IP层只向上提供简单的，无连接的，尽最大努力交付的数据报服务。可能出错，丢失，重复，失序。各分组独立选择路由。

什么是IP地址？互联网是一个抽象的单一的网络，给世界范围内的每台主机分配一个唯一的32位标识符。IP地址由 网络号+主机号 组成。

IP地址32位组成，MAC地址48位组成。IP地址是逻辑地址，MAC地址是网卡硬件地址。

路由器是一种专用计算机，进行路由选择，只负责最低三层协议。

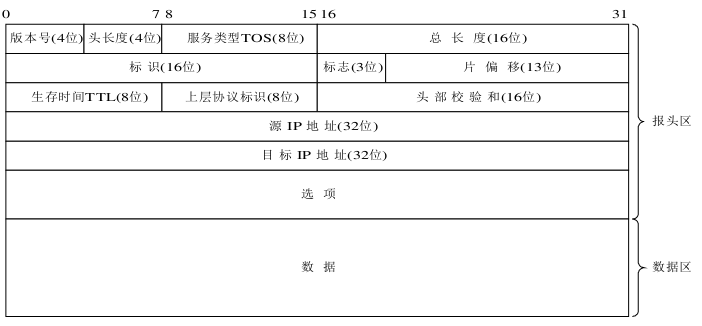
路由主机会现在自己的路由表查找IP是否在本网络，是的话直接交付，由ARP管理具体哪台主机。不是则查找自己路由表，看应该发给哪个另外的路由，间接交付。如果IP在本网络，那么就通过ARP协议，得到目的主机的MAC地址。

ARP协议：在同一局域网中，有IP地址，要得到MAC地址。

每台主机都有一个APR高速缓存，里面有本局域网上各主机和路由器的IP地址到MAC地址的映射表。如果查找的IP在表中有，直接把得到的硬件地址写入MAC帧，如果没有，会在本地局域网广播发送一个ARP请求分组，每台收到的请求IP地址与自己不一致，则丢弃。如果一致就将自己的MAC地址写入ARP响应分组。（请求是广播，响应是单播）。请求主机收到后，将添加新的映射，以备下次用。

但是ARP映射都有一定生存时间。10分钟，为了更新。

IP数据报格式



**首部长度**：4位，最小5，最大15，单位为4字节，所以正常20字节，最长60字节。首部长度不是4字节倍数时，用最后的填充字段填充。

**总长度**：首部+数据，共16位，65535字节。但是以太网的数据链路层MTU最大1500字节。一个IP数据报超过这个长度，会被截断后分片。所以不能超过MTU。

**标识**：一个计数器，每产生一个数据报，计数器+1，数据报被分片时，标识字段的值被复制，因此相同标识字段的值在分片后能够正确地重装成完整的数据报。

**标志：**MF，MF=1表示还有分片，MF=0表示最后一个分片。

DF，DF=1时不允许分片。 第三位没用。

**片偏移：**13位。被分片后，当前数据报在原分组中的相对位置。以8字节为单位。

**生存时间：**TTL。每一跳-1。为0时，丢弃数据报。设为1时，只能在局域网传送。

**协议**：携带的哪种协议，让上层知道，常用的ICMP,IGMP,TCP,UDP,IPV6等等。

**首部校验和**： 只检查数据报的首部，不包括数据部分。（减少工作量）

分组转发算法：P134

路由表并不能给出一条完整的路径，而是指出到某个网络前应该先到某个路由器。一路问一路走。

传统IP地址分5类：

A 0-127 其中0代表本网络，127用于环回测试。网络号1字节，第一位0固定

B 128-191，其中必须从128.1.0.0开始，网络号2字节，前两位10固定

C 192-223 其中必须从192.0.1.0开始，网络号3字节，前三位110固定

D 多播地址， 1110

E 保留未使用 1111

划分子网（两级IP变三级）： 网络号：子网号：主机号

通过网络号来看，对外还是同一个网络。

子网掩码：为了区分IP地址所连接的网络是否进行了子网划分。

无分类域间路由选择：P142

ICMP 网际控制报文

又分为 差错报告报文 和 询问报文。

差错报告报文： 3 终点不可达， 11 时间超过， 12参数问题，5 改变路由。

询问报文： 8或0 回送请求和回答， 测试目的站是否可达及其状态。

13或14 时间戳请求与回答。

报文格式： 类型，代码，校验和。 ICMP数据部分是从IP数据部分的前8个字节（即）TCP或UDP头部的部分信息。

Ping分组网间探测，测试两台主机连通性，使用了ICMP的回送请求或回答（8或0），是应用层直接使用ICMP的例子，跳过传输层。

路由选择算法：内部网管协议，外部网管协议。

整个互联网分为许多自治系统AS（分层次的）。内部网管协议在一个AS内。外部负责各AS之间。

内部有：RIP， OSPF 外部有：BGP-4

RIP：基于距离向量的路由选择协议。以跳数为基准。RIP适用于小型互联网。不管网速，只管跳数短就可以。 它仅和相邻路由交换。交换当前自己路由表知道的所有信息，固定时间交换。RIP是使用运输层的UDP传送。特点的好消息传得快，坏消息传的慢。

OSPF：使用了迪杰斯特拉算法。采用了洪泛滥，即把自身信息传给其他所有路由。因此所有路由能建立一个链路数据库。因此更新过程收敛得很快。 PSPF直接使用IP数据报传送。

外部：BGP

由于整个互联网体系过于庞大，各个AS之间，要找一条最佳路由很难，BGP只是力求找一条比较好的路由。

IGMP

隧道技术： 需要多播的两个路由之间不支持多播，其中一个对多播数据报再次封装成单播，然后发往目的地。

UDP 对应用层交下来的报文，既不合并，也不拆分，保留这些报文边界，一次发送一个报文。如果应用层发送的报文太大，UDP发给IP层后，IP层可能会对其分片，降低IP层效率。

UDP没有拥塞控制，网络出现的拥塞不会让源主机降低发送速率。因此可以用于实时应用。并且UDP首部开销小。

UDP的不可靠传输可以在应用层采用前向纠错或重传已丢失的报文。

首部字段： 源端口，目的端口，长度，检验和。

校验要在用户数据前加12个伪首部。只是为了计算校验和。

TCP是根据对方发送的窗口大小和拥塞窗口的较小值来决定一个报文段包含多少字节，而UDP是应用层一次传多少，就给IP层多少。

停止等待协议：一次发一个分组，收到ACK才发下一个。（信道利用率太低）

自动重传请求（ARQ）： 确认丢失，确认迟到。

连续ARQ协议。

操作系统

进程管理：

上下文：又称进程状态。上下文包括操作系统管理进程及CPU正确执行进程所需要的全部信息。

上下文包括各种处理器寄存器的内容，程序计数器 数据寄存器，还包括进程优先级及是否等待特定I/O等。

上下文切换就是指进程间的切换，先将换下的进程的上下文保存，再将换进进程的上下文信息写入。

操作系统建立和维护一张进程表。表里包含每个进程的存储块地址的指针。还有该进程的上下文。

进程索引器寄存器包含当前正在控制处理器的进程在进程表中的索引。

程序计数器指向该进程下一条待执行的指令。

基址寄存器定义了该进程所占的存储器的开始地址。

界限寄存器定义了该区域的大小。

程序计数器和所有数据引用相对于基址寄存器被解释，且不能超过界限寄存器的值。确保进程间不干涉。·

为了满足多个用户作业或是多个进程同时驻留在内存中工作，但因为进程大小不同，处理器在进程间切换，很难把他们紧密地压入内存中，因此引入了分页系统。进程由固定大小的块组成，被称为页。程序通过虚地址访问。虚地址包括页号和页中偏移量。有硬件支持虚地址和物理地址的映射。 在此基础上，为了满足消除一个进程的所有页同时驻留在内存中，已增大进程的并发量。进程的所有页保存在磁盘中。进程运行时，部分页在内存中，如果需要的页不在内存中，引发缺页中断，调入内存。这个叫虚拟内存。

进程是程序代码段在相关数据集上的一次运行的过程。

进程控制块PCB包括： 进程标识符，进程状态，进程优先级，程序计数器，内存指针，上下文数据，I/O状态信息。

进程的五种状态转换图。创建，就绪，运行，阻塞，结束。其中就绪，运行，阻塞都在内存中。 为了提高效率，加入了挂起。 即有创建，就绪挂起，就绪，运行，阻塞挂起，阻塞，结束。挂起态都在磁盘中。

线程有独立的栈，和TCB，线程控制块包含寄存器的值，优先级等等。

线程比进切换的速度比进程快，而且线程间通信不需要内核介入，而进程需要。

线程可分为用户级线程，内核级线程，混合方式。

用户级线程，完全由应用程序管理，内核不知道其存在，可以省去用户和内核的切换时间，但是用户级线程一个阻塞会使整个进程阻塞。并且用户级线程不支持多处理程序。

内核级线程，可以让一个进程的多个线程在多个核上跑，一个阻塞不会影响整个进程。

但是有切换开销。

死锁预防：破坏4个条件中的一个。但是互斥条件破坏不了，是由资源本身决定的。

死锁避免：存在一种安全状态时，才允许进程运行。典型的：银行家算法。安全状态是指至少有一个资源分配序列不会导致死锁。必须事先声明每个进程请求的最大资源。

死锁检测：每次资源请求时检查死锁。频繁检查会耗费相当多的CPU时间。

内存管理：

页框：内存中一个固定长度的块。

页：一个固定长度的数据块，保存的虚拟地址空间的信息，存储在磁盘中，可以被临时复制到内存的页框中。

段：一个变长的数据块，存储在磁盘，整个段可以临时复制到内存中（段式），或者可以把段分成许多页，每页单独复制到内存。（段页式）

**1、内存分区：**

固定分区，把内存分成大小相同的分区，缺点，内存碎片，大程序放不进。

动态分区，动态分配内存，但是会导致内存中出现许多小空间，外存碎片。

解决方法：紧缩技术，选择**放置算法，**最佳适配，首次适配，下次适配。

伙伴系统：把内存块分成2的幂次方大小的块。有个列表维护着所有块。分配时一直/2，直到分出的块满足2(i-1)<n<=2i大小，用完后，将相邻的块再合并成2(i+1)。

基址寄存器，界限寄存器，前者是被载入程序在内存中的起始地址，后者规定大小。

指令一般通过相对地址+基址，如果没有超过界限，则执行，否则发中断，报错。

**2、分页：**

分页时，仅有基址寄存器不够，还要为**每个进程维护一个页表**，页表给出了该进程的每一页（逻辑地址）对应的页框（物理地址）的位置。每个逻辑地址包括一个页号和一个页内偏移量。而在简单分区的情况下，逻辑地址是一个字相对基址的偏移。

**操作系统为当前内存中可用的所有页框维护一个空闲页框列表（供所有进程用）。**

**3、分段**

和动态分区类似，不过一个程序可以占据多个分区，且分区可以不连续。任然有外碎片。

也需要为进程维护段表，逻辑地址为段号+段内偏移。操作系统维护一个空闲段列表。转换成物理地址时比页式多一步，即确认偏移量是否大于该段长度 。

**虚拟存储**

虚存和前几种方式的不同之处在于，利用了局部性原理，可以让进程运行时只加载部门到内存中，当执行到相应的程序不在内存中时，产生中断请求调页。调页完成继续执行刚才那句程序。

局部性原理。

虚拟页式存储：

一个进程还是需要一个页表，其中页表项为：P（是否在内存中），M（是否被写），页框号。由于只是一些页在内存中，所以页表的长度随进程长度变化而变化。且由于页表也占据空间，所以页表一般也用虚拟存储，只保留部分在内存中，这一部分包括正在运行的页的页表项。寻址是寄存器+页号得到页表中的页框号，页框号+偏移量得到物理地址。

二级分页：把用户页表也采用虚存，放在一个根页表中，让根页表放在内存中，减小大小。多一次寻址，需要更多寄存器支持。

TLB快表，转换检测缓冲区，是一种高速缓存。

一个虚存访问可能引起两次磁盘访问，一次是取页表项，一次取需要的数据。为了加快速度，引入的TLB。 CPU先检查TLB中有没有虚拟地址，有直接求取并返回。没有，再访问页表，查看该页是否在内存中，如果是，更新TLB，并求取返回。如果不在内存中，才产生缺页中断。

虚拟段式：

允许程序独立地重新编译，而不要求整个程序集合重新编译。

有助于共享和保护，因为有基地址和界限。不能访问超出的内存单元。

驻留集：进程换入时，操作系统给该进程分配的内存大小

置换策略：最佳（OPT），先进先出(FIFO)，最近最少（LRU），时钟（CLOCK）

调度策略：

先来先服务（FCFS） 短进程在长进程后面不好。

时间片轮转

短作业优先 长作业饥饿

最高响应比

多级反馈队列。

IO管理和磁盘调度

IO是计算机内部和外部世界交互的接口。由于IO种类过多。比如打印机和磁盘，导致接口的控制难度差别大，难找出同一的IO解决方法，只能以层次化的管理。

IO的传输单位可以分为两种：字节流传输（终端IO）和块传输（磁盘IO）。

磁盘和USB都是面向块的，终端、打印机、通信端口、鼠标是面向流的。

IO功能的发展经历了几个阶段：

忙轮询（CPU和IO串行工作，效率极低）

中断（CPU和IO可以并行，但是频繁的中断导致效率提高不大。

DMA（直接存储器访问），传输过程不需要CPU参与，只开最开始和结束参与。CPU从IO任务中脱离出来，提高了性能。由DMA负责内存和IO模块之间的数据交换。一次传输一个块。

通道：比DMA更高一级，是一个专用的处理器，有专门的IO指令集，一次可以同时传输处理多个块。

IO缓冲： 假设从磁盘读入多个块，每个块512字节，数据被读入用户进程的虚拟空间。此时进程阻塞被换下CPU。一是IO完成比较慢，二是干扰了系统的交换决策。数据传输期间，IO读入的内容所占据的虚存地址要保留在内存中，否则数据丢失。为了避免这些开销和低效。出现了缓冲技术。缓冲是指：输入请求发起前执行输入传送，输出请求发起一段时间后才开始执行输出。（预读取，延迟写）。缓冲技术会在内存中开辟一块系统缓冲区。输入的数据从磁盘先放到系统缓冲区，再从系统缓冲区拷贝到用户进程空间。输出同理。由于输入发生在内存中而非用户进程内，因此可将该进程换出。

如果输入或输出速度比缓冲速度快，还是会阻塞。所以由单缓冲引出了双缓冲，循环缓冲的技术。

缓冲技术不仅可以平滑计算机系统内部速度和IO设备速度之间的差异，还可以把IO传送从应用进程的地址空间中分离出来。

磁盘调度：

寻道时间、旋转时间、传输时间。其中 寻道+旋转=存取时间。

磁盘的速率比内存要低上几个数量级，除了优化IO，还需要优化磁盘以提高速度。

磁盘调度策略就是为了减少磁盘寻道时间的，常用的有以下几种：

FIFO：先进先出，按磁盘请求的先后顺序处理。

SSTF: 最短服务时间优先，离当前磁头臂移动最少的磁盘IO请求优先。

SCAN：电梯算法。沿一个方向依次满足未完成请求，到达一端后（或这个方向没有未完成），换方向再满足未完成的。像电梯一样。这种算法有一点不好。刚过某一道，马上出现了此道后面的请求。这样可能会等2T的时间。

C-SCAN：只一个方向移动。移动到终点马上返回起点继续移动。减小了最大延迟。

前几种因某一段出现频繁的磁盘IO请求，导致磁头臂粘性。可用 NSCAN。即将请求队列分成长度为N的子队列，每个子队列按SCAN来处理。

磁盘高速缓存：

普通的高速缓存是一个比内存小且快的存储器，位于内存和CPU之间。通过利用局部性原理，减少平均存储器存取时间。

同理，磁盘高速缓存是在内存中为磁盘扇区开辟的一个缓冲区。当请求一个磁盘扇区的IO请求时，会先检测磁盘高速缓冲中有没有。如果有，满足。没有，从磁盘调入，并记录进磁盘高速缓冲中。

提高IO性能，最常用的两种方式： 磁盘调度和磁盘高速缓存。

文件管理：

由于磁盘IO请求是以块为基本单位，需要将文件中的 记录 组织成一组块的序列输出。

文件组织和访问的几种形式：

堆（非彼堆）

顺序文件（逻辑组织在磁盘上连续，查找效率不高，对所有记录的访问最快）

索引顺序文件：关建域和指向主文件的指针。

索引文件：一个文件可以按不同属性访问，有多个索引。对放置位置无限制。

B树的插入：

树中搜索关键码，有直接返回，没有会到达最底层的某个节点。

如果这个节点关键码少于n-1,直接插入。如果等于n-1。把该节点从中间分成两份，然后插入的关键码加入某一部分，中间节点提升到上一层。如果没满，直接插，如果满了，继续提升。提升到最上方任然满，就把根节点也分两份，中间变为新根节点。树高度+1.

文件目录：

目录项包含文件的信息，如属性，位置，所有权，同时文件目录自身也是一个文件。

文件目录提供了文件名和文件自身之间的一种映射。

有时候文件目录项中含有的信息太多，可以把某些信息保存在文件相关联的头记录中，比如I节点。可以减少目录的大小。在内存中可以保留更多的目录，提高速度。

为了在共享系统中避免文件名重复，且为了同类文件可以构造文件集合，一般文件的结构都采用树状结构。主目录下是用户目录，用户目录下有 子目录目录项 和 文件的目录项。

记录是访问结构化文件的逻辑单元，块是与辅助存储进行的IO操作的基本单位。为执行IO操作，必须组织成块。固定组块，可变长度跨越式组块，可变长度非跨越式组块。

文件分配： 文件分配表（FAT）用于记录哪些块已经分配给哪些文件用了。

分区：一组连续的已经分配了的块。

一般给一个新文件分配块时，按概念有两种方式，一种是可变的大规模连续分区，一种是按块分（有点像段式和页式的区别）。

具体的可分为：

连续分配：给文件分配一组连续的块，文件分配表中每个文件需要一个表项，用于说明起始块和文件的长度。这种方式可能会出现外碎片，需要定期采用紧缩。

链接分配：基于单个块，每一块指向下一块，一个文件只需要一个表项。用于说明起始块和文件的长度。没有外部碎片。但是局部性原理不再适用，可能多次访问磁盘。

索引分配： 是连续分配和链接分配的结合。每个文件在文件分配表中有一个一级索引。文件可能被分成几个连续的块，连续块之间用指针链接。

空闲空间的管理：

没有分配的空闲空间也需要管理，使用 磁盘分配表（DAT）来管理。

位图：每一个块用一位来表示，0可用，1分配。有点是占用内存小。基本可以常驻内存，缺点是磁盘快满时，穷举地查找会使性能很低。

链接空闲区：和上面的链接分配有点像，不过这里因为是分配，所以会分配出现外部碎片，这里的分配算法可以利用内存中动态分区使用的几种分配方式。

索引：和上面的索引分配有点像。

空闲链表块。

Linux虚拟文件系统（VFS），独立于任何具体的文件系统，负责把用户的文件系统调用映射成目标文件系统的调用。VFS有四个主要的对象：

超级块对象：代表一个特定的已挂载的文件系统

索引节点对象：代表一个特定的文件

目录项对象：代表一个特定的目录项，为访问文件和目录提供了方便。

文件对象：代表一个与进程相关的打开的文件。

RTTI的dynamic\_cast关系。

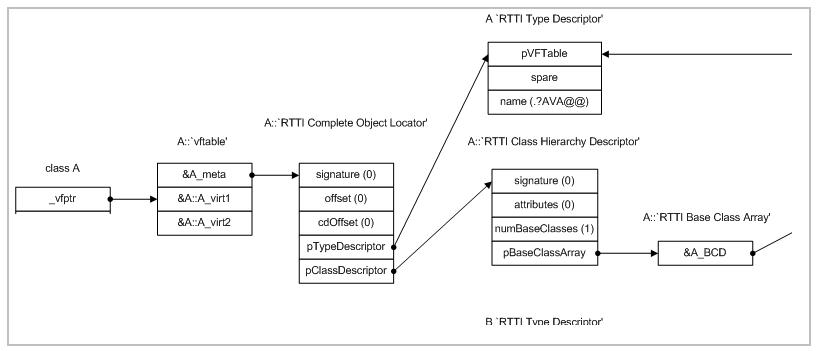
typeid 操作符返回的结果是名为type\_info的标准库类型的对象的引用。 RTTI 运行时类型识别，靠的就是type\_info来判断的类型。

而具有虚函数的类，其虚函数表前面的位置保存了一个指向type\_info的指针。多态就是在程序运行时，通过虚指针指向相应对象的虚函数表，

调用相应方法，对应的通过其虚函数表前的type\_info指针，识别其类型。

dynamic\_cast的底层实现。

原来虚表上面指向是一个 Derive::`RTTI Complete Object Locator 。



编译报错：程序具有语法，词法的错误。

链接报错：在编译完成后，链接过程中找不到使用的函数或者变量的定义。

<https://blog.csdn.net/dreamvyps/article/details/83051367>

C变长结构体。一个业务场景需要使用一个变长数据结构。而且需要使用连续内存。比如说，你想将它映射到一段共享内存中的时候。

struct MyData

{

int nLen;

char data[0];

};

MyData \*myData = (MyData\*)malloc(sizeof(MyData) + 10);

GDB 函数调用栈的查看：

backtrace：查看函数调用的顺序（函数调用栈的信息）

frame N：切换到栈编号为N的上下文中

info frame：查看当前函数调用的栈帧信息

info命令的其他使用方式：



GDB调试多线程

//查看当前运行的进程

ps aux|grep a.out

//查看当前运行的轻量级进程

ps -aL|grep a.out

//查看主线程和新线程的关系

pstree -p 主线程id

