc \*p, struct accept\_args \*uap, int \*retval);

struct accept\_args

{

int s;

caddr\_t name;

int \*anamelen;

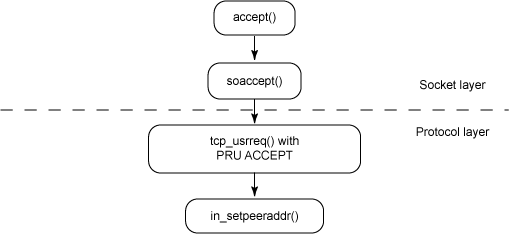
};

在 accept 系统调用中：

* s 是套接字描述符。
* name 是缓冲区（OUT 参数），它包含外来主机的网络传输地址。
* anamelen 是 name 缓冲区的大小。

accept 系统调用是等待传入连接的阻塞调用。处理连接请求后，accept 将返回新的套接字描述符。将此新的套接字连接到客户端，使另外一个套接字 s 保持 LISTEN 状态，以接受进一步连接。

图 6. 用于 accept 系统调用的调用序列



accept 调用首先验证参数，并等待要到达的连接请求。在此之前，函数在 while 循环中阻塞。新的连接到达后，协议层唤醒服务器进程。Accept 然后检查函数阻塞时发生的任何套接字错误。如果存在任何套接字错误，则函数返回，并继续从队列拾取新的连接并调用soaccept。在 soaccept() 中调用 tcp\_usrreq () 函数，并将请求作为 PRU\_ACCEPT。tcp\_usrreq 函数中的切换调用in\_setpeeraddr()，后者从协议控制块复制外来 IP 地址和外来端口号，并将其返回到服务器进程。

Connect

connect (struct proc \*p, struct connect\_args \*uap, int \*retval);

struct connect\_args

{

int s;

caddr\_t name;

int namelen;

};

在 connect 系统调用中：

* s 是套接字描述符。
* name 是指向具有外来 IP/端口地址对的缓冲区的指针。
* namelen 是缓冲区的长度。

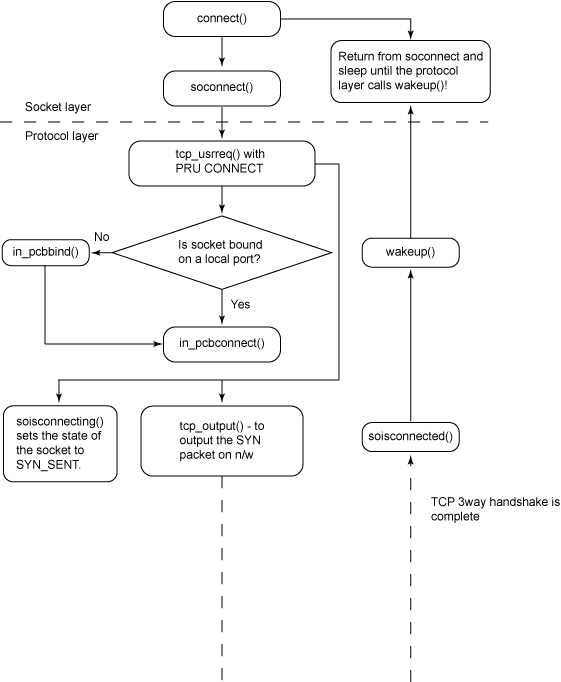
客户端进程通常调用 connect 系统调用，以连接到服务器进程。如果在初始化连接之前，客户端进程没有显式发布 bind 系统调用，则堆栈负责本地套接字上的隐式绑定。

connect 系统调用将外来地址（需要将连接请求发送到地址）从进程复制到内核，并调用 soconnect()。从 soconnect() 返回时，connect() 函数进入睡眠状体，直到协议层将其唤醒，并指示连接是 ESTABLISHED 或套接字上存在错误。soconnect() 函数检查套接字的有效状态，并使用 PRU\_CONNECT 作为请求调用 pr\_usrreq()。

tcp\_usrreq() 函数中的切换实例检查套接字与本地端口的绑定。如果未绑定套接字，则调用执行隐式绑定的 in\_pcbbind()。然后调用in\_pcbconnect()，以获取到达目的地的路线，发现必须输出套接字的接口，并验证 connect() 指定的外来套接字对（IP 地址和端口号）是否唯一。然后使用外来 IP 地址和端口号更新其 Internet 控制块，并返回到 PRU\_CONNECT 示例语句。

tcp\_usrreq () 现在调用 soisconnecting ()，它可以将客户端主机上的套接字的状态设置为 SYN\_SENT。调用函数 tcp\_output，将 SYN 包输出到网络。控制现在返回到 connect() 函数，该函数处于睡眠状态，直到协议层唤醒 — 指示连接现在是 ESTABLISHED，或套接字上存在错误。

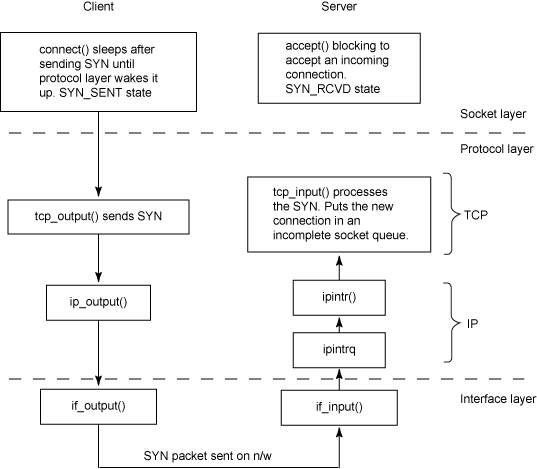
图 7. 用于 connect 系统调用的调用序列



3 向 TCP 握手

[图 8](http://www.ibm.com/developerworks/cn/aix/library/au-tcpsystemcalls/#fig8)、[图 9](http://www.ibm.com/developerworks/cn/aix/library/au-tcpsystemcalls/#fig9) 和[图 10](http://www.ibm.com/developerworks/cn/aix/library/au-tcpsystemcalls/#fig10) 显示了客户端发布 connect 和服务器发布 accept 以指示和建立 TCP 连接时的调用序列。

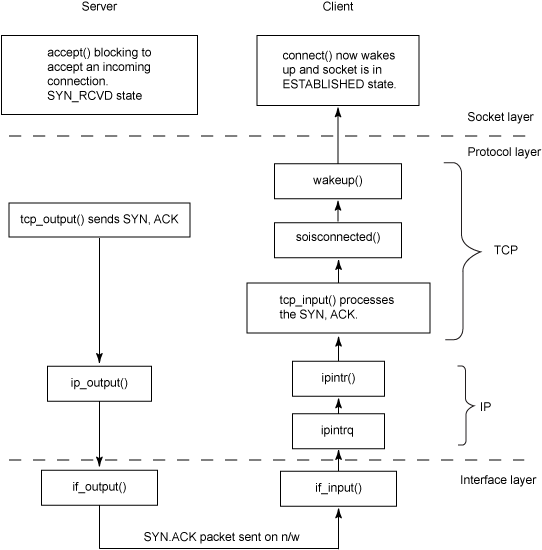
图 8. 用于 SYN 包的流序列



当客户端发布 connect 时，在协议层调用 tcp\_output() 函数，将 SYN 包输出到接口。如[图 9](http://www.ibm.com/developerworks/cn/aix/library/au-tcpsystemcalls/#fig9) 所示，soconnect 现在返回到 connect() 函数，并进入睡眠状态。客户端上的套接字状态现在是 SYN\_SENT。接口层调用 if\_output()（实际上是接口特定的输出函数），将包发送到 n/w。

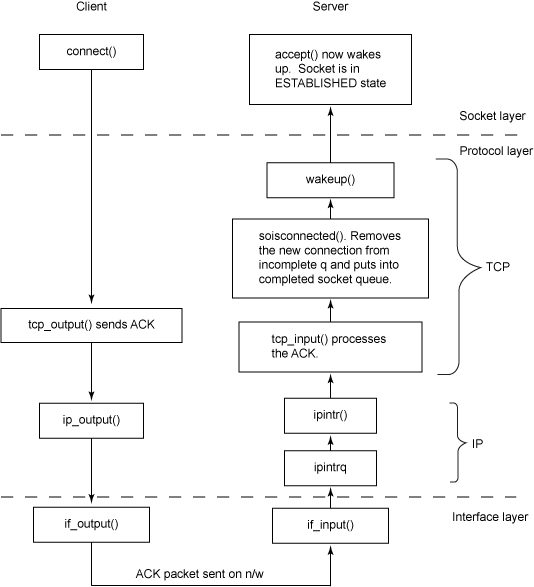
目的地（服务器）上的接口接收传入 SYN 包，将其放在 ipintrq 队列中，并引发软件中断。包然后由调用 tcp\_input 例程的 ipintr() 获取。tcp\_input() 在 s/w 中断时执行，并从 ipintrq 拾取 SYN 包，对其进行处理，并将部分完成的套接字连接放入完成的套接字队列。服务器端的套接字状态现在是 SYN\_RCVD。每次处理后，tcp\_input() 例程都调用 tcp\_output()（如果需要将响应套接字发送到另一端）。

图 9. 用于 SYN ACK 包的流序列



处理 SYN 后，服务器使用 tcp\_output ()、ip\_output () 和 if\_output () 序列发送 SYN ACK 包。客户端上的 n/w 接口接收此包，将其放在 ipintrq 中，并引发 s/w 中断。同样，ipintr () 从 ipintrq 获取该包，并将其传递到客户端 TCP 堆栈上的 tcp\_input () 例程。包现在是经过处理的，并调用了 soisconnected ()，它唤醒连接调用。客户端上的套接字状态现在已建立。

图 10. 用于 ACK 包的流序列



客户端上的 tcp\_input () 例程处理 SYN ACK 包，并调用 tcp\_output () 将 ACK 包发回到服务器。服务器端上的 tcp\_input () 处理此 ACK 包，并调用 soisconnected ()。此函数从未完成的套接字队列移除套接字，并将其放入完成的套接字队列，然后调用 Wakeup ()，以唤醒 accept 调用。服务器端的套接字现在已建立。

Shutdown

shutdown (struct proc \*p, struct shutdown\_args \*uap, int \*retval);

Struct shutdown\_args

{

int s;

int how;

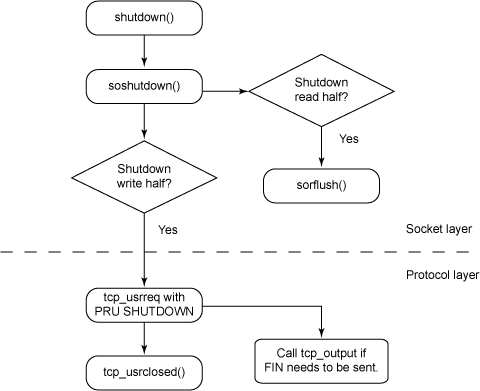
}

在 shutdown 系统调用中：

* s 是套接字描述符。
* how 指定将关闭哪一部分连接。how 的值 0、1 和 2 分别指定关闭连接的读取部分、写入部分和同时关闭连接的读取及写入部分。

shutdown 系统调用关闭连接的任意一端或两端。如果需要关闭读取部分，则会丢弃接收缓冲区中存在的任何数据，并关闭该端的连接。对写入部分，TCP 发送任何剩余的数据，然后终止连接的写入端。

图 11. 用于 shutdown 系统调用的调用序列



如果需要关闭连接的读取部分，则 soshutdown() 函数调用 sorflush()。sorflush() 标记套接字以拒绝任何传入的包，并释放保存的任何系统资源。

如果需要关闭连接的写入部分，则调用 tcp\_usrreq()，并将 PRU\_SHUTDOWN 作为请求。PRU\_SHUTDOWN 的切换实例根据当前的状态调用 tcp\_usrclosed() 函数，以更新套接字的状态。TCP/IP 状态图表可以帮助了解套接字在任何给定的时间存在的不同状态。如果从tcp\_usrclosed() 返回时需要发送 FIN，则调用 tcp\_output() 将其发送到接口。

Close

soo\_close(struct file \*fp , struct proc \*p);

在 close 系统调用中：

* fp 是指向文件结构的指针。
* p 是一个指向调用进程的 proc 结构的指针。

close 系统调用可关闭或中止套接字上任何挂起的连接。

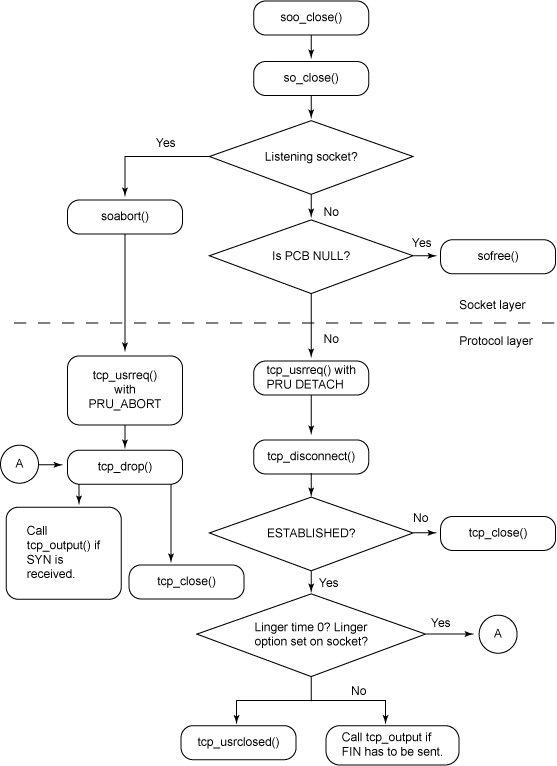
soo\_close() 仅调用 so\_close() 函数，该函数首先检查要关闭的套接字是否为侦听套接字（正在接收传入连接的套接字）。如果是，则遍历两个套接字队列，以检查任何挂起的连接。对每个挂起的连接，将调用 soabort() 以发布 tcp\_usrreq()，并将 PRU\_ABORT 用作请求。此切换实例调用 tcp\_drop() 以检查套接字的状态。

如果状态是 SYN\_RCVD，则通过将状态设置为 CLOSED 并调用 tcp\_output() 发送 RST 段。tcp\_close() 函数然后关闭套接字。tcp\_close 函数更新路由度量结构的三个变量，然后释放套接字持有的资源。

如果套接字不是侦听套接字，则控制开始使用 soclose()，以检查是否已存在附加到套接字的控制块。如果不存在，则 sofree() 释放套接字。如果存在，则调用具有 PRU\_DETACH 的 tcp\_usrreq() 将协议与套接字分离。PRU\_DETACH 的切换实例调用 tcp\_disconnect()，以检查连接状态是否为 ESTABLISHED。如果不是，则 tcp\_disconnect() 调用 tcp\_close()，以释放 Internet 和控制块。否则，tcp\_disconnect() 检查延迟时间和延迟套接字选项。如果设置了该选项，并且延迟时间为零，则调用 tcp\_drop()。如果未设置，则调用 tcp\_usrclosed()，以设置套接字的状态，并调用 tcp\_output()（如果需要发送 FIN 段）。

[图 12](http://www.ibm.com/developerworks/cn/aix/library/au-tcpsystemcalls/#fig12) 显示了 TCP 应用程序发布 close 系统调用时发生的重要调用。

图 12. 用于 close 系统调用的调用序列



Send

sendmsg ( struct proc\*p, struct sendmsg\_args \*uap, int retval);

struct sendmsg\_args

{

int s;

caddr\_t msg;

int flags;

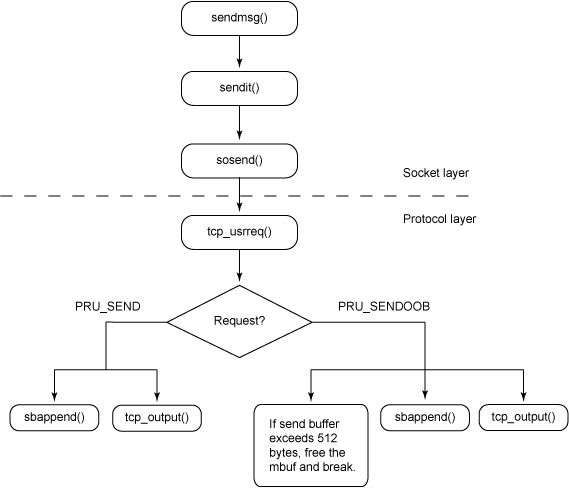
};

在 send 系统调用中：

* s 是套接字描述符。
* msg 是指向 msghdr 结构的指针。
* flags 是控制信息。

n/w 接口上有四个要发送数据的系统调用：write、writev、sendto 和 sendmsg。本文仅讨论 sendmsg() 系统调用。所有的四个调用最终调用 sosend()。尽管 send（进程调用的库函数）、sendto 和 sendmsg 系统调用仅可以对套接字描述符操作，但 write 和 writev 系统调用则可以对任何类型的描述符操作。

图 13. 用于 sendmsg 的调用序列



sendmsg 系统调用将从进程发送的消息复制到内核空间，并调用 sendit()。在 sendit() 中，将初始化一个结构，以便从进程将输出收集到内核中的内存缓冲区。还可以将地址和控制信息从进程复制到内核，然后调用 sosend()，以执行以下四项任务：

* 基于 sendit() 函数传递的值初始化各种参数。
* 验证套接字的条件和连接的状态，并确定传递消息和报告错误所需的空间。
* 分配内存并从进程复制数据。
* 使协议特定的调用将数据发送到网络。

然后调用 tcp\_usrreq()，并根据进程指定的标志，控制切换到 PRU\_SEND 或 PRU\_SENDOOB（以发送带区外数据）。对于 PRU\_SENDOOB，发送缓冲区大小可以超过 512 字节，将释放任何分配的内存并中断控制。否则，sbappend() 和 tcp\_output() 函数由 PRU\_SEND 和 PRU\_SENDOOB 调用。sbappend() 在发送缓冲区的末尾添加数据，并且 tcp\_output() 将该段发送到接口。

Receive

recvmsg(struct proc \*p, struct recvmsg\_args \*uap , int \*retval);

struct recvmsg\_args

{

int s,

struct msghdr \*msg,

int flags,

};

在 receive 系统调用中：

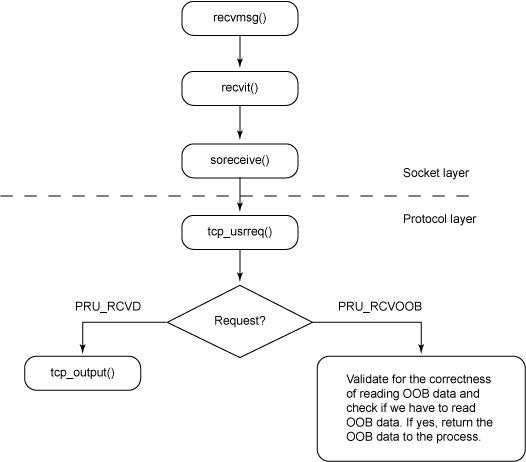
* s 是套接字描述符。
* msg 是指向 msghdr 结构的指针。
* flags 指定控制信息。

有四个系统调用可以用于从连接接收数据：read、readv、recvfrom 和 recvmsg。尽管 recv（进程使用的库函数）、recvfrom 和 recvmsg仅可以对套接字描述符操作，但 read 和 readv 可以对任何种类的描述符操作。所有的 read 系统调用最终调用 soreceive()。

[图 14](http://www.ibm.com/developerworks/cn/aix/library/au-tcpsystemcalls/#fig14) 显示了用于 recvmsg 系统调用的调用序列。recvmsg() 和 recvit() 函数初始化各种数组和结构，将接收的数据从内核发送到进程。recvit() 调用 soreceive()，以便将接收的数据从套接字缓冲区传输到接收缓冲区进程。soreceive() 函数执行各种检查，如：

* 是否设置了 MSG\_OOB 标志。
* 进程是否尝试接收数据。
* 是否应该阻塞，直到足够的数据到达。
* 将读取数据传输到进程。
* 检查数据是带区外数据还是常规数据，并进行相应的处理。
* 当数据接收完成后通知协议。

图 14. 用于 recvmsg 的调用序列



当设置 MSG\_OOB 标志时或数据接收完成后，soreceive() 函数进行与协议相关的请求。在接收带区外数据的情况下，协议层检查不同的条件，以验证接收的数据是否为带区外数据，然后将其返回到套接字层。在后一种情况中，协议层调用 tcp\_output()，将窗口更新段发送到网络。它通知另一端任何空间都可用于接收数据。

结束语

在本文中，您学习了触发低级别调用以完成某些任务的最重要的 TCP 函数调用。图中的调用序列显示了内核级 TCP 调用的简要概述。本文是了解 FreeBSD TCP/IP 堆栈组织的很好起点。

（2） 继上一题，说明socket网络编程有哪些系统调用？其中close是一次就能直接关闭的吗，半关闭状态是怎么产生的？  
（3） 对路由协议的了解与介绍。内部网关协议IGP包括RIP，OSPF，和外部网关协议EGP和BGP.  
（4） 路由协议所使用的算法。

（5） TCP和UDP的区别

TCP和UDP是OSI模型中的运输层中的协议。TCP提供可靠的通信传输，而UDP则常被用于让广播和细节控制交给应用的通信传输。

UDP(User Datagram Protocol)

        UDP不提供复杂的控制机制，利用IP提供面向无连接的通信服务。并且它是将应用程序发来的数据在收到的那一刻，立刻按照原样发送到网络上的一种机制。即使是出现网络拥堵的情况下，UDP也无法进行流量控制等避免网络拥塞的行为。此外，传输途中如果出现了丢包，UDO也不负责重发。甚至当出现包的到达顺序乱掉时也没有纠正的功能。如果需要这些细节控制，那么不得不交给由采用UDO的应用程序去处理。换句话说，UDP将部分控制转移到应用程序去处理，自己却只提供作为传输层协议的最基本功能。UDP有点类似于用户说什么听什么的机制，但是需要用户充分考虑好上层协议类型并制作相应的应用程序。

TCP(Transmission Control Protocol)

        TCP充分实现爱呢了数据传输时各种控制功能，可以进行丢包的重发控制，还可以对次序乱掉的分包进行顺序控制。而这些在UDP中都没有。此外，TCP作为一种面向有连接的协议，只有在确认通信对端存在时才会发送数据，从而可以控制通信流量的浪费。TCP通过检验和、序列号、确认应答、重发控制、连接管理以及窗口控制等机制实现可靠性传输。此处不一一叙述。

TCP与UDP如何加以区分使用？

        TCP用于在传输层有必要实现可靠性传输的情况。由于它是面向有连接并具备顺序控制、重发控制等机制的。所以它可以为应用提供可靠传输。 另一方面，UDP主要用于那些对高速传输和实时性有较高要求的通信或广播通信。举一个IP电话进行通话的例子。如果使用TCP，数据在传送途中如果丢失会被重发，但是这样无法流畅地传输通话人的声音，会导致无法进行正常交流。而采用UDP，它不会进行重发处理。从而也就不会有声音大幅度延迟到达的问题。即使有部分数据丢失，也只是影响某一小部分的通话。此外，在多播与广播通信中也使用UDP而不是UDP。RIP、DHCP等基于广播的协议也要依赖于UDP。

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

（6） TCP和UDP相关的协议与端口号

* 端口号的范围是从1～65535。其中1～1024是被RFC 3232规定好了的，被称作“众所周知的端口”(Well Known Ports)；从1025～65535的端口被称为动态端口（Dynamic Ports），可用来建立与其它主机的会话，也可由用户自定义用途。

一些常见的端口号及其用途如下：

TCP 21端口：FTP 文件传输服务

TCP 23端口：TELNET 终端仿真服务

TCP 25端口：SMTP 简单邮件传输服务

UDP 53端口：DNS 域名解析服务

TCP 80端口：HTTP 超文本传输服务

TCP 110端口：POP3 “邮局协议版本3”使用的端口

TCP 443端口：HTTPS 加密的超文本传输服务

TCP 1521端口：Oracle数据库服务

TCP 1863端口：MSN Messenger的文件传输功能所使用的端口

TCP 3389端口：Microsoft RDP 微软远程桌面使用的端口

TCP 5631端口：Symantec pcAnywhere 远程控制数据传输时使用的端口

UDP 5632端口：Symantec pcAnywhere 主控端扫描被控端时使用的端口

TCP 5000端口：MS SQL Server使用的端口

UDP 8000端口：腾讯QQ

TCP 25端口：SMTP 25仔

TCP 110端口：POP3

**使用TCP协议的常见端口主要有以下几种：**  
（1）FTP：定义了文件传输协议，使用21端口。常说某某计算机开了FTP服务便是启动了文件传输服务。下载文件，上传主页，都要用到FTP服务。  
（2） Telnet：它是一种用于远程登陆的端口，用户可以以自己的身份远程连接到计算机上，通过这种端口可以提供一种基于DOS模式下的通信服务。如以前的BBS是纯字符界面的，支持BBS的服务器将23端口打开，对外提供服务。  
（3） SMTP：定义了简单邮件传送协议，现在很多邮件服务器都用的是这个协议，用于发送邮件。如常见的免费邮件服务中用的就是这个邮件服务端口，所以在电子邮件设置中常看到有这么SMTP端口设置这个栏，服务器开放的是25号端口。  
（4） POP3：它是和SMTP对应，POP3用于接收邮件。通常情况下，POP3协议所用的是110端口。也是说，只要你有相应的使用POP3协议的程序（例如Foxmail或Outlook），就可以不以Web方式登陆进邮箱界面，直接用邮件程序就可以收到邮件（如是163邮箱就没有必要先进入网易网站，再进入自己的邮箱来收信）。

**使用UDP协议的常见端口有：**

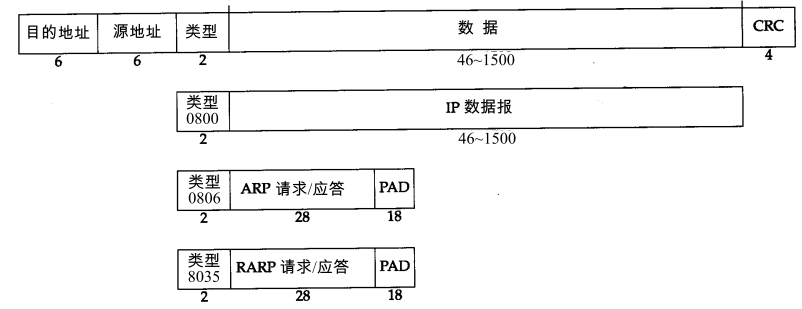
（1）HTTP:超文本传输协议，上网浏览网页时，就得在提供网络资源的计算机上打开端口80来提供服务，常用的“www服务”，“WEB服务器”用的都是这个端口

（2）DNS：域名解析服务。因特网上的每一台计算机都有一个网络地址与之对应，这个地址就是常说的IP地址，访问计算机的时候只需要知道域名，域名和IP地址之间的对应关系由DNS服务器来完成，使用53端口。

（3）SNMP：简单网络管理协议，使用161端口，用来管理网络设备。

（7） TCP（UDP，IP）等首部的认识（http请求报文构成）

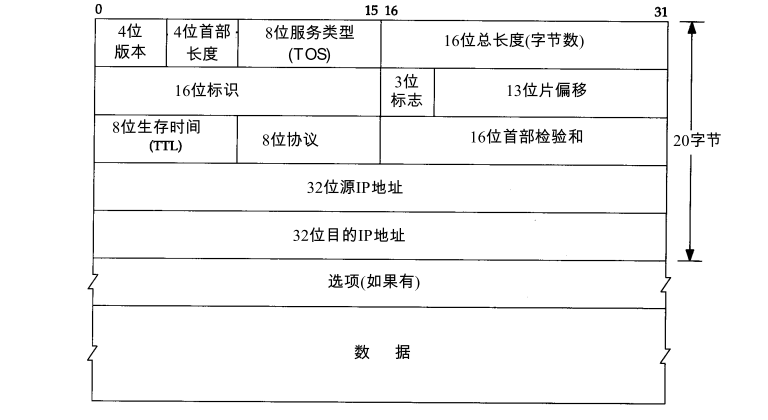
**1.以太网帧的格式**



以太网封装格式

**2.IP报头格式**

　　IP是TCP/IP协议簇中最为重要的协议。所有的TCP，UDP, ICMP和IGMP数据都以IP数据报格式传输。IP提供的是不可靠、无连接的协议。

   
　　普通的IP首部长为20个字节，除非含有选项字段。

4位版本：目前协议版本号是4，因此IP有时也称作IPV4.

4位首部长度：首部长度指的是首部占32bit字的数目，包括任何选项。由于它是一个4比特字段，因此首部长度最长为60个字节。

服务类型（TOS）：服务类型字段包括一个3bit的优先权字段（现在已经被忽略），4bit的TOS子字段和1bit未用位必须置0。4bit的TOS分别代表：最小时延，最大吞吐量，最高可靠性和最小费用。4bit中只能置其中1比特。如果所有4bit均为0，那么就意味着是一般服务。

总长度：总长度字段是指整个IP数据报的长度，以字节为单位。利用首部长度和总长度字段，就可以知道IP数据报中数据内容的起始位置和长度。由于该字段长16bit，所以IP数据报最长可达65535字节。当数据报被分片时，该字段的值也随着变化。

标识字段：标识字段唯一地标识主机发送的每一份数据报。通常每发送一份报文它的值就会加1。

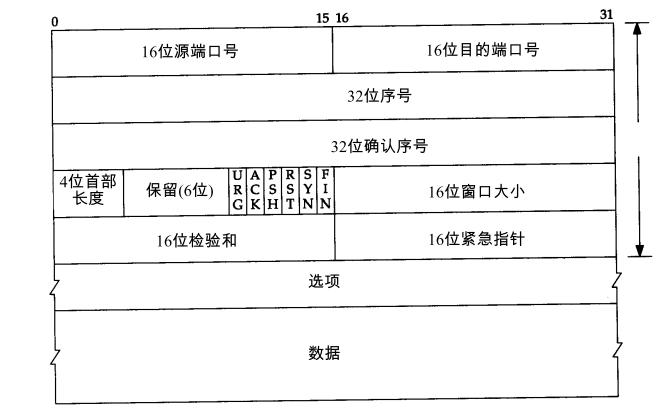
生存时间：T T L（time-to-live）生存时间字段设置了数据报可以经过的最多路由器数。它指定了数据报的生存时间。TTL的初始值由源主机设置（通常为 3 2或64），一旦经过一个处理它的路由器，它的值就减去 1。当该字段的值为 0时，数据报就被丢弃，并发送 ICM报文通知源主机。

首部检验和：首部检验和字段是根据 I P首部计算的检验和码。它不对首部后面的数据进行计算。 I C M P、 I GMP、U D P和T C P在它们各自的首部中均含有同时覆盖首部和数据检验和码。

**3.TCP首部格式**

　　尽管T C P和U D P都使用相同的网络层（ I P），T C P却向应用层提供与U D P完全不同的服务。T C P提供一种面向连接的、可靠的字节流服务。

　　如果不计任选字段，它通常是 2 0个字节。



源端口号和目的端口号：用于寻找发端和收端应用进程。这两个值加上I P首部中的源端I P地址和目的端I P地址唯一确定一个T C P连接。

序号字段：序号用来标识从T C P发端向T C P收端发送的数据字节流，它表示在这个报文段中的的第一个数据字节。如果将字节流看作在两个应用程序间的单向流动，则 T C P用序号对每个字节进行计数。序号是32 bit的无符号数，序号到达 232-1后又从0开始。

当建立一个新的连接时，SYN标志变1。序号字段包含由这个主机选择的该连接的初始序号ISN（Initial Sequence Number）。该主机要发送数据的第一个字节序号为这个ISN加1，因为SYN标志消耗了一个序号（将在下章详细介绍如何建立和终止连接，届时我们将看到 F I N标志也要占用一个序号）

确认序号：既然每个传输的字节都被计数，确认序号包含发送确认的一端所期望收到的下一个序号。因此，确认序号应当是上次已成功收到数据字节序号加 1。只有ACK标志（下面介绍）为 1时确认序号字段才有效。发送ACK无需任何代价，因为 32 bit的确认序号字段和A C K标志一样，总是T C P首部的一 部分。因此，我们看到一旦一个连接建立起来，这个字段总是被设置， ACK标志也总是被设置为1。TCP为应用层提供全双工服务。这意味数据能在两个方向上独立地进行传输。因此，连接的每一端必须保持每个方向上的传输数据序号。

首都长度：首部长度给出首部中 32 bit字的数目。需要这个值是因为任选字段的长度是可变的。这个字段占4 bit，因此T C P最多有6 0字节的首部。然而，没有任选字段，正常的长度是 2 0字节。

标志字段：在T C P首部中有 6个标志比特。它们中的多个可同时被设置为1.   
URG紧急指针（u rgent pointer）有效（见2 0 . 8节）。 ACK确认序号有效。 PSH接收方应该尽快将这个报文段交给应用层。 RST重建连接。 SYN同步序号用来发起一个连接。这个标志和下一个标志将在第 1 8章介绍。 FIN发端完成发送任务。

窗口大小：T C P的流量控制由连接的每一端通过声明的窗口大小来提供。窗口大小为字节数，起始于确认序号字段指明的值，这个值是接收端正期望接收的字节。窗口大小是一个 16 bit字段，因而窗口大小最大为 65535字节。

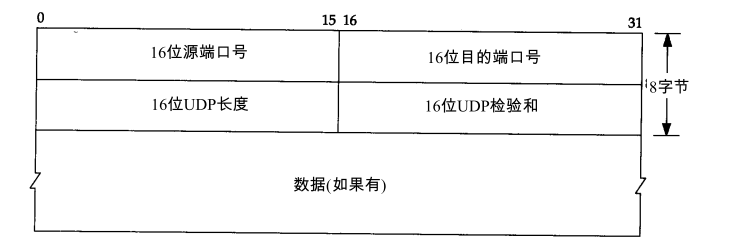
检验和：检验和覆盖了整个的 TCP报文段：TCP首部和TCP数据。这是一个强制性的字段，一定是由发端计算和存储，并由收端进行验证。

紧急指针：只有当URG标志置1时紧急指针才有效。紧急指针是一个正的偏移量，和序号字段中的值相加表示紧急数据最后一个字节的序号。 T C P的紧急方式是发送端向另一端发送紧急数据的一种方式。

选项：最常见的可选字段是最长报文大小，又称为 MSS (Maximum Segment Size)。每个连接方通常都在通信的第一个报文段（为建立连接而设置 S Y N标志的那个段）中指明这个选项。它指明本端所能接收的最大长度的报文段。

**4. UDP首部**

UDP是一个简单的面向数据报的运输层协议：进程的每个输出操作都正好产生一个UDP数据报，并组装成一份待发送的 I P数据报。这与面向流字符的协议不同，如 T C P，应用程序产生的全体数据与真正发送的单个 I P数据报可能没有什么联系。



端口号：用来表示发送和接受进程。由于 I P层已经把I P数据报分配给T C P或U D P（根据I P首部中协议字段值），因此T C P端口号由T C P来查看，而 U D P端口号由UDP来查看。T C P端口号与UDP端口号是相互独立的。

长度：UDP长度字段指的是UDP首部和UDP数据的字节长度。该字段的最小值为 8字节（发送一份0字节的UDP数据报是 O K）。

检验和：UDP检验和是一个端到端的检验和。它由发送端计算，然后由接收端验证。其目的是为了发现UDP首部和数据在发送端到接收端之间发生的任何改动。

（8） 网页解析的过程与实现方法  
（9）    在浏览器中输入URL后执行的全部过程（如[www.baidu.com](http://www.baidu.com)）  
（10） 网络层分片的原因与具体实现  
（11） TCP的三次握手与四次挥手的详细介绍（TCP连接建立与断开是热门问题）  
（12） TCP握手以及每一次握手客户端和服务器端处于哪个状态（11种状态）  
（13） 为什么使用三次握手，两次握手可不可以？  
（14） TIME\_WAIT的意义（为什么要等于2MSL）  
（15） 超时重传机制（不太高频）  
（16） TCP怎么保证可靠性（面向字节流，超时重传，应答机制，滑动窗口，拥塞控制，校验等）？  
（17） 流量控制的介绍，采用滑动窗口会有什么问题（死锁可能，糊涂窗口综合征）？  
（18） tcp滑动窗口协议  
（19） 拥塞控制和流量控制的区别

（20） TCP拥塞控制，算法名字？（极其重要）

（21） http协议与TCP联系  
（22） http/1.0和http/1.1的区别  
（23） http的请求方法有哪些？get和post的区别。  
（24） http的状态码  
（25） http和https的区别，由http升级为https需要做哪些操作  
（26） https的具体实现，怎么确保安全性  
（27） http中浏览器一个URL的流程，这个过程中浏览器做了什么，URL包括哪三个部分？  
（28） 一个机器能够使用的端口号上限是多少，为什么？可以改变吗？那如果想要用的端口超过这个限制怎么办？  
（29） 对称密码和非对称密码体系  
（30） 数字证书的了解（高频）  
（31） 客户端为什么信任第三方证书  
（32） RSA加密算法，MD5原理（MD5不算加密算法）  
（33） 单条记录高并发访问的优化

（34） 介绍一下ping的过程，分别用到了哪些协议

ping 的原理

ping 程序是用来探测主机到主机之间是否可通信，如果不能ping到某台主机，表明不能和这台主机建立连接。ping 使用的是ICMP协议，它发送icmp回送请求消息给目的主机。ICMP协议规定：目的主机必须返回ICMP回送应答消息给源主机。如果源主机在一定时间内收到应答，则认为主机可达。

ICMP协议通过IP协议发送的，IP协议是一种无连接的，不可靠的数据包协议。在Unix/Linux，序列号从0开始计数，依次递增。而Windows　ping程序的ICMP序列号是没有规律。

ICMP协议在实际传输中数据包：20字节IP首部 + 8字节ICMP首部+ 1472字节<数据大小>38字节

ICMP报文格式:IP首部(20字节)+8位类型+8位代码+16位校验和+(不同的类型和代码，格式也有所不同)

Ping工作过程——

假定主机A的IP地址是192.168.1.1，主机B的IP地址是192.168.1.2，都在同一子网内，则当你在主机A上运行“Ping 192.168.1.2”后，都发生了些什么呢?

首先，Ping命令会构建一个固定格式的ICMP请求数据包，然后由ICMP协议将这个数据包连同地址“192.168.1.2”一起交给IP层协议（和ICMP一样，实际上是一组后台运行的进程），IP层协议将以地址“192.168.1.2”作为目的地址，本机IP地址作为源地址，加上一些其他的控制信息，构建一个IP数据包，并在一个映射表中查找出IP地址192.168.1.2所对应的物理地址（也叫MAC地址，熟悉网卡配置的朋友不会陌生，这是数据链路层协议构建数据链路层的传输单元——帧所必需的），一并交给数据链路层。后者构建一个数据帧，目的地址是IP层传过来的物理地址，源地址则是本机的物理地址，还要附加上一些控制信息，依据以太网的介质访问规则，将它们传送出去。

其中映射表由ARP实现。ARP(Address Resolution Protocol)是地址解析协议,是一种将IP地址转化成物理地址的协议。ARP具体说来就是将网络层（IP层，也就是相当于OSI的第三层）地址解析为数据连接层（MAC层，也就是相当于OSI的第二层）的MAC地址。

主机B收到这个数据帧后，先检查它的目的地址，并和本机的物理地址对比，如符合，则接收；否则丢弃。接收后检查该数据帧，将IP数据包从帧中提取出来，交给本机的IP层协议。同样，IP层检查后，将有用的信息提取后交给ICMP协议，后者处理后，马上构建一个ICMP应答包，发送给主机A，其过程和主机A发送ICMP请求包到主机B一模一样。

即先由IP地址，在网络层传输，然后再根据mac地址由数据链路层传送到目的主机

（35） TCP/IP的分片粘包过程  
（36） 有没有抓过TCP包，描述一下  
（37） 一个ip配置多个域名，靠什么识别？  
（38） 服务器攻击（DDos攻击）