**网络**

# 操作系统

## 操作系统其实就是负责管理协调我们计算机硬件与软件资源工作的一种系统软件。

### 计算机安装操作系统,主要就是为了帮助我们屏蔽硬件层的复杂性,给上层的应用软件或用户提供易用的服务。

## 进程

### 进程是动态执行的过程，是操作系统进行资源分配和调度的独立单位，一个应用程序可以有多个进程

## 线程

线程是程序执行中一个单一的顺序控制流程，是程序执行的最小单元，是处理器调度和分派的基本单位。一个进程可以有一个或多个线程,各个线程之间共享进程的内存空间。

### 多个线程是如何执行的？

* 串行方式：任务依次进行 执行完一个程序在执行下一个程序。
* 并行方式：任务一起干多个cpu执行。
* 并发方式：一个cpu，任务依次做一段时间交替进行，看起来像一起完成

### 线程的创建

HANDLE handle = CreateThread(nullptr/\*安全描述符\*/,0/\*线程栈大小，默认1M\*/, &TESDA/\*线程函数的地址\*/, (void\*)50/\*线程函数的参数\*/, 0 /\*线程函数的初始化状态，0：创建即运行，CREATE\_SUSPENDED：挂起\*/,&threadld01/\*线程id\*/);

### 销毁线程的两种方法

* 线程执行完任务自己退出
* 使用TerminateThread();函数销毁线程―—通常在线程while(1)执行任务或者线程卡住不能正常结束任务时使用，防止进程结束时线程资源没有回收

### 杀死线程为什么要等一会

* 有可能线程没有分配时间片，或者线程函数才走到一半
* 或者线程在堵塞状态
* 线程里面有公共资源，强制杀死会来不及保存，导致程序出现错误

### HANDLE 线程句柄

* SuspendThread(handle);//挂起线程
* ResumeThread(handle);//恢复线程

### 为什么要使用线程

为了更快速的完成任务，或者某些场景需要同时做多件事情，就需要使用线程，因为线程可以"同时"执行任务。cpu大部分时间处于空闲时间，浪费了cpu资源，多线程可以让一个程序“同时”处理多个事情，提高效率。

### 内核对象

每一次对createThread函数的成功调用,系统就会在内部为新的线程分配一个内核对象。内核对象是实现管理线程的函数，通过句柄,我们可以得到内核对象。内核对象是提供给用户与系统内核之间交互的接口。内核对象包含挂起计数器，内核对象使用计数器，以及线程上下文(cpu寄存器状态)等。

* 内核对象用计数器管理
* 内核对象回收

• 1、线程退出

• 2、关闭句柄-----closehandle(handle);

• 为什么先关闭线程在关闭句柄

因为每个线程创建时操作系统给每个线程分配内核对象，内核对象有个引用计数为二，关闭线程计数减一关闭句柄计数减一，回收内核对象需要让引用计数变为零

## 进程和线程的区别与联系

### 进程类似于工厂，是系统分配资源的基本单位;线程类似于工厂中的工人，是CPU调度和执行的基本单位

### 一个进程由一个或者多个线程组成

### 进程之间相互独立，但同一个进程下的线程之间是共享程序的内存空间(包括代码段、数据集、堆等)

### 线程上下文切换比进程上下文切换快的多

## 线程状态

### Sleep（）函数不是精准延时 原因：

* sleep（100）是在阻塞状态等100毫秒再到就绪状态没有直接100毫秒分配时间片

### 线程生命周期：产生 存活 消亡

## 线程安全问题/并发问题

线程安全就是并发问题

### .为什么会有线程安全问题?

* 当多个线程同时共享一个全局变量,或者静态变量, 进行写的操作时, 可能会发生数据的冲突问题 ,也就是线程安全问题, 但是做读的操作不会引发线程安全问题

### 什么是线程安全问题

* 就是 多线程环境中 , 且存在数据共享 , 一个线程访问的共享 数据被其他线程修改了, 那么就发生了线程安全问题

### 并发问题

多个线程在在运作时，由于时间片是固定的，有可能在此时间片内线程没有完成任务或者函数没有执行完就切换到下个线程，由于多个线程之间使用的是同一个内存空间并且资源共享特性，就会产生一号线程没执行完二号线程接着执行，等切换到一号线程接着执行原先一号线程没执行完的数据，就会产生数据的结果和预期不同

### 解决

* 解决方法:线程同步，就是通过协调线程执行的顺序，避免多个线程同时操作同一个资源导致并发问题，使结果多次执行结果一致。
* 线程同步方式

• 常见的线程同步方式:原子访问、关键段、事件、互斥量、条件变量、信号量

• 关键段

CRITICAL\_SECTION cs;首先介绍下如何使用关键段。关键段CRITICAL\_SECTION一共就四个函数，使用很是方便。下面是这四个函数的原型和使用说明。函数功能:初始化,定义关键段变量后必须先初始化。函数参数是一个指向关键段的指针。函数原型: void lnitializeCriticalSection(LPCRITICAL\_SECTION lpCriticalSection);函数功能:销毁，用完之后记得销毁。函数参数是一个指向关键段的指针。函数原型:void DeleteCriticalSection(LPCRITICAL\_SECTION lpCriticalSection);函数功能:进入关键区域，系统保证各线程互斥的进入关键区域。函数参数是一个指向关键段的指针。函数原型: void EnterCriticalSection(LPCRITICAL\_SECTION lpCriticalSection);函数功能:离开关键区域。函数参数是一个指向关键段的指针。函数原型: void LeaveCriticalSection(LPCRITICAL\_SECTION IpCriticalSection);

# 库

## 静态库

### 之所以称为静态库，是因为在连接阶段，会将汇编生成的目标文件.o与引用到的库一起打包到可执行的文件中。因此对应的连接方式，成为静态链接

### 静态库对函数库的链接是放在编译时期完成的。

### 程序在运行时与函数库再无瓜葛，移植方便。

### 浪费空间和资源，因为所有相关的目标文件与牵涉到的函数库被链接合成一个可执行文件。

## 动态库

动态库在程序编译时并不会被连接到目标代码中，而是在程序运行是才被载入。不同的应用程序如果调用相同的库，那么在内存里只需要有一份该共享库的实现，规避了空间浪费问题。动态库在程序运行时才被载入，也解决了静态库对程序的更新、部署和发布页会带来麻烦。用户只需要更新动态库即可，增量更新。

### 动态库把对一些库函数的链接载入推迟到程序运行的时期。

### 可以实现进程之间的资源共享。(因此动态库也称为共享库)

### 将一些程序升级变得简单。

### 甚至可以真正做到链接载入完全由程序员在程序代码中控制(显示调用)

## 区别

### 1.代码被载入的时刻不同。

### 2.静态库是在编译期被连接到可执行文件中的，运行时不再需要该静态库因此体积比较大

### 3.动态库是在程序运行时才被载入，引出程序运行时依赖动态库，没代码体积比较小

## 静态库缺点

### 1.空间浪费（多个程序依赖同一个静态库，内存中拷贝多份）

### 2.程序更新、部署、发布相对（动态库）复杂

# 网络基础知识

网络是信息传输、接收、共享的虚拟平台

## 网络基本模型

### iso/osi七层网络模型

* 应用层

• 主要协议

HTTP 80HTTPS 443FIP 21DNS 53DHCP 68Telnet 23Smtp 25SSH 22

• HTTP/HTTPS

• FTP文件传输协议

• DHCP获取IP的协议

• 其作用是通过应用程序间的交互来完成特定的网络应用。该层协议定义了应用进程之间的交互规则，通过不同的应用层协议为不同的网络应用提供服务

* 表示层

该层提供的服务主要包括数据压缩，数据加密以及数据描述。这使得应用程序不必担心在各台计算机中表示和存储的内部格式差异。

• 表示层的作用是使通信的应用程序能够解释交换数据的含义，其位于 OSI 参考模型的第六层，向上为应用层提供服务，向下接收来自会话层的服务

• 提供数据格式转换服务，常见:数据加解密、数据解压缩、图片/视频编解码。

* 会话层

• 会话层就是负责建立、管理和终止表示层实体之间的通信会话

* 传输层

• 传输层的主要任务是为两台主机进程之间的通信提供服务

• 提供应用进程之间的逻辑通信，常见:TCP、UDP、进程、端口:socket。

* 网络层

在发送数据时，网络层把传输层产生的报文或用户数据报封装成分组和包向下传输到数据链路层。在网络层使用的协议是无连接的网际协议（Internet Protocol）和许多路由协议，因此我们通常把该层简单地称为 IP 层

• 网络层的主要任务就是选择合适的网间路由和交换节点，确保数据按时成功传送

• 寻址和路由选择，分组转发数据，常见:路由器、多层交换机、防火墙、IP。

* 数据链路层

• 提供链路管理(数据分帧、物理地址寻址mac、重发等)，常见:网卡、二层交换机。

• 三个服务

• 无确认的无连接服务

无确认的无连接服务是源机器向目的机器发送独立的帧，而目的机器对收到的帧不作确认。 如果由于线路上的噪声而造成帧丢失，数据链路层不作努力去恢复它，恢复工作留给上层去完成。 这类服务适用于误码率很低的情况，也适用于像语音之类的实时传输，实时传输情况下有时数据延误比数据损坏影响更严重。 大多数局域网在数据链路层都使用无确认的无连接服务。

• 有确认的无连接服务

有确认的无连接服务 这种服务仍然不建立连接，但是所发送的每一帧都进行单独确认。 以这种方式，发送方就会知道帧是否正确地到达。如果在某个确定的时间间隔内，帧没有到达，就必须重新发此帧。

• 有确认的面向连接的服务

采用这种服务，源机器和目的机器在传递任何数据之前，先建立一条连接。 在这条连接上所发送的每一帧都被编上号，数据链路层保证所发送的每一帧都确实已收到。 而且，它保证每帧只收到一次，所有的帧都是按正确顺序收到的。面向连接的服务为网络进程间提供了可靠地传送比特流的服务。

* 物理层

• 该层的主要任务是确定与传输媒体的接口的一些特性（机械特性、电气特性、功能特性，过程特性）。

• 提供比特流传输，常见:网线、光缆。在物理层将数字信号转换成电信号或者是光信号

• 实现计算机节点之间比特流的透明传送，尽可能屏蔽掉具体传输介质和物理设备的差异。使其上面的数据链路层不必考虑网络的具体传输介质是什么

### C/S模型

C/S模型，即Client/Server(客户机/服务器)结构，一般要求的特定的客户端。常见的CIS模型像微信、QQ、LOL、酷狗、迅雷这种，每个应用程序有自己的客户端，互相之间不能互通。客户端和服务器之间可以使用任意协议通信，常见使用的是TCP/UDP协议。

### B/S模型

* 浏览器和服务器之间使用HTTP/HTTPS协议

# 应用层协议

## DNS协议（域名解析协议）应用层协议

### DNS服务器将好记的域名解析成ip地址

列如访问百度网站www.baidu.com ，将百度域名转换成ip地址，接着在去访问百度服务器

### 域名解析查询的两种方式

* 递归查询

如果主机所询问的本地域名服务器不知道被查询域名的 IP 地址，那么本地域名服务器就以 DNS 客户端的身份，向其他根域名服务器继续发出查询请求报文，即替主机继续查询，而不是让主机自己进行下一步查询

* 迭代查询：

当根域名服务器收到本地域名服务器发出的迭代查询请求报文时，要么给出所要查询的 IP 地址，要么告诉本地服务器下一步应该找哪个域名服务器进行查询，然后让本地服务器进行后续的查询

## DHCP

# 网络层协议

## 地址解析协议(address revolution protocol)，是根据IP地址获取物理地址的一个TCP/IP协议。

### 什么时候使用

* 在你要求访问一个服务器，你知道ip地址但是不知道对方mac地址

### 计算机网络互通需要ip地址（唯一的）但是计算机从其他计算机获取数据时需要解码这时需要mac ARP就是通过ip地址获取mac mac地址相当于身份证号

### ARP代理

当发送端广播ARP请求时，本地网络上不会有主机回应（因为IP地址是外网的)此时路由器将会回应该请求，则发送源误认为路由器就是目的主机。会将报文全部转发给它，再由路由器转发报文到外网，则该路由器就被称为ARP代理。

### 免费ARP

* 在主机开机配置时，会发送一个目的IP地址为自己IP地址的ARP请求报文，该报文称为免费ARP

• 1.避免在自己IP地址和别人ip重复，确保IP地址的唯一性

• 2.告诉广播域或路由器本机IP绑定的是什么mac方便网络互通

## IP 协议（Internet Protocol）又称互联网协议，是支持网间互联的数据包协议

## RARP反向地址转换协议

## ICMP Internet控制报文协议

## IGMP Internet组管理协议

## RIP 路由信息协议

## OSPF 分布式链路状态协议

## BGP 边界网关协议

# 传输层

## tcp

### tcp协议头

* 1、URG:紧急标志位，如果置1，代表紧急指针生效，说明此包中有紧急需要看的数据，紧急指针指向数据地址;
* 2、ACK:确认序号标志位，如果置1，就代表当前的包是确认包，确认序号生效;
* 3、PSH: push标志位，如果置1，就代表当前包中的数据需要应用层尽快处理，不需要等到接收缓冲区填满后，尽快就将数据传递给应用层;
* 4、RST: reset标志位，重置连接，当连接的两端发生了一些异常的错误导致两端没有连接关系了，需要发送RST为1的包。通知对端需要重新建立连接;
* 5.窗口大小:TCP里面有一个非常重要的知识叫做滑动窗口，是用来做流量控制的。简单来说就是一次性最多能发多少个包。
* 6.校验和:就是TCP头里面的数据需要校验一下，确保数据传输的时候没有问题，如果有问题就告诉对方重发一遍。

• 为什么要校验呢?因为在网络里面数据是一种电信号的形式传播的，电信号会受到电磁波的干扰，数据可能就会有问题，所以就需要校验。

* 7.紧急指针:跟紧急位配合使用，用来识别紧急数据的

### ACK应答机制

* 一次性发送端发送了N个包接收端接收到了会返回一个ack=N+1，ACk=1，数据表示接收端接收到了数据

### RTT（Round-Trip Time）

* 在计算机网络中它是一个重要的性能指标，表示从发送端发送数据开始，到发送端收到来自接收端的确定（接收端收到数据后便立刻发送确定），总共经历的时延。

• RTT由三部分决定：即链路的传播时间、末端系统的处理时间以及路由器的缓存中排队和处理时间。

### RTO(Retransmission Timeout 超时重传时间)

* TCP每发送一个报文段，就对此报文段设置一个超时重传计时器。此计时器设置的超时重传时间RTO应当略大于TCP报文段的平均往返时延RTT，一般可取RTO=2RTT。可以人为调整RTO

### 连接与断开

* 三次握手是 TCP 连接的建立过程

• 三次握手是 TCP 连接的建立过程。在握手之前，主动打开连接的客户端结束 CLOSE 阶段，被动打开的服务器也结束 CLOSE 阶段，并进入 LISTEN 阶段。随后进入三次握手阶段：

• 首先客户端向服务器发送一个 SYN 包，并等待服务器，确认序号为 Seq = x（x 一般取随机数）随后客户端进入 SYN-SENT 阶段。

• 服务器接收到客户端发来的 SYN 包后，对该包进行确认后结束 LISTEN 阶段，并返回一段 TCP 报文，其中：标志位为 SYN 和 ACK，确认号为 Ack = x + 1，随后服务器端进入 SYN-RECV同步接收 阶段。

• 客户端接收到发送的 SYN + ACK 包后，明确了从客户端到服务器的数据传输是正常的，从而结束 SYN-SENT 同步发送阶段。并返回最后一段报文

标志位为 ACK，表示确认收到服务器端同意连接的信号；序号为 Seq = x + 1，表示收到服务器端的确认号 Ack，并将其值作为自己的序号值；确认号为 Ack= y + 1，表示收到服务器端序号 seq，并将其值加 1 作为自己的确认号 Ack 的值。随后客户端进入 ESTABLISHED已确立的。

• 当服务器端收到来自客户端确认收到服务器数据的报文后，得知从服务器到客户端的数据传输是正常的，从而结束 SYN-RECV 阶段，进入 ESTABLISHED 阶段，从而完成三次握手。

* 四次挥手即 TCP 连接的释放

• 首先客户端向服务器发送一段 TCP 报文表明其想要释放 TCP 连接，随后客户端进入 FIN-WAIT-1 阶段，即半关闭阶段，并且停止向服务端发送通信数据。

标记位为 FIN，表示请求释放连接；序号为 Seq = u；

• 服务器接收到客户端请求断开连接的 FIN 报文后，结束ESTABLISHED 阶段，进入 CLOSE-WAIT 阶段并返回一段 TCP 报文，表示接收到客户端释放连接的请求；

标记位为 ACK，表示接收到客户端释放连接的请求；序号为 Seq = v；确认号为 ack= u + 1，表示是在收到客户端报文的基础上，将其序号值加 1 作为本段报文确认号 ack的值；随后服务器开始准备释放服务器端到客户端方向上的连接。客户端收到服务器发送过来的 TCP 报文后，确认服务器已经收到了客户端连接释放的请求，随后客户端结束 FIN-WAIT-1 阶段，进入 FIN-WAIT-2 阶段。

• 服务器端在发出 ACK 确认报文后，服务器端会将遗留的待传数据传送给客户端，待传输完成后即经过 CLOSE-WAIT 阶段，便做好了释放服务器端到客户端的连接准备，再次向客户端发出一段 TCP 报文，随后服务器端结束 CLOSE-WAIT 阶段，进入 LAST-ACK 阶段。并且停止向客户端发送数据

标记位为 FIN 和 ACK，表示已经准备好释放连接了；序号为 Seq = w；确认号 ack= u + 1，表示是在收到客户端报文的基础上，将其序号 Seq 的值加 1 作为本段报文确认号 ack的值。随后服务器端结束 CLOSE-WAIT 阶段，进入 LAST-ACK 阶段。并且停止向客户端发送数据。

• 客户端收到从服务器发来的 TCP 报文，确认了服务器已经做好释放连接的准备，于是结束 FIN-WAIT-2 阶段，进入 TIME-WAIT 阶段，并向服务器发送一段报文

标记位为 ACK，表示接收到服务器准备好释放连接的信号；序号为 Seq= u + 1，表示是在已收到服务器报文的基础上，将其确认号 Ack 值作为本段序号的值；确认号为 ack= w + 1，表示是在收到了服务器报文的基础上，将其序号 Seq 的值作为本段报文确认号的值。

• 随后客户端开始在 TIME-WAIT 阶段等待 2 MSL。服务器端收到从客户端发出的 TCP 报文之后结束 LAST-ACK 阶段，进入 CLOSED 阶段。由此正式确认关闭服务器端到客户端方向上的连接。客户端等待完 2 MSL 之后，结束 TIME-WAIT 阶段，进入 CLOSED 阶段，由此完成「四次挥手」。

* 问题

• 如果三次握手的时候每次握手信息对方没有收到会怎么样

若第一次握手服务器未接收到客户端请求建立连接的数据包时，服务器不会进行任何相应的动作，而客户端由于在一段时间内没有收到服务器发来的确认报文， 因此会等待一段时间后重新发送 SYN 同步报文，若仍然没有回应，则重复上述过程直到发送次数超过最大重传次数限制后，建立连接的系统调用会返回 -1。若第二次握手客户端未接收到服务器回应的 ACK 报文时，客户端会采取第一次握手失败时的动作，这里不再重复，而服务器端此时将阻塞在 accept() 系统调用处等待 client 再次发送 ACK 报文。若第三次握手服务器未接收到客户端发送过来的 ACK 报文，同样会采取类似于客户端的超时重传机制，若重传次数超过限制后仍然没有回应，则 accep() 系统调用返回 -1，服务器端连接建立失败。但此时客户端认为自己已经连接成功了，因此开始向服务器端发送数据，但是服务器端的 accept() 系统调用已返回，此时没有在监听状态。因此服务器端接收到来自客户端发送来的数据时会发送 RST 报文给 客户端，消除客户端单方面建立连接的状态。

• 为什么要进行三次握手？两次握手可以吗？

三次握手的主要目的是确认自己和对方的发送和接收都是正常的，从而保证了双方能够进行可靠通信。若采用两次握手，当第二次握手后就建立连接的话，此时客户端知道服务器能够正常接收到自己发送的数据，而服务器并不知道客户端是否能够收到自己发送的数据。 我们知道网络往往是非理想状态的（存在丢包和延迟），当客户端发起创建连接的请求时，如果服务器直接创建了这个连接并返回包含 SYN、ACK 和 Seq 等内容的数据包给客户端，这个数据包因为网络传输的原因丢失了，丢失之后客户端就一直接收不到返回的数据包。由于客户端可能设置了一个超时时间，一段时间后就关闭了连接建立的请求，再重新发起新的请求，而服务器端是不知道的，如果没有第三次握手告诉服务器客户端能否收到服务器传输的数据的话，服务器端的端口就会一直开着，等到客户端因超时重新发出请求时，服务器就会重新开启一个端口连接。长此以往， 这样的端口越来越多，就会造成服务器开销的浪费。

• 第 2 次握手传回了 ACK，为什么还要传回 SYN

• ACK 是为了告诉客户端发来的数据已经接收无误，而传回 SYN 是为了把自己的初始序列号（Seq）同步给客户端。

* 问题

释放 TCP 连接时之所以需要四次挥手，是因为 FIN 释放连接报文和 ACK 确认接收报文是分别在两次握手中传输的。 当主动方在数据传送结束后发出连接释放的通知，由于被动方可能还有必要的数据要处理，所以会先返回 ACK 确认收到报文。当被动方也没有数据再发送的时候，则发出连接释放通知，对方确认后才完全关闭TCP连接。

• 为什么要四次挥手？

释放 TCP 连接时之所以需要四次挥手，是因为 FIN 释放连接报文和 ACK 确认接收报文是分别在两次握手中传输的。 当主动方在数据传送结束后发出连接释放的通知，由于被动方可能还有必要的数据要处理，所以会先返回 ACK 确认收到报文。当被动方也没有数据再发送的时候，则发出连接释放通知，对方确认后才完全关闭TCP连接。

• TIME-WAIT 为什么是 2MSL

当客户端发出最后的 ACK 确认报文时，并不能确定服务器端能够收到该段报文。所以客户端在发送完 ACK 确认报文之后，会设置一个时长为 2 MSL 的计时器。MSL（Maximum Segment Lifetime），指一段 TCP 报文在传输过程中的最大生命周期。2 MSL 即是服务器端发出 FIN 报文和客户端发出的 ACK 确认报文所能保持有效的最大时长。 若服务器在 1 MSL 内没有收到客户端发出的 ACK 确认报文，再次向客户端发出 FIN 报文。如果客户端在 2 MSL 内收到了服务器再次发来的 FIN 报文，说明服务器由于一些原因并没有收到客户端发出的 ACK 确认报文。客户端将再次向服务器发出 ACK 确认报文，并重新开始 2 MSL 的计时。 若客户端在 2MSL 内没有再次收到服务器发送的 FIN 报文，则说明服务器正常接收到客户端 ACK 确认报文，客户端可以进入 CLOSE 阶段，即完成四次挥手。 所以客户端要经历 2 MSL 时长的 TIME-WAIT 阶段，为的是确认服务器能否接收到客户端发出的 ACK 确认报文。

## udp

### 特点

* 1、面向非链接，接收数据时，可以接收任意设备发送的数据，可以是一对一，也可以是一对多
* 2、数据报文的通讯方式，数据包不可拆分
* 3、传输效率高（相对tcp）
* 4、会产生丢包，没有校验，还可能出现乱序的问题

### 基于udp协议的服务器小例子

* 加载库

• // 1、选项目——加载库WSAStartupwVersionRequested = MAKEWORD(2, 2); err = WSAStartup(wVersionRequested, &wsaData);

* 加载套接字

• sock = socket(AF\_INET, SOCK\_DGRAM, IPPROTO\_UDP);

* 绑定ip

//3、摆摊——绑定IP地址bind    sockaddr\_in ipbang;    ipbang.sin\_family = AF\_INET;//family表示是什么IP    ipbang.sin\_port = htons(571089);//将整型变量从主机字节序转换成网络字节序    ipbang.sin\_addr.S\_un.S\_addr = INADDR\_ANY;//服务器需要绑定任意网卡   // inet\_pton(AF\_INET, "10.100.26.187", &addrClient.sin\_addr.S\_un.S\_addr);//inet\_addr:将ip地址从字符串转换成u\_long类型\*\*\*\*这是vs2012；2019用前面函数

* 接收消息/回消息

iResult = recvfrom(sock,RecvBuf, sizeof(RecvBuf), 0, (SOCKADDR\*)&addrClient, &addrClientSize); iResult = sendto(sock, sendBuf, sizeof(sendBuf), 0, (sockaddr\*)&addrClient, addrClientSize);

• iResult = recvfrom(sock,RecvBuf, sizeof(RecvBuf), 0, (SOCKADDR\*)&addrClient, &addrClientSize);

* 关闭套接字卸载库

// 6、收摊回家——关闭套接字、卸载库closesocket、WSACleanup    closesocket(sock);    WSACleanup();

# IP地址分类子网掩码

## iP地址组成结构：网络号+子网号+主机号

### A类地址是从1.0.0.0 到 127.255.255.255， B类地址是从128.0.0.0 到191.255.255.255， C类地址是从192.0.0.0到223.255.255.255， D类地址是从224.0.0.0 到239.255.255.255， E类地址是从240.0.0.0 到 255.255.255.255。

## 网关，又称网间连接器，协议转换器。网关都是具有路由功能的IP地址,就是连接两个子网之间的设备或者软件

## 子网掩码：某主机的网络地址(对外的ip地址) = 某主机ip地址 & 子网掩码

### 又叫网络掩码，地址掩码，子网络遮罩，就是说把子网络遮起来了，不让外界窥探到。

## 太网帧结构

### 以太网帧结构 = 目标mac(6) +源mac(6) + 类型(2) + 数据 (46-1500 ) + 帧校验 (4)

### 数据 (46-1500 )MTU = IP头 (20) + TCP头(20 + tcp选项 ) + 用户数据 MSS

### 以太网帧的大小范围 64 ~ 1518 MSS最大大小 = 1500 - 20(ip头) – 20(tcp头) =1460

## 有限广播：255.255.255.255

### 

# 分支主题 7

# TCP 流量控制与拥塞控制

## 流量控制

### 所谓流量控制就是让发送方的发送速率不要太快，让接收方来得及接收。如果接收方来不及接收发送方发送的数据，那么就会有分组丢失。

### 在 TCP 中利用可变长的滑动窗口机制可以很方便的在 TCP 连接上实现对发送方的流量控制。

### 主要的方式是接收方返回的 ACK 中会包含自己的接收窗口大小，以控制发送方此次发送的数据量大小（发送窗口大小）。

### 窗口大小由哪一方决定

* 三次握手的时候，会交换窗口大小，通常窗口大小是由接收方的窗口大小决定

## 拥塞控制

在实际的网络通信系统中，除了发送方和接收方外，还有路由器，交换机等复杂的网络传输线路，此时就需要拥塞控制。拥塞控制是作用于网络的，它是防止过多的数据注入到网络中，避免出现网络负载过大的情况。常用的解决方法有：慢开始和拥塞避免、快重传和快恢复

### 慢开始

当发送方开始发送数据时，由于一开始不知道网络负荷情况，如果立即将大量的数据字节传输到网络中，那么就有可能引起网络拥塞。一个较好的方法是在一开始发送少量的数据先探测一下网络状况，即由小到大的增大发送窗口（拥塞窗口 cwnd）。

* 慢开始的慢指的是初始时令 cwnd为 1，即一开始发送一个报文段。如果收到确认，则 cwnd = 2，之后每收到一个确认报文，就令 cwnd = cwnd\* 2。
* 但是，为了防止拥塞窗口增长过大而引起网络拥塞，另外设置了一个慢开始门限 ssthresh。

• ① 当 cwnd < ssthresh 时，使用上述的慢开始算法；

• ② 当 cwnd > ssthresh 时，停止使用慢开始，转而使用拥塞避免算法；

• ③ 当 cwnd == ssthresh 时，两者均可。

### 拥塞避免

拥塞控制是为了让拥塞窗口 cwnd 缓慢地增大，即每经过一个往返时间 RTT （往返时间定义为发送方发送数据到收到确认报文所经历的时间）就把发送方的 cwnd 值加 1，通过让 cwnd 线性增长，防止很快就遇到网络拥塞状态

* 当网络拥塞发生时，让新的慢开始门限值变为发生拥塞时候的值的一半,并将拥塞窗口置为 1 ,然后再次重复两种算法（慢开始和拥塞避免）,这时一瞬间会将网络中的数据量大量降低。

### 快重传

快重传算法要求接收方每收到一个失序的报文就立即发送重复确认，而不要等到自己发送数据时才捎带进行确认，假定发送方发送了 Msg 1 ~ Msg 4 这 4 个报文，已知接收方收到了 Msg 1，Msg 3 和 Msg 4 报文，此时因为接收到收到了失序的数据包，按照快重传的约定，接收方应立即向发送方发送 Msg 1 的重复确认。 于是在接收方收到 Msg 4 报文的时候，向发送方发送的仍然是 Msg 1 的重复确认。这样，发送方就收到了 3 次 Msg 1 的重复确认，于是立即重传对方未收到的 Msg 报文。由于发送方尽早重传未被确认的报文段，因此，快重传算法可以提高网络的吞吐量。

* 所谓的快重传算法，就是让发送方尽快重传，而不是等待超时重传计时器超时再重传
* 要求接收方不要等待自己发送数据时才捎带确认，而是要立即发送确认
* 即使是失序的报文段，也要立即发送对已收到的报文段的重复确认
* 发送方一旦收到3个连续的重复确认，就将相应的报文段立即重传，而不是等待该报文的重传计时器超时再重传

### 快恢复

* 快恢复算法是和快重传算法配合使用的，该算法主要有以下两个要点：

• ① 当发送方连续收到三个重复确认，执行乘法减小，慢开始门限 ssthresh变为发生拥塞时候的窗口值的一半；

• ② 由于发送方可能认为网络现在没有拥塞，因此与慢开始不同，把 cwnd 值设置为发生拥塞时候的窗口值的一半，然后执行拥塞避免算法，线性增大 cwnd。

## 拥塞控制和流量控制的区别

拥塞控制往往是一种全局的，防止过多的数据注入到网络之中，而TCP连接的端点只要不能收到对方的确认信息，猜想在网络中发生了拥塞，但并不知道发生在何处，因此，流量控制往往指点对点通信量的控制，是端到端的问题。

### 拥塞控制是作用与网络的，流量控制作用与端对端