《孙钟秀 . 操作系统教程》注释 (稿)

(下册)

编注: 陈怀临

谨以此注纪念前南京大学副校长，计算机系博士生导师，中国科学院院士孙钟秀教授（1936.12.22-2013.05.18）

(In Memory of Professor SUN Zhongxiu, Nanjing University, China)

Everything is a file是现代操作系统的一个重要原理。通过文件系统的接口来统一抽象和管理。

第五章

块设备（Block Device）

设备管理

字符设备（Character Device）

现代计算机系统配置大量外部设备，外部设备分为两大类：一类是存储型设备（如磁带机、磁盘机等），以存储大量信息和快速检索为目标，在系统中存储持久性信息，它作为主存储器的扩充，所以又称辅助存储器；另一类是 I/O 型设备（如显示器、卡片机、打印机、通信设备等），它们把外界信息输入计算机，把计算结果从计算机中输出，完成计算机之间的通信或人机交互。

设备管理是操作系统中最为庞杂和琐碎的部分，通常使用 I/O 中断、缓冲区管理、通道、设备驱动调度等多种技术，这些措施较好地克服了由于设备和 CPU 速度的不匹配所引起的问题，使主机和设备能够并行工作，提高设备使用效率。另一方面，操作系统将所有设备（如磁带机、打印机、显示器和终端等）都定义为文件，将其统一在文件系统之下，赋予文件属性，对设备的操作就类似于对文件的操作，其优点是：尽可能统一文件和设备的 I/O 处理；尽可能把设备文件和普通文件纳入同一保护机制下。为了方便用户或高层进程使用，设备管理还对各种设备进行抽象，配置驱动程序，提供统一界面，屏蔽设备的物理细节和操作过程。为此，设备管理应具有以下功能：设备中断处理；缓冲区管理；设备分配和去配；设备驱动调度；虚拟设备及其实现。

I/O 硬件原理

5.1.1 I/O 系统

通常把 I/O 设备及其接口线路、控制部件、通道和管理软件称为 I/O 系统，把计算机的主存储器和设备介质之间的信息传送操作称为 I/O 操作。随着计算机技术的飞速进步和应用领域的不断扩大，计算机的 I/O 信息量急剧增加，I/O 设备的种类和数量越来越多，它们与主机的联络和信息交换方式各不相同，I/O 操作不仅影响计算机的通用性和可扩充性，而且成为计算机系统综合处理能力及性能价格比的重要因素。

可按照不同方式对设备进行分类：按 I/O 操作特性，分为输入型设备、输出型设备和存储型设备；按 I/O 信息交换单位，分为字符设备和块设备。输入型设备和输出型设备通常是字符设备，它与主存储器进行信息交换的单位是字节，即一次交换 1 个或多个字节；存储型设备通常是

通常I/O设备都是Memory Mapping的I／O。意味着，对I/O设备就像对内存操作一样的读写控制寄存器和缓冲区。

对通信系统，FPGA，ASIC芯片等，从CPU的角度看，都是设备。但这些设备本身就具备计算处理能力。都是在并行工作，例如，处理网络报文。

块设备。存储型设备又可分为顺序存取存储设备和直接存取存储设备。前者严格依赖信息的物理位置进行定位和读写，如磁带。后者的重要特性是存取任何一个物理块所需的时间几乎不依赖于此信息所处的位置，如磁盘。所谓块是存储介质上连续信息所组成的一个区域，块设备每次与主存储器交换一个或多个块信息。

不同设备的物理特性存在很大的差异，主要表现在：数据传输速率差别大，数据表示方式和传输单位不尽相同，错误的性质、报错方法及应对措施都不一样。这些差异使得无论从操作系统还是用户的角度都难以获得规范、一致的I/O解决方案。

5.1.2 I/O控制方式

I/O控制在计算机处理中占据重要的地位，为了有效地实现物理I/O操作，必须通过软硬件技术，对CPU和设备的职能进行合理分工，以平衡系统性能和硬件成本之间的矛盾。按照I/O控制器功能的强弱以及它和CPU之间联系方式的不同，可以把设备的控制方式分为4类，它们之间的差别在于：CPU和设备并行工作的方式和程度不同。CPU和设备并行工作具有重要的意义，能大幅度提高计算机系统的效率和资源利用率。

1. 轮询方式

轮询方式又称程序直接控制方式，使用查询指令测试设备控制器的忙闲状态位，确定主存储器和设备是否能交换数据。假如CPU上所运行的程序需要从设备读入一批数据，则CPU程序设置交换字节数和数据读入主存储器的起始地址，然后，向设备发出查询指令，设备控制器便把状态返回给CPU。如果I/O操作忙或未就绪，则重复测试过程，继续进行查询；否则，开始数据传送，CPU从I/O接口（控制器数据寄存器）读取一个字，再用存储指令保存到主存储器。如果传送尚未结束，再次向设备发出查询指令，直到全部数据传输完成。轮询方式使用3条指令：

(1) 查询指令：查询设备是否就绪；

(2) 读写指令：当设备就绪时，执行数据交换；

(3) 转移指令：当设备未就绪时，执行转移指令转向查询指令继续查询。

有时系统中同时有多个设备要求执行I/O操作，那么对每个设备都编写一段I/O数据处理程序，然后，轮流查询这些设备的状态位，当某一个设备准备就绪时，就调用这个设备的I/O处理程序进行数据传输，否则依次轮询下一个设备是否准备好。

一方面，CPU轮询设备当前的状态，会终止原程序的执行，浪费宝贵的时间；另一方面，I/O准备就绪后，需要CPU参与数据传输工作。可见CPU和设备只能串行工作，使主机不能充分发挥功效，设备也不能得到合理利用，整个系统效率很低。

2. 中断方式

中断方式要求CPU与设备控制器及设备之间存在中断请求线，设备控制器的状态寄存器有相应的中断允许位，CPU与设备之间传输数据的过程如下。

(1) 进程发出启动I/O指令，这时CPU会加载控制信息到设备控制器的寄存器，然后，进程继续执行不涉及本次I/O数据的任务，或放弃CPU等待设备I/O操作完成；

有一种混合方式是：通过时钟中断，
然后去定期的查询是否有I/O，例如网卡是否有报文。

例如，Packet Buffer

(2) 设备控制器检查状态寄存器的内容，按照I/O指令的要求，执行相应的I/O操作(如发现读请求时，命令设备把数据传送到I/O缓冲寄存器)，一旦传输完成，设备控制器通过中断请求线发出I/O中断信号；

(3) CPU收到并响应I/O中断后，转向设备的I/O中断处理程序执行；

(4) 中断处理程序(有的操作系统把控制权再交给设备驱动程序，并退出中断)执行数据读取操作，将I/O缓冲寄存器的内容写入主存储器，操作结束后退出中断处理程序，返回发生中断前的执行状态；

(5) 进程调度程序在适当时刻对于得到数据的进程恢复执行。

在I/O中断方式中，如果设备控制器的数据缓冲区较小，当缓冲器装满后便会发生中断，那么在数据传输的过程中，发生中断的次数会较多，这样会耗用大量CPU时间。若系统配置多种设备，这些设备都通过中断处理方式实现并行工作，会使中断次数急剧增加，造成CPU来不及响应或丢失数据的现象。但是，程序中断方式I/O由于不必忙式轮询I/O的准备情况，CPU和设备可实现部分并行操作，与程序查询I/O的串行工作方式相比，使CPU资源得到更充分的利用。

3. DMA方式

虽然中断方式消除程序轮询中的忙式测试，提高了CPU资源的利用率，但在响应中断请求之后，必须停止现行程序，转入中断处理程序并参与数据传输操作。如果设备能直接与主存储器交换数据而不占用CPU，那么，CPU资源的利用率还可以提高，这就出现了直接存储器存取(Direct Memory Access，DMA)方式。在DMA方式中，主存储器和设备之间有一条数据通路，成块地传送数据，无须CPU干预，实际数据传输操作由DMA直接完成。为了实现直接存储器存取操作，至少需要以下一些逻辑部件。

(1) 主存地址寄存器：存放主存中需要交换数据的地址，DMA传送之前，由程序送入首地址；DMA传送过程中，每次交换数据都把地址寄存器的内容加1；

(2) 字计数器：记录传送数据的总字数，每次传送一个字就把字计数器减1；

(3) 数据缓冲寄存器或数据缓冲区：暂存每次传送的数据；

(4) 设备地址寄存器：存放I/O信息的地址，如磁盘的柱面号、磁道号、块号；

(5) 中断机制和控制逻辑：用于向CPU提出I/O中断请求，及保存CPU发来的I/O命令，管理DMA的传送过程。

DMA与主存储器之间采用字传送，DMA与设备之间可能是字位或字节传送，所以，DMA中要设置数据移位寄存器、字节计数器等硬件逻辑。可能会出现这样的问题：为什么控制器从设备读取数据后不立即将其送入主存储器，而需要内部缓冲区呢？其原因是，一旦磁盘开始读数据，从磁盘读出比特流的速率是恒定的，无论控制器是否做好接收这些比特的准备，若此时控制器要将数据直接复制到主存储器中，必须在每个字传送完毕后获得对系统总线的控制权。如果其他设备也在争用总线，则有可能暂时等待，当上一个字还未送入主存储器而下一个字已经到达时，控制器只能另寻暂存之地，如果总线非常忙，则控制器可能需要对大量的信息暂存。可见，采用内部缓冲区，在DMA操作启动前不需要使用总线，这样控制器的设计就比较简单，因为从

从总线的角度，CPU，内存和I／O设备其实都是挂在Bus上的器件。CPU本身也是设备。DMA方式就是I／O设备通过Bus直接把数据发送给内存DRAM。

http://en.wikipedia.org/wiki/Channel_I/O

DMA 到主存储器的传输对时间的要求并不十分严格。

DMA 不仅设有中断机制，还增加 DMA 传输控制机制，若出现 DMA 与 CPU 同时经总线访问主存储器的情况，CPU 总是把总线占有权让给 DMA，DMA 的这种占有称为"周期窃用"。窃取时间通常为一个存取周期，让设备和主存储器之间交换数据，而不再需要 CPU 干预，这样可以减轻 CPU 的负担。每次传输数据时，不必进入中断系统，进一步提高 CPU 资源的利用率。

由于 DMA 方式的实现线路简单，价格低廉，小型计算机、微型计算机中的快速设备均采用这种方案。但是，DMA 传输需要窃用时钟周期，会降低 CPU 的处理效率；DMA 的功能不够强，不能满足复杂的 I/O 操作要求，因而在大中型计算机中一般使用通道技术。

4. 通道方式

Channel

DMA 方式与程序中断方式相比，进一步减少了 CPU 对 I/O 操作的干预，但每发出一次 I/O 指令，只能读写一个数据块，用户希望一次能够读写多个离散的数据块，并把它们传送到不同的主存区域或相反，则需要由 CPU 发出多条启动 I/O 指令及进行多次 I/O 中断处理才能完成。通道方式是 DMA 方式的发展，能够再次减少 CPU 对 I/O 操作的干预。同时，为了充分发挥 CPU 和设备之间的并行工作能力，也为了让种类繁多且物理特性各异的设备能够以标准的接口连接到系统中，计算机系统引入自成体系的通道结构，通道方式的出现是现代计算机系统功能不断完善、性能不断提高的结果，是计算机技术的重要进步。

通道又称 I/O 处理器，能完成主存储器和设备之间的信息传送，与 CPU 并行地执行操作。采用通道技术主要解决 I/O 操作的独立性和硬部件工作的并行性，由通道来管理和控制 I/O 操作，大大减少设备和 CPU 之间的逻辑联系，把 CPU 从琐碎的 I/O 操作中解放出来，实现设备和 CPU 并行操作，通道之间并行操作，设备之间并行操作，达到提高整个系统效率的目的。

具有通道装置的计算机系统的主机、通道、控制器和设备之间采用四级连接，实施三级控制。一个 CPU 通常可以连接若干通道，一个通道可以连接若干控制器，一个控制器可以连接若干台设备。CPU 通过执行 I/O 指令对通道实施控制，通道通过执行通道命令对控制器实施控制，控制器发出动作序列对设备实施控制，设备执行相应的 I/O 操作。

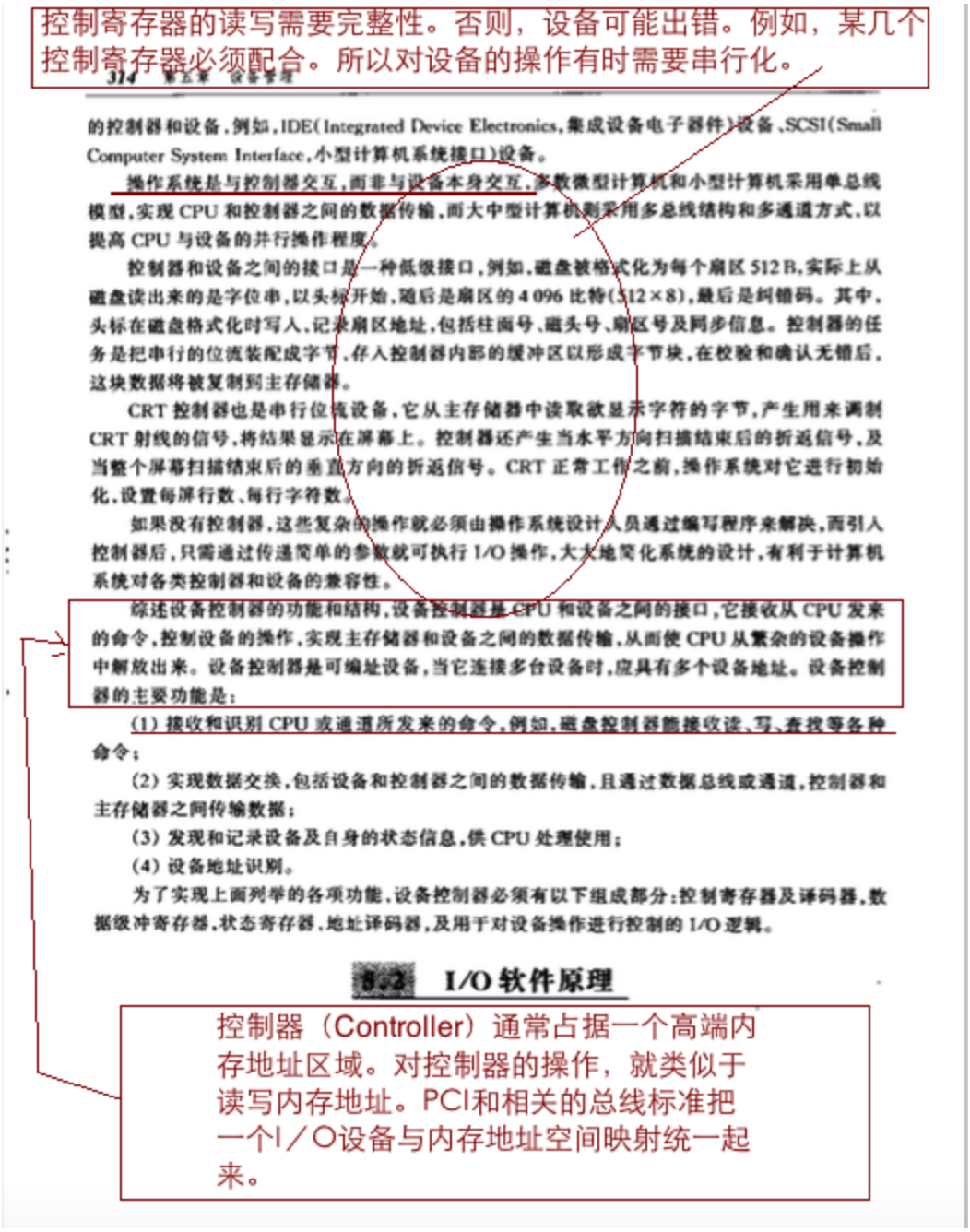
采用 I/O 通道设计后，I/O 操作的过程如下：CPU 在执行主程序时遇到 I/O 请求，启动在指定通道上选址的设备，一旦启动成功，通道开始控制设备进行操作，这时 CPU 就可以执行其他任务并与通道并行工作，直到 I/O 操作完成；当通道发出 I/O 操作结束中断时，处理器才响应并停止当前的工作，转而处理 I/O 操作结束事件。

5.1.3 设备控制器

I/O 设备通常由机械部件和电子部件所组成，一般是将其分开处理，以达到模块化和通用性的设计目标。电子部件称为设备控制器或适配器，在微型计算机中，它是可插入主板扩充槽的印刷电路板；机械部件是指设备本身。

控制器卡上一般都设有接线器，与设备相连的电缆线相接，许多控制器可以控制两个或更多的同类设备，它和设备之间的接口采用国际标准，这样各计算机厂商都可制造与标准接口相匹配

通道（Channel）方式之前主要用在大型机上。在中小计算机，通信设备系统不常见。channel有点类似协处理器。现代DMA Engine融合了早期Channel的工作方式，是独立的bus竞争，而非盗用。



控制寄存器的读写需要完整性。否则，设备可能出错。例如，某几个控制寄存器必须配合。所以对设备的操作有时需要串行化。

的控制器和设备，例如，IDE(Integrated Device Electronics,集成设备电子器件)设备，SCSI(Small Computer System Interface,小型计算机系统接口)设备。

操作系统是与控制器交互，而非与设备本身交互。多数微型计算机和小型计算机采用单总线模型，实现 CPU 和控制器之间的数据传输，而大中型计算机则采用多总线结构和多通道方式，以提高 CPU 与设备的并行操作程度。

控制器和设备之间的接口是一种低级接口，例如，磁盘被格式化为每个扇区 512 B，实际上从磁盘读出来的是字位串，以头标开始，随后是扇区的 4 096 比特(512×8)，最后是纠错码。其中，头标在磁盘格式化时写入，记录扇区地址，包括柱面号、磁头号、扇区号及同步信息。控制器的任务是把串行的位流装配成字节，存入控制器内部的缓冲区以形成字节块，在校验和确认无错后，这块数据将被复制到主存储器。

CRT 控制器也是串行位流设备，它从主存储器中读取欲显示字符的字节，产生用来调制 CRT 射线的信号，将结果显示在屏幕上。控制器还产生当水平方向扫描结束后的折返信号，及当整个屏幕扫描结束后的垂直方向的折返信号。CRT 正常工作之前，操作系统对它进行初始化，设置每屏行数、每行字符数。

如果没有控制器，这些复杂的操作就必须由操作系统设计人员通过编写程序来解决，而引入控制器后，只需通过传送简单的参数就可执行 I/O 操作，大大地简化系统的设计，有利于计算机系统对各类控制器和设备的兼容性。

综述设备控制器的功能和结构，设备控制器是 CPU 和设备之间的接口，它接收从 CPU 发来的命令，控制设备的操作，实现主存储器和设备之间的数据传输，从而使 CPU 从繁杂的设备操作中解放出来。设备控制器是可编址设备，当它连接多台设备时，应具有多个设备地址。设备控制器的主要功能是：

(1) 接收和识别 CPU 或通道所发来的命令，例如，磁盘控制器能接收读、写、查找等各种命令；

(2) 实现数据交换，包括设备和控制器之间的数据传输，且通过数据总线或通道，控制器和主存储器之间传输数据；

(3) 发现和记录设备及自身的状态信息，供 CPU 处理使用；

(4) 设备地址识别。

为了实现上面列举的各项功能，设备控制器必须有以下组成部分：控制寄存器及译码器，数据缓冲寄存器，状态寄存器，地址译码器，及用于对设备操作进行控制的 I/O 逻辑。

5.2 I/O 软件原理

控制器（Controller）通常占据一个高端内存地址区域。对控制器的操作，就类似于读写内存地址。PCI和相关的总线标准把一个I／O设备与内存地址空间映射统一起来。

[注释]：“Everything is a file”是现代操作系统的一个重要概念。通过文件系统的抽象和接口来屏蔽设备多线性。理解设备时，要站在总线角度看：CPU和其他设备，例如ASIC芯片，都是设备。都具备计算处理能力。需要和内存打交道，大家共享总线。

现在有的设备控制器可以有选项，攒多少个事件之后，才向CPU发一个中断申请。

盘I/O操作的效率。通用性意味着用统一标准的方法来管理所有设备。为了达到这些目标，把软件组织成层次结构，低层软件用来屏蔽硬件细节，高层软件向用户提供简洁、友善的界面。I/O软件设计需要考虑的问题有：

(1) 设备无关性：程序员编写访问文件数据的程序时，与具体的物理设备无关；

(2) 出错处理：数据传输过程中产生的错误应该在尽可能靠近硬件的地方处理，低层软件能够解决的错误不让高层软件感知；

(3) 同步/异步传输：CPU在启动I/O操作后既可继续执行其他工作，直至中断到达，称此为异步传输；又可采用阻塞方式，让启动I/O操作的进程挂起等待，直至数据传输完成，称此为同步传输，I/O软件应支持这两种工作方式；

(4) 缓冲技术：建立数据缓冲区，让数据的到达率与离去率相匹配，以提高系统吞吐率。

为了合理、高效地解决以上问题，操作系统把I/O软件依次组织成4个层次：I/O中断处理程序；I/O设备驱动程序；独立于设备的I/O软件和用户空间的I/O软件。

user land device driver

5.2.2 I/O中断处理程序

通常中断处理程序是设备驱动的组成部分之一，位于操作系统底层，是与硬件设备密切相关的软件，它与系统的其余部分尽可能少地发生联系。当进程请求I/O操作时，通常将被挂起，直到数据传输结束并产生I/O中断时，操作系统接管CPU后转向中断处理程序执行。

当设备向CPU提出中断请求时，CPU响应请求并转入中断处理程序执行，通常需要做到：检查设备状态寄存器的内容，判断产生中断的原因，根据I/O操作的完成情况进行相应的处理；若数据传输有错，应向上层软件报告设备出错信息，实施重执行；若正常结束，应唤醒等待传输的进程，使其转换为就绪态；若有等待传输的I/O命令，应通知相关软件启动下一个I/O请求。

5.2.3 I/O设备驱动程序

设备驱动程序包括与设备密切相关的所有代码，其任务是：把用户提交的逻辑I/O请求转化为物理I/O操作的启动和执行，如设备名转化为端口地址、逻辑记录转化为物理记录、逻辑操作转化为物理操作等。同时，监督设备是否正确执行，管理数据缓冲区，进行必要的纠错处理。

笼统地说，设备驱动程序的功能是从独立于设备的软件中接收并执行I/O请求。一条典型的I/O请求是读第 n 块，如果请求到来时驱动程序空闲，它立即执行这个请求；但如果它正在处理请求，会将新到来的请求放入I/O请求队列中，待当前请求完成后，再依次从I/O请求队列中取出I/O请求进行处理。

执行I/O请求的第一步是将其转换为更具体的形式。对于磁盘驱动程序而言，它包含：计算所请求块的物理地址、检查驱动器是否运转、检测磁头臂是否定位在正确的柱面位置等，简而言之，它必须确定需要哪些控制器命令及其执行次序，然后就向控制器的设备寄存器写入这些命令和相应的参数。某些控制器一次只能接收一条命令(如DMA)，另一些控制器可接收一串命令并自动进行处理(如通道方式)。

现代OS都支持异步I/O。进程不需要阻塞等待。

设备驱动程序发出控制命令之后，系统有两种处理方式。在大多数情况下，执行设备驱动程序的进程需要等待命令完成，在命令开始执行后，进程阻塞自己，直到中断处理时才解除阻塞。另一种情况是执行驱动程序的进程无须阻塞，例如，有些终端的滚动屏幕只需向控制器的设备寄存器中写入几个字节，整个操作可在几微秒内完成。无论是哪种情况，在操作完成后都要检查数据传输是否有错，如果有错，返回错误状态提示信息给调用者；如果正常，驱动程序将数据传送给与设备无关的软件层；如果 I/O 请求队列中有请求在排队，则选中其中一个请求执行，否则等待下一个 I/O 请求的到来。

设备驱动程序主要包含三部分功能：

1. 设备初始化

在系统初次启动或设备传输数据时，预置设备和控制器以及通道的状态。

2. 执行设备驱动例程

负责启动设备，进行数据传输，对于具有通道的 I/O 系统，此例程还负责形成通道指令，启动通道工作。

3. 执行中断处理例程

负责处理设备和控制器及通道所发出的各种中断。

5.2.4 独立于设备的 I/O 软件

虽然设备驱动程序是设备专用的，但大部分 I/O 软件却与设备无关，独立于设备的 I/O 软件和设备驱动程序之间的界限取决于具体的系统，其基本功能是执行适用于所有设备的常用 I/O 功能，并向用户层软件提供一致性接口。

1. 设备命名和设备保护

对于操作系统而言，设备都被看做文件，它与磁盘文件一样，通过路径名来寻址，每个设备具有文件名、inode、文件所有者、权限位等属性。例如，在 UNIX/Linux 中，设备名/dev/tty00 唯一地确定为一个特别文件设置的 inode 节点，其中包含主设备号，通过它来分配正确的终端设备驱动程序；次设备号作为参数，确定驱动程序要读写的是哪台终端设备。设备不仅具有文件名，且支持与文件相关的所有系统调用，如 open、close、read、write、stat 及 lseek 等。

设备保护需要检查用户是否有权访问所申请的设备，在多数大中型计算机系统中，用户进程对 I/O 设备的直接访问是绝对禁止的，I/O 指令定义为特权指令，通过系统调用的方式间接地供用户使用。此外，在 UNIX/Linux 中还使用一种灵活的方法，对应于 I/O 设备的特别文件采用 rwx 保护机制，以判断文件的所有者和组成员是否具有向设备发送和读取数据的权限。Windows 中的设备作为命名对象出现在文件系统中，对文件的保护规则也适用于 I/O 设备。

设备文件依赖于 inode 来实现，文件目录并不能区分文件名是代表一个磁盘文件还是设备文件，但 inode 的内容是不同的，磁盘文件的 inode 包含指向数据块的指针，而设备文件的 inode 则包含指向内核设备驱动程序的指针，用来控制设备的 I/O 操作。

换言之，写设备驱动程序就是要实现该设备相应的open，close，read，write等函数。

有时还分为两个缓冲区域，例如，描述符缓冲和数据（Payload）缓冲。

这是为什么user land driver很流行的原因。

2. 提供与设备无关的块尺寸

当创建文件并向其输入数据时，此文件必须被分配新的磁盘块。为了完成这种分配工作，操作系统需要为每个磁盘都配置一张已录空闲块的表或位示图，分配空闲块的算法是独立于设备的，可在高于设备驱动程序的层次处理。

不同磁盘的扇区大小可能不同，独立于设备的 I/O 软件屏蔽这一事实并向高层软件提供统一的数据块尺寸，它可将若干扇区合并为一个逻辑块，称为簇。这样，高层软件就只能和大小相同的簇交互，而不必关心物理扇区的尺寸。类似地，字符设备每次对一个字符进行操作，有些字符设备操作更大单位的数据（如 80 个字符），这类差别也必须在本层进行屏蔽。

3. 缓冲技术

数据离开设备之后，通常不能直接送达目的地，如网络上传送的数据包需要经过分析、检查才能知道置于何处。某些设备有严格的时间约束，如数字音频设备，所以，必须通过缓冲区来消除填满速率和清空速率的影响。块设备和字符设备都需要缓冲技术，可通过在主存储器建立缓冲区的方法来解决。对于块设备，硬件每次读写均以块为单位，而应用程序可按照任意大小单元处理数据，如果应用程序读写的长度和位置不恰好是完整的扇区时，也必须通过缓冲区来实现，先将数据读或写入缓冲区，再从缓冲区将数据传送给用户，或当缓冲区满时再写至磁盘。对于字符设备，当用户把数据写入系统的速度快于系统输出数据的速度，或字符设备提供数据的速度快或慢于应用程序消耗数据的速度时，也需要建立缓冲区。

通常在内核空间开辟缓冲区，数据在缓冲区中缓冲后，再在用户缓冲区和设备之间传输。缓冲涉及大量复制操作，故对 I/O 性能会产生很大影响。

4. 设备分配和状态跟踪

根据设备的物理特性，操作系统制订分配策略，可采用静态分配、动态分配和虚拟分配。一些独占型设备，如 CD-ROM，同一时刻只能被一个进程使用。当用户发出设备使用的请求时，要求操作系统检查相应设备的使用状态，并根据其忙闲状况来决定接受或拒绝这一请求。一种简单的处理方法是，直接用 OPEN 打开相应设备的特别文件来进行申请，若设备不可用，则 OPEN 失败；当进程用完独占型设备时，应该关闭并释放之。

5. 错误处理和报告

I/O 设备出错是经常发生的，错误应尽可能在接近硬件的地方加以处理，如果控制器发现错误，应该设法纠正和加以解决。如果未能处理错误，再交给设备驱动程序。多数错误是与设备紧密相关的，驱动程序知道应如何对其处理（如重试、忽略或报警）。一种典型的错误是磁盘块受损导致不能读写，驱动程序在尝试若干次读写操作不成功后才会放弃。如果发现错误处理不成功，或不能处理错误时，可向独立于设备的 I/O 软件报错，此后这个错误的处理就与设备驱动程序无关。低层软件所不能处理的情况交给高层软件处理。在许多情况下，错误恢复可由低层软件透明地得到解决，而高层软件甚至不知道错误的存在。如果发现 I/O 编程出错，如无效操作码、无效设备或无效缓冲区地址，则返回一个错误码并向调用者报错；如果是在读关键系统数据结构（如磁盘位示图）时出错，则只能向操作员报告，打印出错提示信息并终止运行。诸如此类与设备

例如，网卡驱动通常有一个Packet Buffer指针，指向DRAM中的一个区域。

I/O 地址传送给控制器，向其发出读、写或控制命令，控制设备进行数据传输。控制器接收通道发来的命令后，检查设备的状态，若设备不忙，则告知通道释放 CPU，并开始 I/O 操作，向设备发出动作序列，设备执行相应的动作；之后，通道独立执行通道程序中的各条通道命令字，直到通道程序执行结束。本次 I/O 操作结束之后，通道向 CPU 发出 I/O 操作结束中断，再次请求 CPU 的干预。

3. I/O 结束阶段

通道发现 I/O 系统中出现通道结束、控制器结束、设备结束或其他能产生中断的信号时，就向 CPU 申请 I/O 中断。同时，把产生中断的通道号和设备号以及通道状态字存入主存固定单元，CPU 响应 I/O 中断后，暂停现行程序的执行，转向 I/O 中断处理程序。

5.4 缓冲技术

为了解决 CPU 与设备之间速度不匹配的矛盾，及协调逻辑记录大小与物理记录大小不一致的问题，提高 CPU 和设备的并行性，减少 I/O 操作对 CPU 的中断次数，放宽对 CPU 中断响应时间的要求，操作系统普遍采用缓冲技术。缓冲用于平滑两种不同速度的硬部件或设备之间的信息传输，在主存储器中开辟一个存储区，称为缓冲区，专门用于临时存放 I/O 操作的数据。

实现缓冲技术的基本思想如下：当进程执行写操作输出数据时，先向系统申请一个输出缓冲区，然后，将数据送至缓冲区，若是顺序写请求，则不断地把数据填入缓冲区，直到装满为止，此后，进程可以继续计算，同时，系统将缓冲区的内容写到设备上。当进程执行读操作输入数据时，先向系统申请一个输入缓冲区，系统将设备上的一条物理记录读至缓冲区，然后根据要求，把当前所需要的逻辑记录从缓冲区中选出并传送给进程。

用于上述目的的专用主存区称为 I/O 缓冲区。在输出数据时，只有在系统来不及腾空缓冲区而进程又要写数据时，它才需要等待；在输入数据时，仅当缓冲区为空而进程又要从中读取数据时，它才被迫等待。其他时间可进一步提高 CPU 和设备的并行性，设备和设备之间的并行性，从而提高系统效率。在操作系统的管理下，常辟出缓冲区以服务于各种设备，常用的缓冲技术有单缓冲、双缓冲和多缓冲。

5.4.1 单缓冲

单缓冲是最简单的缓冲技术，每当应用进程发出 I/O 请求时，操作系统在主存储器的系统区中开设一个缓冲区。

对于块设备输入，单缓冲机制的工作过程是：先从磁盘把一块数据读至缓冲区，假设花费时间 T；接着，系统把缓冲区中的数据送到用户区，设所消耗的时间为 M，由于此时缓冲区已空，系统可预读紧接着的下一块，大多数应用将使用邻接块，然后应用进程对这批数据进行计算，共耗时 C。若不采用缓冲技术，数据直接从磁盘传送到用户区，每批数据处理时间约为 $T+C$，而采用单缓冲，每批数据处理时间为 $\max(C,T)+M$，通常 M 远远小于 C 或 T，故速度会快很多。

T
M
C

T和C有并行的部分了。所以效率提高了。

对于块设备输出，单缓冲机制的工作方式类似，先把数据从用户区复制到系统缓冲区，应用进程可继续请求输出，直到缓冲区填满，由系统写到磁盘上。对于字符设备输入，缓冲区用于暂存用户输入的一行数据，在输入时，应用进程挂起，等待一行数据输入完毕；在输出时，应用进程将第一行数据送入缓冲区后继续执行，如果在第一个输出操作未腾空缓冲区之前，又有第二行数据要输出，应用进程等待。如果希望实现 I/O 的并行工作，如把输入设备中的数据输入并加工，再从输出设备上输出，必须引入双缓冲技术。

5.4.2 双缓冲

两个缓冲并行的工作

为了加快 I/O 操作的执行速度，实现 I/O 的并行工作和提高设备利用率，需要引入双缓冲。在输入数据时，首先从设备读出数据填充缓冲区 1，系统从缓冲区 1 把数据传送到用户区，应用进程便可对数据进行加工和计算；与此同时，从设备读出数据填充缓冲区 2。当缓冲区 1 为空时，再次从设备读出数据到缓冲区 1，系统又可以把缓冲区 2 的数据传送到用户区，应用进程开始加工缓冲区 2 中的数据。两个缓冲区交替使用，使 CPU 和设备、设备和设备之间的并行性进一步提高，仅当两个缓冲区都为空且进程还要提取数据时，它才被迫等待。粗略地估计传输和处理一块的时间，如果 $C<T$，输入操作比计算操作速度慢，这时由于 M 远远小于 T，故将磁盘上的一块数据传送到一个缓冲区期间（所花费时间为 T），计算机已完成将另一个缓冲区中的数据传送到用户区并对这块数据进行计算的工作，所以，一块数据的传输和处理时间为 T，即 $\max(C,T)$，显然，这种情况可保证块设备连续工作；如果 $C>T$，计算操作比输入操作速度慢，每当上一块数据计算完毕后，需要把一个缓冲区中的数据传送到用户区，所花费时间为 M，再对这块数据进行计算，所花费时间为 C，所以，一块数据的传输和处理时间为 $C+M$，即 $\max(C,T)+M$，显然，这种情况使得进程不必等待 I/O 操作。双缓冲技术使系统效率提高，但复杂性也随之增加了。

采用双缓冲读卡及打印可这样进行：第一张卡片读入缓冲区 1，在打印缓冲区 1 中数据的同时，又把第二张卡片读入缓冲区 2。缓冲区 1 打印完毕时，缓冲区 2 也刚好输入完毕，让读卡机和打印机交换缓冲。这样，I/O 设备就能够处于并行工作的状态。

5.4.3 多缓冲

采用双缓冲技术虽然能提高设备的并行工作程度，但在设备和处理进程速度不匹配的情况下仍不十分理想。举例来说，若输入设备的速度快于进程消耗数据的速度，则输入设备很快就会把两个缓冲区填满；反之，若进程处理数据的速度快于数据输入速度，很快会把两个缓冲区抽空，造成进程经常处于等待态。为了改善这种情况，获得较高的并行度，常常采用多缓冲所组成的循环缓冲技术。

操作系统从主存区域中分配一组缓冲区，每个缓冲区都有一个链接指针指向下一个缓冲区，最后一个缓冲区指针指向第一个缓冲区，组成循环缓冲，缓冲区的大小等于物理记录的大小，多缓冲的缓冲区是系统的公共资源，可供进程共享，并由系统统一分配和管理。缓冲区用途可分为输入缓冲区、处理缓冲区和输出缓冲区。为了管理各类缓冲区，执行各种操作，必须设计专门的

例如，网卡驱动缓冲区可以有报文描述符缓冲区和报文数据缓冲区。

[注释]：设备通常是Memory Based I/O。控制Reg和缓冲区被映射到CPU地址空间，例如通过PCI。读写控制寄存器和缓冲区通常与DRAM内存一样。通常设备地址map到高端地址区域。通信系统里，还要事先设置一些cache属性等，例如，non-cacheable。总之，通过系统总线控制器或者MMU存储子系统部件来调控。

http://en.wikipedia.org/wiki/RAID

The most common types today are RAID 0 (striping), RAID 1 and variants (mirroring), RAID 5 (distributed parity) and RAID 6 (dual parity).

RAID技术被广泛的用在数据容错方面。大多数现代操作系统都已经支持。另外，RAID也可以是软件实现方案。许多文件系统可以支持Software – RAID。

5.5.5 独立磁盘冗余阵列

独立磁盘冗余阵列(Redundant Array of Independent Disks,RAID)的概念在1987年由美国加利福尼亚大学提出,已得到业界的认可,被广泛应用于大中型计算机和计算机网络系统中,其目的是:增加容错性,获得高性能。通过在不同的磁盘上维护冗余数据来增加容错性;通过把数据分布到多个磁盘,采用并行存取以加快数据传输速率来获得高性能。其策略是:用一组容量较小的、独立的、可并行工作的磁盘组或阵列来代替单一的大容量磁盘,再加入冗余技术,数据能以多种方式组织和分布存储,于是,独立的I/O请求能够被并行处理,分布式的单个I/O请求也能并行地从多个磁盘同时得到处理,从而增加容错性,改进I/O性能。

在基本的组织方式中,有多种类型的RAID,其主要差别在于冗余信息数量(增加磁盘的数量)和容错性级别(可纠正错位的数目),及冗余信息是集中在一个磁盘上还是分散在多个磁盘上。具体类型列举如下。

(1) RAID0:连续的数据条带(每个条带可规定为1个或多个扇区)以轮转方式写到全部的磁盘上,然后,采用并行交叉方式存取,缩短I/O请求的排队时间,适用于大数据量的I/O请求,

RAID 0方案是数据块分拆在各个磁盘上。

混合（Nested）RAID，例如RAID10的意思是RAID1+RAID 0两种方案混合，不是Ten的意思。RAID10:先做Mirror镜像，然后再做一次Data Striping分布

但并无冗余校验功能，可靠性较差。

镜像

(2) RAID1：采用镜像盘备份所有数据来提高容错性，读请求拥有最短寻道时间，写请求可并行完成。其缺点是容量下降一半，成本很高。

(3) RAID2：采用数据字或字节方式交叉存放，并行存取以获得高性能，使用海明校验码，适合大量顺序数据的访问。由于使用多个冗余盘，其成本较高。基本上不用。

(4) RAID3：是 RAID2 的简化版本，其差别是它仅使用一只冗余盘，采用奇偶校验技术。

(5) RAID4：采用独立存取磁盘阵列，数据条带交叉存放，访问请求可并行地获得满足，适合要求较高 I/O 请求速度的应用场合，使用一只冗余盘存放奇偶校验码。

(6) RAID5：与 RAID4 的组织方式类似，但奇偶校验码循环分布在每个盘上，使得容错性更好。

多常用。

(7) RAID6：采用双重冗余技术，即使有两个数据磁盘发生错误，也可以重新生成数据。

5.5.6 提高磁盘 I/O 速度的方法

通常为磁盘设置数据缓冲区高速缓存，这样能够显著缩短等待磁盘 I/O 的时间。下面所介绍的一些方法也能有效地提高磁盘 I/O 速度。

1. 提前读

可以理解为是利用了空间局部性原理。

对于采用顺序方式所访问的文件数据，在读当前盘块时已知下次要读出的盘块地址，因此，可在读当前盘块的同时，提前把下一盘块的数据也读入磁盘缓冲区。这样一来，当下次要读盘块中的那些数据时，由于已经提前把它们读入缓冲区，便可直接使用，从而缩短读数据的时间，相当于提高磁盘 I/O 速度。"提前读"功能已被许多操作系统如 UNIX、OS/2、Windows 所采用。

2. 延迟写

懒惰算法。Lazy Algorithm

在执行写操作时，磁盘缓冲区中的数据本应立即写回磁盘，但考虑到此缓冲区中的数据不久之后会再次被进程访问，因此，并不立即将缓冲区中的数据写盘，而是把它挂在空闲缓冲区队列的末尾。随着空闲缓冲区的使用，存有输出数据的缓冲区不停地向队列头移动，直至移动到空闲缓冲区队列之首，当再有进程申请缓冲区且分配到此缓冲区时，才把其中的数据写到磁盘上，于是这个缓冲区可作为空闲缓冲区分配。只要存有输出数据的缓冲区还在队列中，任何对此数据的访问均可直接从中找回，不必再去访问磁盘，这样做可以减少磁盘 I/O 的次数。同样，在 UNIX、OS/2 和 Windows 系统中也都采用这一技术。

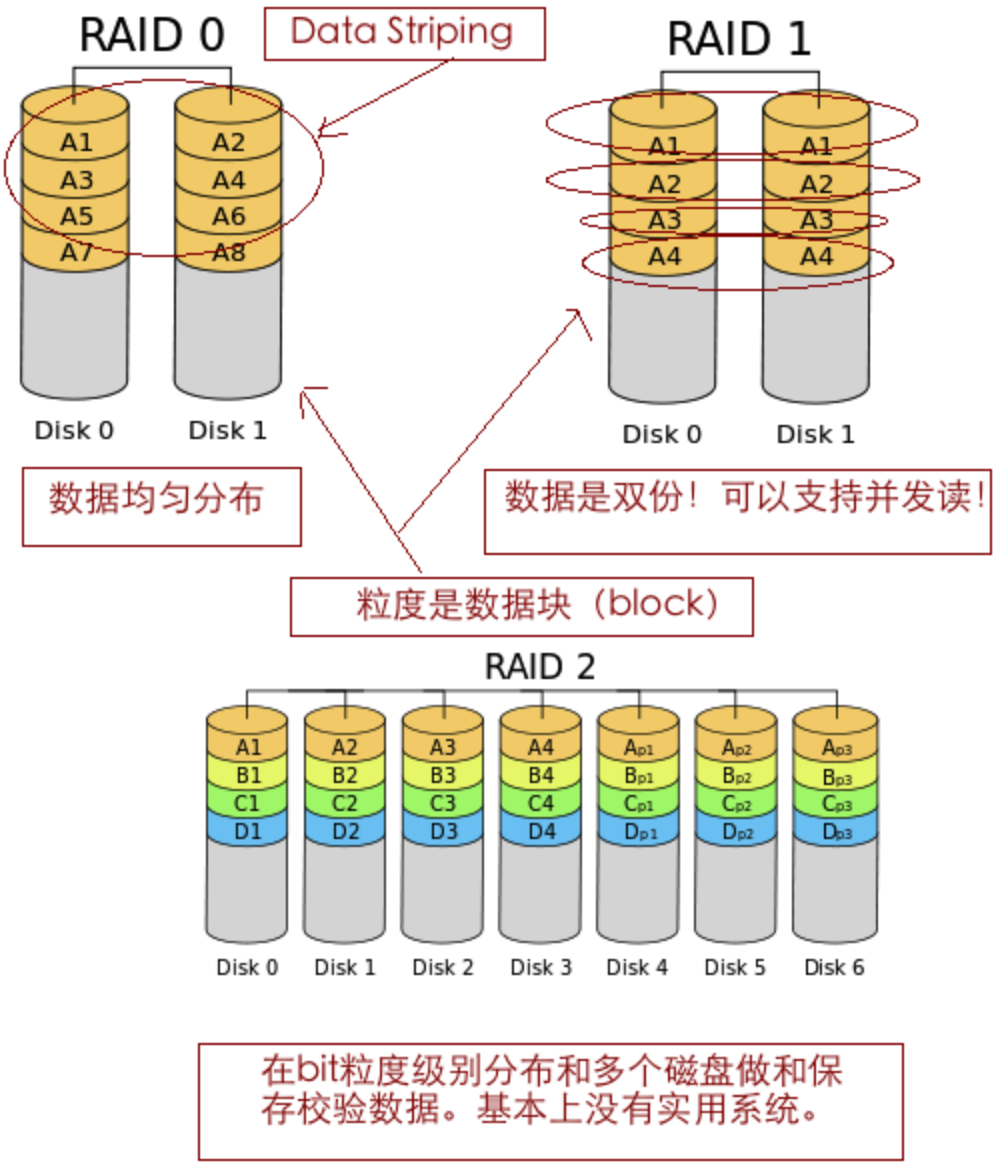
UNIX/Linux 提供两种读盘方式和三种写盘方式。"正常读"是指把磁盘块信息读入主存缓冲区；"提前读"是指在读磁盘当前块时，把下一磁盘块也读入主存缓冲区；"正常写"是指把主存缓冲区中的信息写至磁盘块，且写进程应等待写操作完成；"异步写"是指写进程无须等待写盘结束就可返回工作；"延迟写"是指仅在缓冲区首部设置"延迟写"标志，然后，释放此缓冲区并将其链入空闲缓冲区链表的尾部，当其他进程申请到此缓冲区时，才真正把缓冲区信息写回磁盘块。

3. 虚拟盘

虚拟盘是指用主存空间去仿真磁盘，又叫 RAM 盘。虚拟盘的设备驱动程序可接收所有标

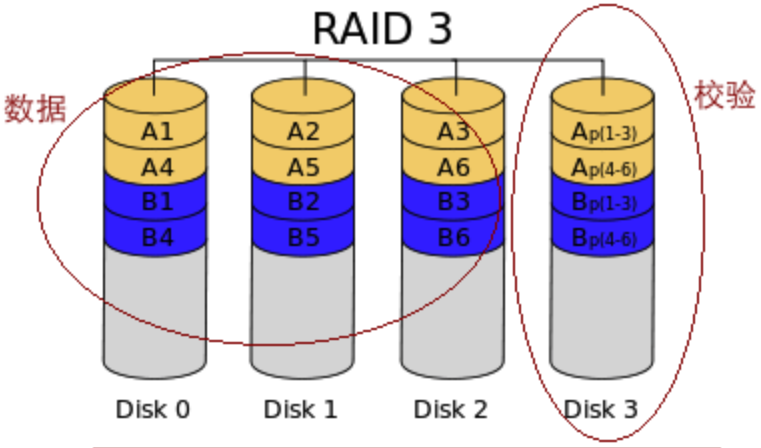
类似CPU Cache的Write Back。不要立刻写回。

RAID图解 (1)

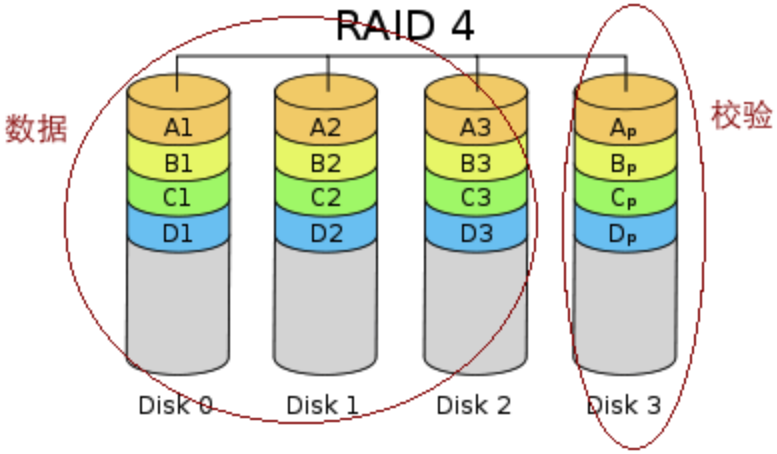


在bit粒度级别分布和多个磁盘做和保存校验数据。基本上没有实用系统。

RAID图解（2）

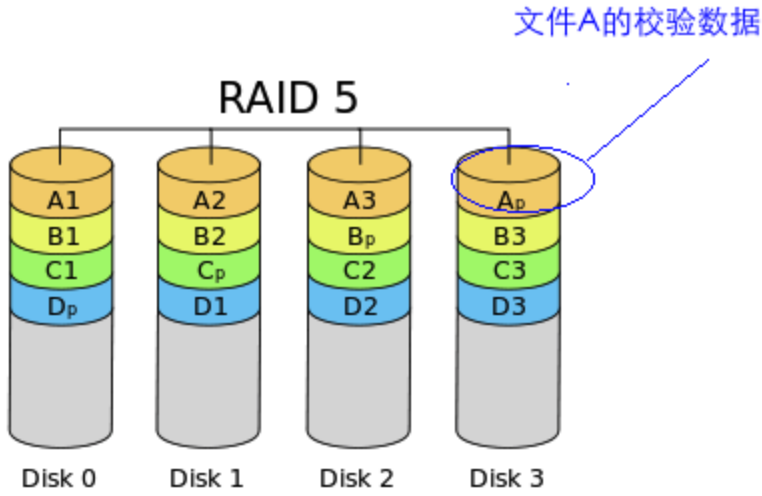


粒度是字节（Byte）。用一个专门的磁盘存放校验纠错数据。

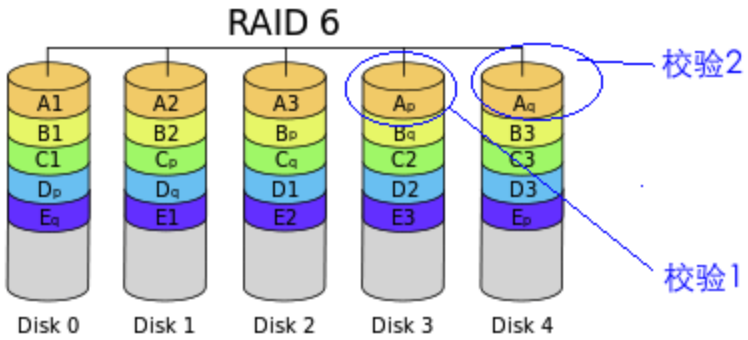


粒度是Block块。可以支持并发。例如，对A1和B2的READ。

RAID图解 (3)



粒度是Block；不用专门的校验磁盘。校验数据分布在数据磁盘阵列里。



RAID 6可以坏两个磁盘，仍然正常工作。

粒度是Block；不用专门的校验磁盘。与RAID 5比较，RAID 6用两个校验磁盘。

Level	Description 粒度和相关属性	number of drives 至少的磁盘数目	Fault tolerance 可以容错的磁盘数目
RAID 0	Block-level striping without parity or mirroring	2	0 (none)
RAID 1	Mirroring without parity or striping	2	n−1 drives
RAID 2	Bit-level striping with dedicated Hamming-code parity	3	One drive
RAID 3	Byte-level striping with dedicated parity	3	One drive
RAID 4	Block-level striping with dedicated parity	3	One drive
RAID 5	Block-level striping with distributed parity	3	One drive
RAID 6	Block-level striping with double distributed parity	4	Two drives

[注释：] RAID磁盘阵列是通过多个小的廉价的磁盘的组合，可以比一个单一的昂贵的大磁盘提供更好的性价比。例如并发的操作和容错。RAID 0是数据分布；RAID 1是备份镜像；之后的都是有校验纠错磁盘的。RAID 6是基于数据块和分布式存放纠错，可以支持2个磁盘坏还正常工作。

"Everything is a file，except process"

第六章

除了进程，其他的对象都是文件。

文件管理

文件系统是操作系统中负责存取和管理信息的模块，它采用统一方法管理用户信息和系统信息的存储、检索、更新、共享和保护，并为用户提供一整套行之有效的文件使用及操作方法。"文件"这一术语不但反映用户概念中的逻辑结构，而且同存放它的辅助存储器的存储结构紧密相关，所以，必须从逻辑文件和物理文件两个侧面来观察文件。对于用户而言，可按照需要并遵循文件系统的规则来定义文件信息的逻辑结构，由文件系统提供"按名存取"方式来实现对文件信息的存储和检索；对于系统而言，必须采用特定的数据结构和有效算法，实现文件的逻辑结构到存储结构的映射，实现对文件存储空间和文件信息的管理，提供多种存取方法。例如，用户希望与具体的存储硬件无关，使用路径名、文件名、文件内位移就可执行数据的读、写、修改、删除操作；而作为实现这些功能的文件系统来说，其工作与存储硬件紧密相关，是将用户的文件操作请求转化为对磁盘上的信息按照所在的物理位置进行寻址、读写和控制。所以，文件系统的功能就是在逻辑文件与物理文件、逻辑地址与物理地址、逻辑结构与物理结构之间实现转换，使得存取速度快、存储空间利用率高、数据可共享、安全可靠性好。

是一个逻辑概念。

文件系统的主要功能有：文件的按名存取，实现从逻辑文件到物理文件的转换；文件目录的建立和维护；文件的查找和定位；文件存储空间的分配和管理；提供文件的存取方法和文件存储结构；实现文件的共享、保护和保密；提供一组易用的文件操作和命令；提供与设备管理交互的统一接口。

6.1 文　件

6.1.1 文件概念

早期计算机系统中，用户自行管理辅助存储器，按照物理地址安排信息，记录信息的分布情况，组织数据输入/输出，繁琐复杂，极易出错，可靠性差。大容量直接存取存储器的问世为建立文件系统提供良好的物质基础，多道程序和分时系统的出现，使得多个用户及操作系统都要共享大容量辅助存储器。因而，现代操作系统中都配备文件系统，以适应系统管理和用户使用信息资源的需要。对计算机系统中信息资源的管理形成了操作系统的文件系统。

文件是一个抽象。通过MetaData控制信息，把物理数据块，有效的，逻辑相关联的联系在一起。Metadata：关于数据的数据。Data的Data。

设备驱动程序其实就是要完成相应的open，read，write等文件系统层面的APIs

成：只读文件、读写文件和不保护文件；按信息流向分成：输入文件、输出文件和输入输出文件；按存放时限分成：临时文件、永久文件、档案文件；按数据类型分成：源程序文件、目标文件和可执行文件；按设备类型分成：磁盘文件、磁带文件、软盘文件。此外，还可按文件的逻辑结构或物理结构进行分类。在现代操作系统中，不但信息组织成文件，对设备的访问也都基于文件进行。例如，打印一批数据就是向打印机设备文件写数据，从键盘接收一批数据就是从键盘设备文件读数据。

UNIX/Linux 操作系统支持以下不同类型的文件。

(1) 普通文件：源程序文件、数据文件、目标代码文件及操作系统文件、库文件、实用程序文件都是普通文件，它们通常存储在磁盘上。

(2) 目录文件：是由文件目录所构成的用来维护文件系统结构的系统文件。普通文件的查找依赖于目录文件，由于它也是由字符信息所组成的文件，故可进行与普通文件类似的读写等各种目录操作。

(3) 特别文件：指各种外部设备文件，又可分为：块设备文件，如存放在磁盘或光盘等块设备上的文件；字符设备文件，如终端、打印机等设备文件；FIFO 命名管道文件，socket 套接字文件。

一般来说，普通文件包括 ASCII 文件或者二进制文件。ASCII 文件由多行正文组成，在 Windows 等系统中，每行均以回车换行结束，整个文件以 Ctrl + Z 结束；在 UNIX 等系统中，每行以换行结束，文件以 Ctrl + D 结束。ASCII 文件的最大优点是可以原样显示和打印，也可用通常的文本编辑器进行编辑。另一种正规文件是二进制文件，它具有一定的内部结构，组织成字节流，如可执行文件是指令和数据流，记录式文件是逻辑记录流。块设备文件和字符设备文件合称特殊文件，它们与普通文件类似，也需要查找目录、验证访问权限、进行读写操作等。把所有 I/O 设备统一在文件系统下，有利于系统管理，方便用户使用。

6.1.4 文件属性

Attributes，或者MetaDta

大多数操作系统设置专门的文件属性用于文件的管理控制和安全保护，它们虽非文件的信息内容，但对于系统的管理和控制是十分重要的。这组属性包括：

(1) 文件基本属性：文件名称和扩展名、文件属主 ID、文件所属组 ID 等。

(2) 文件类型属性：如普通文件、目录文件、系统文件、隐式文件、设备文件、pipe 文件、socket 文件等。也可按文件信息分为：ASCII 码文件、二进制码文件等。

(3) 文件保护属性：规定谁能够访问文件，以何种方式访问。常用的文件访问方式有可读、可写、可执行、可更新、可删除等；上锁标志和解锁标志；有的系统还为文件设置口令，用做保护。

(4) 文件管理属性：如文件创建时间、最后访问时间、最后修改时间等。

(5) 文件控制属性：逻辑记录长、文件当前长、文件最大长，关键字位置、关键字长度、信息位置、文件打开次数等。

文件保护属性用于防止文件被破坏，包括两个方面：一是防止因系统发生崩溃所造成的文件破坏；二是防止文件主和其他用户有意或无意的非法操作所造成的文件不安全性。为了防止系

每一个文件对应一个iNode。

数据块
数据块
数据块

编排的目录便于查找，防止冲突，目录的检索方便、迅速。

文件系统的信息空间可认为是由一系列逻辑块所组成的，每个逻辑块的长度为 512 B 的整数倍数。不同文件系统的逻辑块大小允许不同，在系统生成时指定。至于逻辑块号到物理块号的转换则由独立于设备的操作系统软件负责，这些逻辑块被用于存放文件数据、文件目录或文件管理信息。UNIX/Linux 系统采用巧妙的文件目录建立方法，能够减少检索文件所需访问的磁盘物理块数。FCB 中的文件名和其他管理信息分开，其他信息单独组成一个数据结构，称为索引节点 inode，此索引节点的位置由 inode 号标识。于是，目录项中仅剩下 14 B 的文件名和 2 B 的 inode 号，称为基本目录项。因此，一个磁盘块可以存放几十个基本目录项，系统对于由文件目录项组成的目录文件和普通文件同等对待，均存放在磁盘中，文件系统中的每个文件都有一个磁盘 inode 与之对应，这些 inode 被集中存放于磁盘上的 inode 区。FCB 对于文件的作用，犹如 PCB 对于进程的作用，集中这个文件的所有相关信息，找到 inode，就能获得此文件的必要信息。每个磁盘 inode 结构使用 32 B 或 64 B，包含如下信息：文件长度及文件在存储设备上的物理位置、文件主标识、文件类型(普通/目录/特别/管道文件)、存取权限、文件链接数(共享数)、文件大小(字节数)、文件创建、访问和修改时间，及 inode 节点是否空闲，其中的 addr[40]数组用于存放文件数据或基本目录项所在的磁盘块号的索引表，每 3 个字节记录一个磁盘块号，因此索引表共 13 项。

由于磁盘 inode 记录文件的属性和相关信息，文件访问过程中会频繁地用到它，不断来回于主辅存之间引用它，当然是极不经济的。为此，UNIX/Linux 在系统所占用的主存区内开辟一张主存索引节点表，又称活动 inode 表，含有 100 个表项，每个表项称为一个活动 inode。磁盘 inode 反映文件的静态特性，活动 inode 还需反映文件的动态特性，为此增加部分信息，包括文件活动标志 i_flag(占用/修改/安装点/上锁位等)、共享索引节点数 i_count、索引节点所在设备名 i_dev、索引节点号 i_number、文件链接数 i_nlink、文件属性 i_mode 及空闲链和占用链哈希表指针。当访问某文件时，若在活动 inode 表中找不到其 inode，就申请一个空闲活动 inode，把磁盘 inode 内容复制给它，随之就可用来控制文件的读写。当用户关闭此文件后，活动 inode 的内容被回写到对应的磁盘 inode 中，然后释放活动 inode 以供它用。把 FCB 的主要内容与索引节点号分开，不仅能够加快目录检索速度，而且，便于实现文件共享。

UNIX System V 对文件目录项的内容进行扩充，共有 4 个部分：inode 号(4 B)、本记录长度、文件名长度和文件名(255 B)。

6.2.2 层次目录结构

最简单的文件目录是一级目录结构，所有 FCB 排列在一张线性表中，其缺点是文件重名和文件共享问题难以解决。实际上，所有文件系统都支持多级层次结构，根目录是唯一的，每一级目录可以是下一级目录的说明，也可以是文件的说明，从而形成树状目录结构。图 6.1 给出 Linux 和 Windows 目录层次结构，它是一棵倒置的有根树，树根是根目录；从根向下，每个树枝是子目录；而树叶是文件。树状多级目录结构有许多优点，较好地反映现实世界中具有层次关系的数据集合，确切地反映系统内部文件的分支结构；不同的文件可以重名，只要它们不位于同一末

一个iNode数据结构里，有多个pointer，指向物理磁盘数据块，例如，15个。

本身是关于home目录的一个数据块。　数据块里，内容是目录。

否则仅仅把相关记录从父目录中删除。

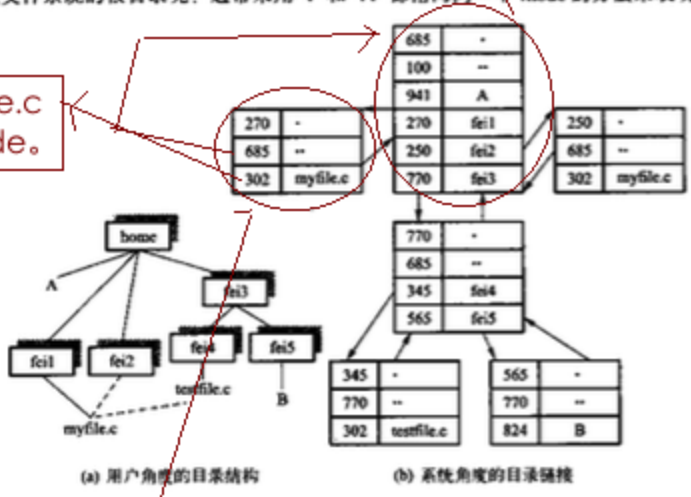
Linux 系统支持多父目录，但其中一个是主父目录，它是文件拥有者，且文件被物理地存储在此目录下，其他次父目录通过 link 方式来连接和引用文件，允许任一父目录删除共享文件。图 6.1(a)中便示例这种文件共享的情形，文件/home/fei1 是 myfile.c 的主父目录(图中以实线表示)，/home/fei2 和/home/fei3/fei4 均为文件 myfile.c 的次父目录(图中以虚线表示)。

Windows 实现被称作"快捷方式"的多父目录连接，"快捷方式"是一些指向不同文件夹(子目录)和菜单之间任意复制和移动的文件及文件夹的指针，删除"快捷方式"就是删除其所对应的指针。

6.2.3 文件目录的检索

每个目录在创建时都自动含有两个特殊目录项，"."项指出目录自身的 inode 入口，".."项指出其父目录的 inode 入口，如图 6.2 所示。图 6.2(a)是用户概念中的目录结构，在系统中是按图 6.2(b)来实现目录链接的。父目录包含子目录意味着包含一个指向子目录的 inode 链接"."，反之，子目录具有父目录也通过指向 inode 来实现，它有一项".."，这是父目录的保留名字。如何知道是文件系统的根目录呢？通常采用"."和".."都指向同一个 inode 的方法来表明。

302是myfile.c对应的iNode。



(a) 用户角度的目录结构　(b) 系统角度的目录链接

图 6.2 不同角度的目录结构

执行打开文件操作时，根据用户所提供的文件路径名，从根目录或当前工作目录开始，逐级查找路径名中的各子目录名，用其作为索引，逐层搜索各级目录文件，最后找到相匹配的文件目录项。下面是目录查找的例子，假设应用进程要打开文件/home/fei1/myfile.c，文件系统开始搜

iNode 270对应的数据block的内容

索，首先，遇到根目录"/"，它通常被存放在磁盘的固定盘块中，根据活动 inode 表中根目录的活动 inode，把它作为当前工作索引节点并将其第一个物理块读入主存缓冲区，接着读入路径的第一个分量字符串 home；文件系统对根目录的内容进行搜索，若找不到，则依次读入第二、第三个物理块，…，进行比较，直至找到 home 的 inode 号；检查活动 inode 表，若找不到 home 的 inode，为其分配一个活动 inode，由于每个 inode 位于磁盘上已分配好的固定位置，可从磁盘 home 的 inode 中装入其内容；否则直接查找 home 的活动 inode，通过属性查明 home 为子目录，经核对符合访问权限，把它作为当前工作索引节点，读入路径的第二个分量字符串 fei1，子目录 home 对应的 inode 为 685，从目录文件第一个物理块中可找到子目录 fei1 的 inode 为 270，类似地读入子目录 fei1 目录文件的物理块便能找到 myfile.c 的 inode 号为 302；文件系统为此文件在活动 inode 表中分配一个活动 inode，从 myfile.c 的磁盘 inode 中装入内容。中间任何一步出错都会返回错误码，由于路径名分析完毕，这时修改活动 inode 的有关内容，打开文件的目录查找操作到此结束。

类似于文件操作，文件系统也提供一系列处理文件目录的系统调用和操作命令，有的系统甚至允许应用程序像对待文件一样打开、读写和操作目录。与文件目录有关的操作有：创建目录、删除目录、打开目录、关闭目录、读目录、列目录、改变目录、移动目录、重命名目录、改变保护、链接和删除链接目录等。

目录本身是一个文件，对应一个iNode和数据块。

6.3 文件组织与数据存储

6.3.1 文件的存储

目前广泛使用的文件存储介质是磁盘、光盘和磁带，一盘磁带、一张光盘、一个硬盘分区都称为一卷。卷是存储介质的物理单位，块是存储介质上连续信息所组成的一个区域，也称物理记录。块是主存储器和辅助存储器进行信息交换的物理单位，每次总是交换一块或整数块信息。决定块的大小要考虑到用户使用方式、数据传输效率和存储设备类型等多种因素，不同类型的存储介质，其块的长短各不相同；同一类型的存储介质，其块的长短也可以不同。有些外部设备出于启停机械动作的要求或识别不同块的特殊需要，相邻块之间必须留有间隙，间隙是块之间不记录用户代码信息的区域。

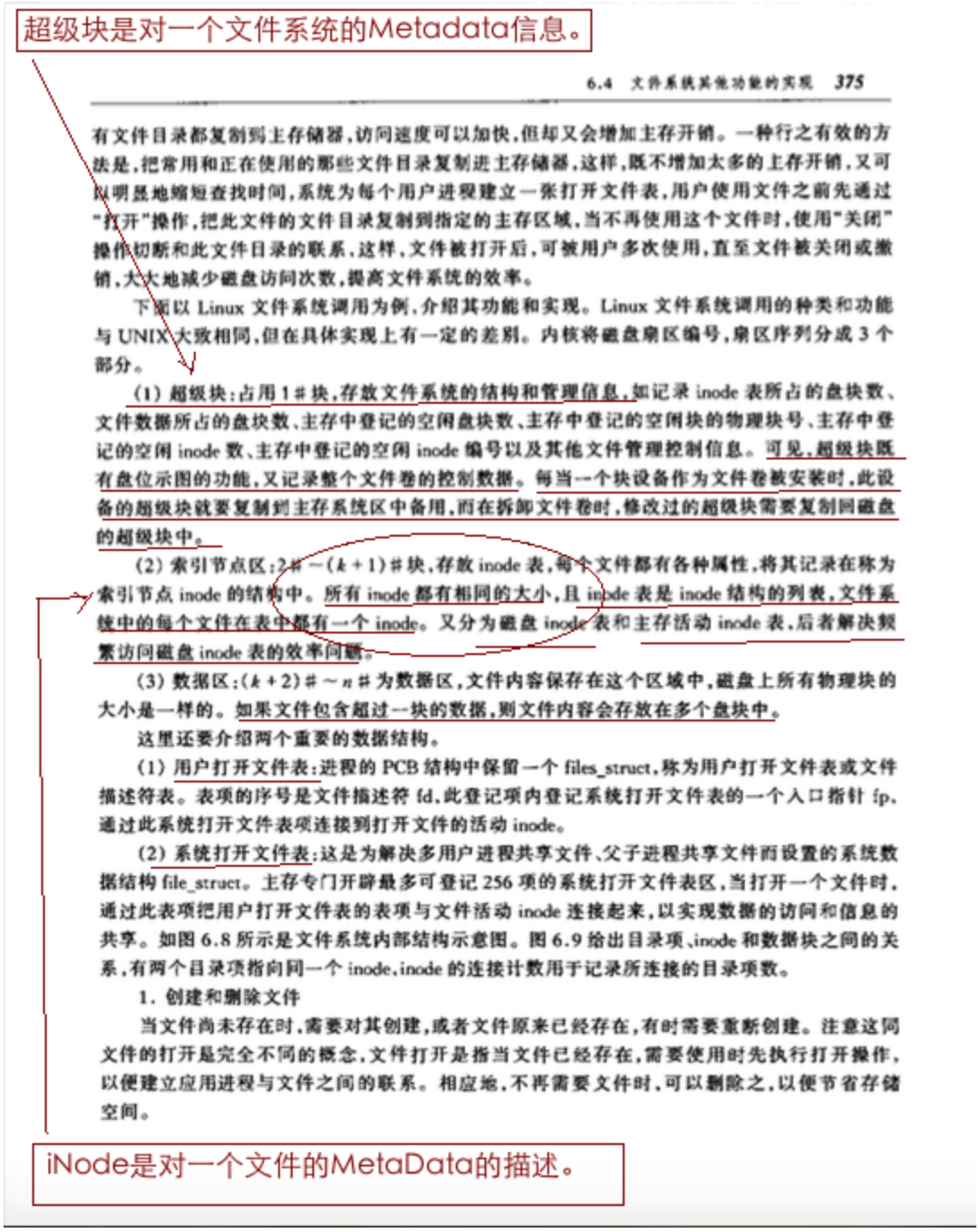
通过一层层的读目录文件的数据块（通过iNode里指向数据块的指针），最后可以找到某个具体文件的iNode ＃。

6.3.2 文件的逻辑结构

1. 流式文件和记录式文件

文件组织是指文件中信息的配置和构造方式，应该从文件的逻辑结构和组织以及文件的物理结构和组织两方面加以考虑。文件的逻辑结构和组织是从用户的观点出发，研究用户概念中抽象的信息组织方式，这是用户所能观察到的、可加以处理的数据集合。由于数据可独立于物理环境构造，故称为逻辑结构，相关数据的集合构成逻辑文件。系统提供若干操作以便使用者构造文件，这样，用户不必顾及文件信息的物理结构，利用文件名和有关操作就能存储、检索和处理文

Stream Based， Record Based

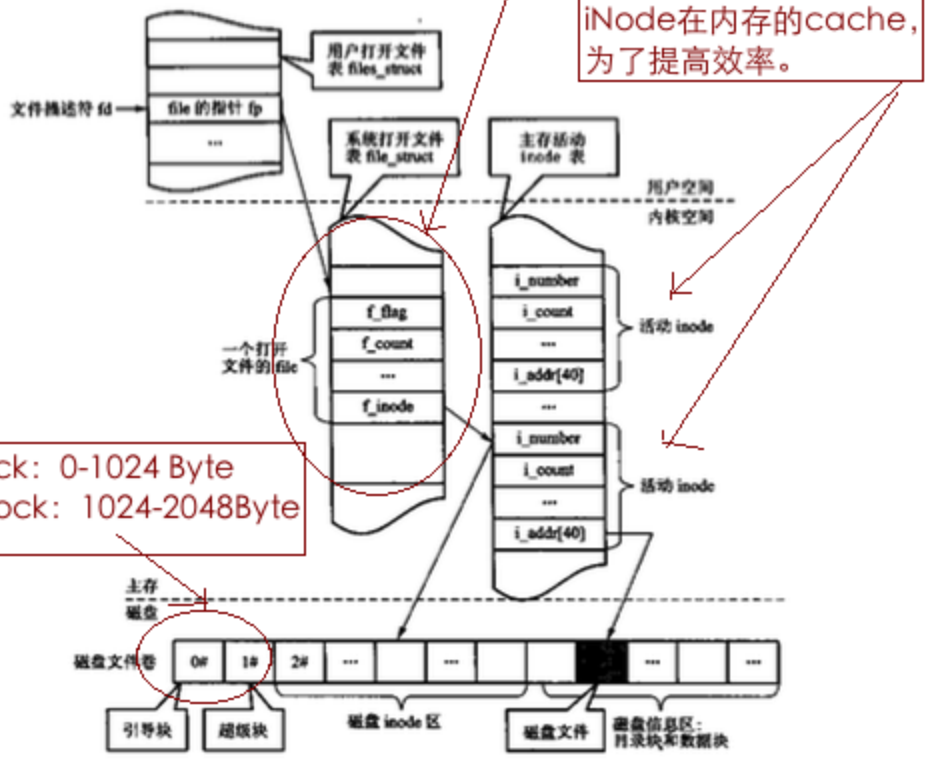


[注释] ： Unix - like文件系统一个重要概念是："Everything is a file,except process". 除了进程都是文件。如何把进程与文件联系起来？抽象化。通过进程文件打开表，指向系统文件打开表。open（文件）调用返回的id其实就是进程文件打开表的数组下标。其中一个重要数据结构是inode：每个文件对应和只对应一个inode。1:1。通过inode数据结构里的指针可

以定位文件的数据块s。通过inode的#，可以在磁盘里通过读写相应的inode数据。第一个inode数据跟在Boot（1024Byte），SuperBlock（1024）后面。inode数据结构里除了指向数据块的指针（15个）之外，还含有许多MetaData控制信息。MetaData是文件系统常用的一个词汇。MetaData就是“Data that Describes Data”。描述数据的数据。这个词来源于以前图书馆里的卡片，卡片上的数据除了书的简单信息之外，有描述图书馆的书在几楼，那个书架上的索引，从而同学们可以在相应的书架上找到想阅读的书。

通过系统打开表，可以实现多个进程打开同一个文件（iNode），从而实现共享读写。

iNode在内存的cache，为了提高效率。



Boot Block：0-1024 Byte
Super Block：1024-2048Byte

图 6.8 文件系统内部结构

在disk的固定区域。可以定位并读写出来。

(1) 创建文件

系统调用的 C 语言格式为：

```
fd = create(filenamep, mode);
```

其中，参数 filenamep 是指向所要创建的文件路径名的字符串指针；参数 mode 是文件所具有的存取权限，在文件成功创建之后，权限被记录在相应索引节点的 i_mode 中。变量 fd 是创建成功后系统所返回的文件描述符，即用户打开文件表中相应文件表项的序号。由此可见，create 兼具文件的"打开"功能，随后执行文件写系统调用，就可使用这个 fd 进行写操作。例如，要创建文件的路径名是/home/home1/newfile，可使用如下系统调用：

两个文件指向相同的数据。例如，softlink的case。

iNode里有多个指针指向数据块。

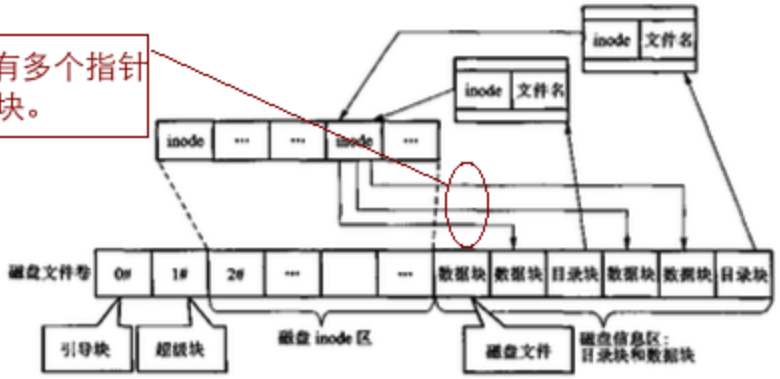


图 6.9 目录项、inode 和数据块之间的关系

fd = create("/home/home1/newfile",0775);

下面简述其实现过程，假定文件是首次创建，即在执行之前，文件尚未存在。

① 为新文件 newfile 分配磁盘 inode 和活动 inode，并把 inode 号与文件分量名 newfile 组成新的目录项，记录到目录路径/home/home1 的目录文件中。在这一过程中，需要执行目录检索程序。

② 在新文件所对应的活动 inode 中置初值，包括把存取权限 i_mode 置为 0775，连接计数 i_link 置为"1"，等等。

MetaData

③ 为新文件分配用户打开文件表项和系统打开文件表项，置系统打开文件表项的初值，包括在 f_flag 中置"写"标志，读写位移 f_offset 清 0，等等；把用户打开文件表项、系统打开文件表项及 newfile 所对应的活动 inode 用指针连接起来，最后，把文件描述符 fd 返回给调用者。

由于上述步骤中兼有文件"打开"功能，因此在以后的操作中，无须执行"打开"操作即可进行读写。

(2) 删除文件

删除的任务是把指定文件从其所在的目录文件中去除。如果没有连接的用户，即如果在执行删除之前 i_link 为"1"，还要把这个文件所占用的存储空间释放。文件删除系统调用的形式是 unlink(filenamep)，其中参数与 create 中的含义相同，在执行删除操作时，必须要求用户对文件具有"写"操作权。

2. 打开和关闭文件

文件在使用之前必须先"打开"，以建立进程与文件之间的联系，而文件描述符唯一地标识这样一种连接，其任务是把文件的磁盘 inode 复制到主存的活动 inode 表中，同时建立一个独立的读写文件数据结构，即系统打开文件表的一个表项。另一方面，活动 inode 表的大小受到主存容

设置初值，为共享做好准备。

量的限制，这就要求用户一旦不再对文件进行操作时，应立即释放相应的活动 inode，以便让其他进程使用，这就是"关闭"文件的主要功能。

(1) 打开文件的调用形式为：

fd = open(filenamep,mode);

其中，mode 是打开方式，表示打开后的操作要求，如读(0)、写(1)或读写(2)，其余参数的意义与 create 中的相同。打开操作的实现过程如下。

① 检索目录，要求打开的文件应该是已经创建的文件，它应登记在文件目录中，否则就会出错。在检索到指定的文件之后，就将其磁盘 inode 复制到活动 inode 表中。

② 把参数 mode 所给出的打开方式与活动 inode 中在创建文件时所记录的文件访问权限相比较，如果非法，则此次打开操作失败。

③ 当"打开"合法时，为文件分配用户打开文件表项和系统打开文件表项，并为系统打开文件表项设置初值，通过指针建立表项与活动 inode 之间的联系，然后把文件描述符 fd 返回给调用者。

需要指出的是，如果在执行这一调用之前，其他用户已打开同一文件，则活动 inode 表中已有此文件的 inode，于是，不用执行第①步中复制 inode 的工作，而仅把活动 inode 中的计数器 i_count 加 1 即可。在此 i_count 反映通过不同的系统打开文件表项来共享同一活动 inode 的进程数目，它是以后执行文件关闭操作时，活动 inode 能否被释放的依据。

(2) 文件的关闭

文件使用完毕，执行关闭系统调用，切断应用进程与文件之间的联系。其调用形式为 close(fd)。显然，要关闭的文件先前已打开，故文件描述符 fd 一定存在，其执行过程如下。

① 根据 fd 找到用户打开文件表项，再找到系统打开文件表项；释放用户打开文件表项。

② 把对应的系统打开文件表项中的 f_count 减 1，如果其值不为 0，说明进程族中还有进程正在共享它，不用释放此表项直接返回；否则释放此表项，并找到与之连接的活动 inode。

③ 把活动 inode 中的 i_count 减 1，若其值不为 0，表明其他进程正在使用此文件，直接返回；否则，把此活动 inode 的内容复制回文件卷上的相应磁盘 inode 中，释放此活动 inode。

可见，f_count 和 i_count 分别反映进程动态地共享一个文件的两种方式，前者反映不同进程通过同一个系统打开文件表项共享一个文件的情况；而后者反映不同进程通过不同系统打开文件表项共享一个文件的情况。通过这两种方式，进程之间既可用相同的位移指针 f_offset，也可用不同的位移指针 f_offset 共享同一个文件。

3. 文件的读和写

文件的读和写是文件最基本的操作，"读"是指文件的内容读入用户数据区中，"写"是指把用户数据区中的信息写入文件中。从文件的哪个逻辑位置读入数据，或者把数据写入文件的哪个逻辑位置，均由系统打开文件表中的 f_offset 决定。

(1) 读文件

nr = read(fd,buf,count);

第一种情况：父子进程；dup或者fork调用。
第二种情况：完全不同的进程打开同一个文件。不共享相同的“系统文件打开表项”。

home/fei3/fei4/testfile.c。图 6.2(b)中相对应的目录链接中有 3 个指针指向文件 myfile.c 的 inode(inode 号为 302)。

文件解除链接的调用形式为 unlink(namep)。实际上,UNIX/Linux 中解除链接与文件删除所执行的是同一段系统调用代码。删除文件是从文件属主角度来讲的,而解除文件链接是从共享文件的其他用户角度来讲的。无论删除还是解除链接,都要删去对应的目录项,把 i_nlink 值减 1。

在文件的静态链接中,为了反映共享同一文件的用户数,在每个索引节点中设立一个变量 i_nlink。当文件第一次创建时,i_nlink 值为 1,以后,每次链接时就把 i_nlink 值加 1。当用户删除文件(对于文件主)或解除链接(对于其他用户)时,就把此计数值减 1,直到发现结果为 0 时,才释放文件的物理存储空间,从而真正删除这个文件。

2. 文件动态共享

文件动态共享是指系统中不同的应用进程或同一用户的不同进程并发地访问同一文件,这种共享关系只有当进程存在时才可能出现,一旦进程消亡,其共享关系也就随之消失。在 UNIX/Linux 系统中,文件打开后,它的 inode 就在活动 inode 表中。活动 inode 表是整个系统公用的,为了使用户掌握当前使用文件的情况,系统在每个进程的 PCB 中设立用户打开文件表,并通过它与各自打开文件的活动 inode 联系。 中间还有个“系统打开文件表”

由于 UNIX/Linux 系统中的文件采用无结构字符流序列,因此,文件的每次读写都要由一个位移指针指出所要读写的位置。现在的问题是:若一个文件可被多个进程所共享,那么,应让多个进程共用同一个位移指针,还是应让各个进程具有各自的读写位移指针呢?下面分两种情况进行讨论。

在 UNIX/Linux 系统中,可以动态地创建进程,被创建的进程继承父进程的一切资源,包括 PCB 中用户打开文件表中所指定的打开文件。同一用户的父、子进程往往要协同完成同一任务,若使用同一读/写位移指针,那么,当一个进程改变它时,另一个进程就能够感觉到它的变化。这样,使父、子进程很容易同步地对文件进行操作。因此,这个位移指针似乎适合放在相应文件的活动 inode 中。此时,当执行 fork 函数创建子进程时,父进程的 PCB 被复制到子进程的 PCB 之中,使得两个进程的打开文件表指向同一活动 inode,达到共享同一位移指针的目的。如图 6.11 所示是共享位移指针的文件共享。

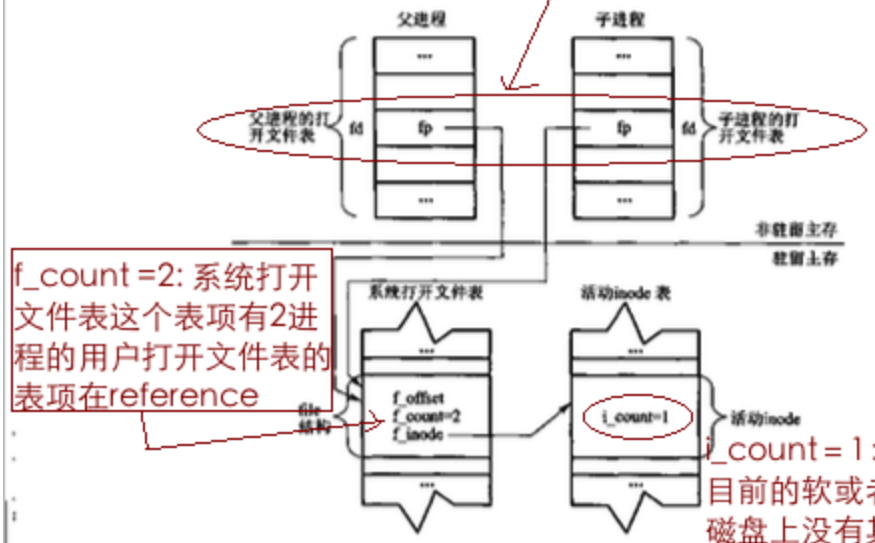
另一方面,若一个文件为两个以上的用户所共享,每个用户必然都希望能独立地读写此文件,彼此互不干扰。这时不能只设置一个读写位移指针,而必须为每个应用进程分别设置读写位移指针。据此,位移指针不应放在相应文件的活动 inode 中,而需独立设置。这样,当一个进程读写文件并修改位移指针时,另一个进程的位移指针不会随之改变,从而,两个进程能够独立地访问同一个文件。如图 6.12 所示是不共享指针的文件共享。

在上述两种动态共享方式中,对读写位移指针的位置设置和数目是不同的。为了解决这个矛盾,系统建立一张系统打开文件表,在此表的每个表项中,除了含有读写位移指针 f_offset 之外,还有文件的访问计数 f_count、读写标志 f_flag 和指向对应文件的活动索引节点指针 f_inode,其中 f_count 指出使用同一系统打开文件表目的进程数目,它是系统打开文件表目资源能否被释

两个不同进程的用户打开文件表项可以指向同一个“系统打开文件表”表项,从而共享同一个文件的读写和当年文件读写指针的偏移量。

例如，创建子进程时，CLONE_FILES，
子进程完全继承了父进程的用户打开文件表的内容。

放的标志。用户打开文件表、系统打开文件表和活动 inode 表之间的关系已在图 6.11 和图 6.12 中表明。



f_count =2: 系统打开文件表这个表项有2进程的用户打开文件表的表项在reference

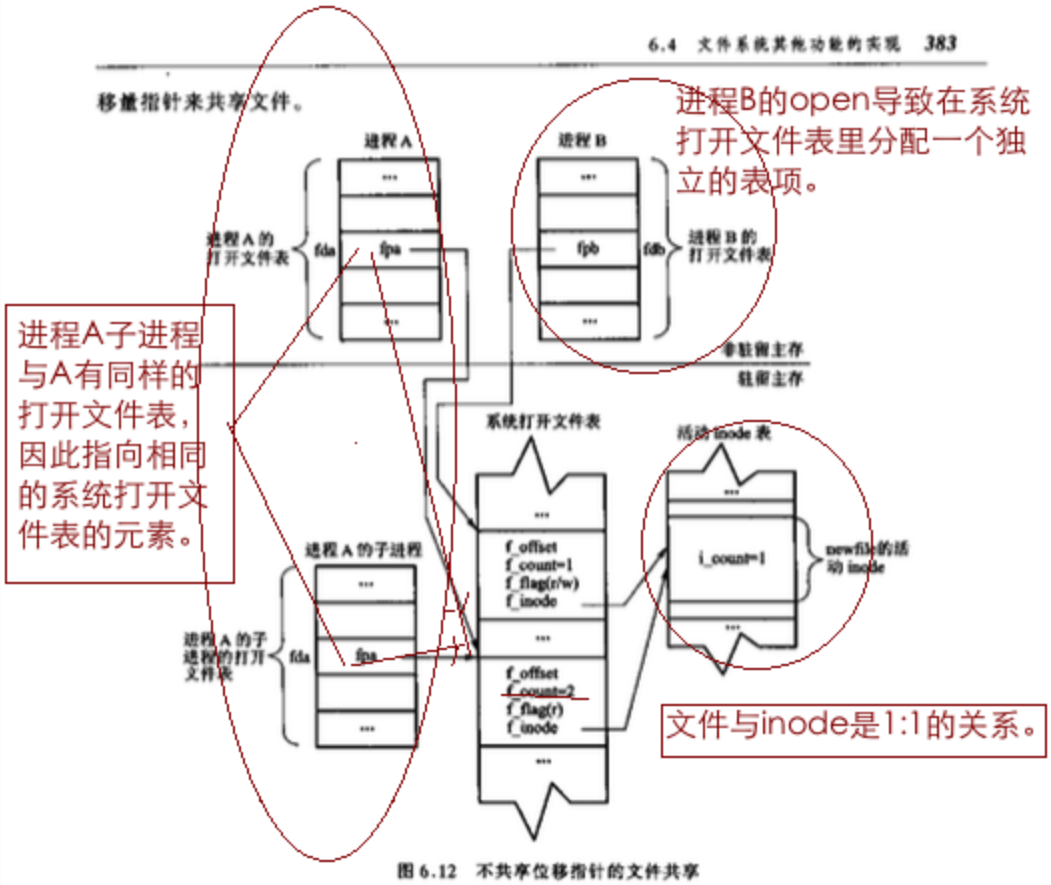
i_count = 1： 这个文件目前的软或者硬链接是1. 磁盘上没有其他的文件在lnk。

图 6.11 共享位移指针的文件共享

在图 6.12 中，进程 A 及其子进程通过同一个系统打开文件表项共享文件 newfile，其 f_count 值为 2。而进程 B 独立使用一个系统打开文件表项，其 f_count 值为 1，但它却通过同一个活动 inode 与进程 A 及其子进程共享文件 newfile。

下面来看系统打开文件表项的申请和释放过程。当进程要求打开文件时，首先，系统为它申请一个系统打开文件表项，并建立此表项与相应文件活动 inode 之间的联系；然后，把用户打开文件表项中的一个空闲项指向它。如果在此之后，应用进程通过系统调用 fork 创建一个子进程，系统自动把用户打开文件表项所指的系统打开文件表项中的 f_count 值加 1。反之，当进程关闭一个文件时，系统不能简单地释放系统打开文件表项，必须首先判断 f_count 的值是否大于 1。如果其值大于 1，说明还有进程共享相应的系统打开文件表项，此时只需把 f_count 值减 1 即可。由此可知，进程之间到底采用哪一种方式动态地共享同一个文件，若是先执行 fork，再用"打开"系统调用来打开同一名称的文件，则系统分别为两个进程分配不同的系统打开文件表项，并指向同一活动 inode，子是两个进程独立地使用各自的读写位移量指针。但是如果进程在打开一个文件之后，再用 fork 系统调用建立子进程，由于子进程的用户打开文件表项是从父进程那里复制得到的，它自然会指向父进程所使用的系统打开文件表项，因此，父子进程就共用同一个位

移量指针来共享文件。



进程A子进程与A有同样的打开文件表，因此指向相同的系统打开文件表的元素。

进程B的open导致在系统打开文件表里分配一个独立的表项。

文件与inode是1:1的关系。

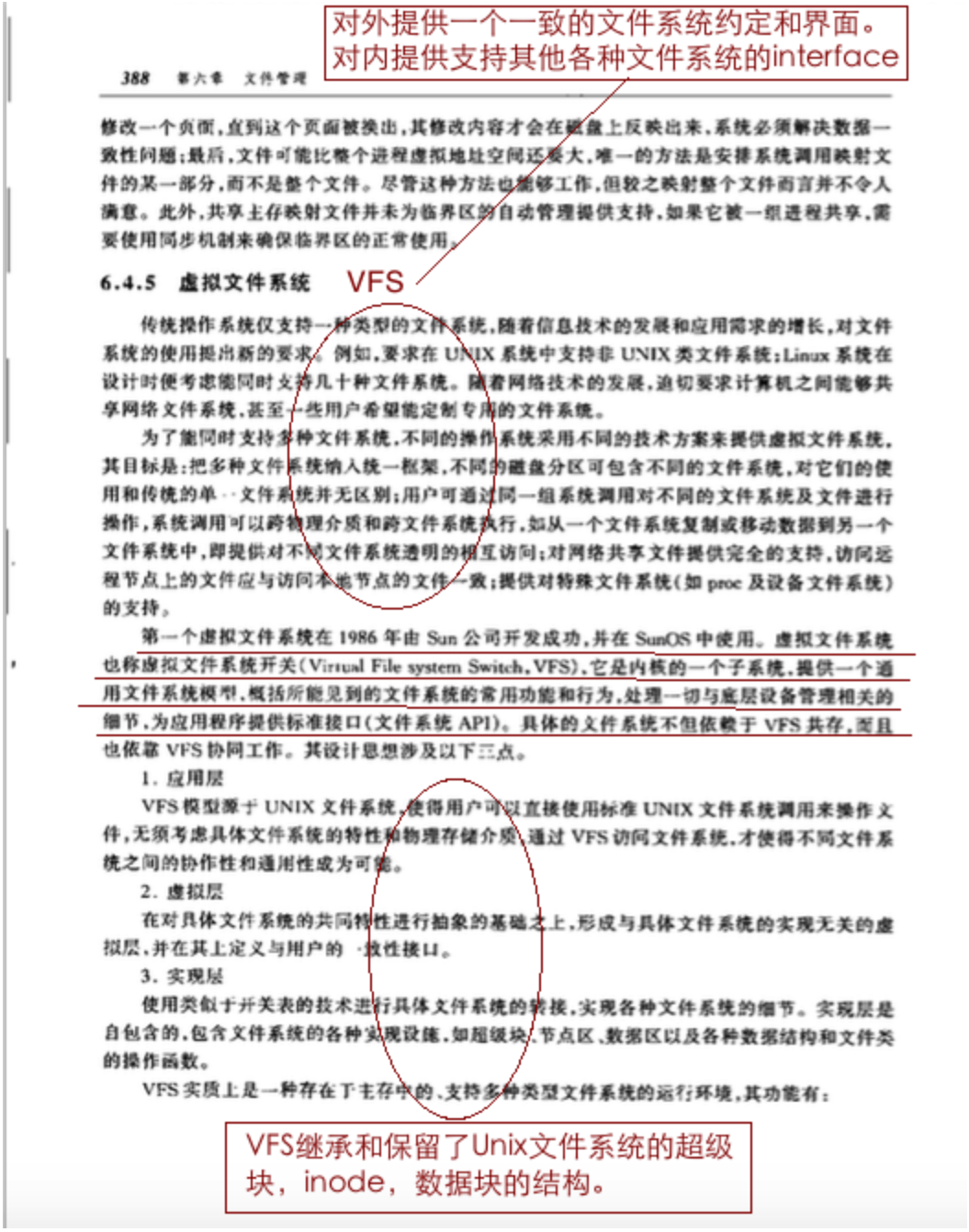
图 6.12 不共享位移指针的文件共享

3. 文件符号链接共享

操作系统可以支持多个物理磁盘或逻辑磁盘(分区)，那么，文件系统是建立一棵目录树还是多棵目录树呢？有两种方法，将盘符或卷标分配给磁盘或分区，并将其名字作为文件路径名的一部分。Windows 操作系统采用这种方法。

UNIX/Linux 系统使用另外的方法，每个分区都有自己的文件目录树，当有多个文件系统时，可通过安装的方法整合成一棵更大的文件目录树。现在出现一个小问题，系统中的每个文件对应于一个 inode，其编号是唯一的，但是两个不同的磁盘或分区可能都含有相同 inode 号所对应的文件，也就是说，在整合的目录树中，inode 号并不唯一地标识一个文件，因而，无法做到从不同的文件系统生成指向同一个文件的链接。对于 link 之类的系统调用，只能拒绝创建跨越文件系

[注释]：系统文件打开表是一个共享的数据结构，可以使得进程之间对一个文件（inode）共享成为可能。两种方式：1. 父子进程之间指向同一个系统文件表的元素, 共享对一个文件的操作，包括当前读写的偏移量。2. 不同进程之间通过不同系统文件表表项指向同一个inode（文件）。



对外提供一个一致的文件系统约定和界面。
对内提供支持其他各种文件系统的interface

388 第六章 文件管理

修改一个页面，直到这个页面被换出，其修改内容才会在磁盘上反映出来。系统必须解决数据一致性问题；最后，文件可能比整个进程虚拟地址空间还要大，唯一的方法是安排系统调用映射文件的某一部分，而不是整个文件。尽管这种方法也能够工作，但较之映射整个文件而言并不令人满意。此外，共享主存映射文件并未为临界区的自动管理提供支持，如果它被一组进程共享，需要使用同步机制来确保临界区的正常使用。

6.4.5 虚拟文件系统 VFS

传统操作系统仅支持一种类型的文件系统，随着信息技术的发展和应用需求的增长，对文件系统的使用提出新的要求。例如，要求在 UNIX 系统中支持非 UNIX 类文件系统；Linux 系统在设计时便考虑能同时支持几十种文件系统。随着网络技术的发展，迫切要求计算机之间能够共享网络文件系统，甚至一些用户希望能定制专用的文件系统。

为了能同时支持多种文件系统，不同的操作系统采用不同的技术方案来提供虚拟文件系统，其目标是：把多种文件系统纳入统一框架，不同的磁盘分区可包含不同的文件系统，对它们的使用和传统的单一文件系统并无区别；用户可通过同一组系统调用对不同的文件系统及文件进行操作，系统调用可以跨物理介质和跨文件系统执行，如从一个文件系统复制或移动数据到另一个文件系统中，即提供对不同文件系统透明的相互访问；对网络共享文件提供完全的支持，访问远程节点上的文件应与访问本地节点的文件一致；提供对特殊文件系统（如 proc 及设备文件系统）的支持。

第一个虚拟文件系统在 1986 年由 Sun 公司开发成功，并在 SunOS 中使用。虚拟文件系统也称虚拟文件系统开关（Virtual File system Switch，VFS），它是内核的一个子系统，提供一个通用文件系统模型，概括所能见到的文件系统的常用功能和行为，处理一切与底层设备管理相关的细节，为应用程序提供标准接口（文件系统 API）。具体的文件系统不但依赖于 VFS 共存，而且也依靠 VFS 协同工作。其设计思想涉及以下三点。

1. 应用层

VFS 模型源于 UNIX 文件系统，使得用户可以直接使用标准 UNIX 文件系统调用来操作文件，无须考虑具体文件系统的特性和物理存储介质，通过 VFS 访问文件系统，才使得不同文件系统之间的协作性和通用性成为可能。

2. 虚拟层

在对具体文件系统的共同特性进行抽象的基础之上，形成与具体文件系统的实现无关的虚拟层，并在其上定义与用户的一致性接口。

3. 实现层

使用类似于开关表的技术进行具体文件系统的转接，实现各种文件系统的细节。实现层是自包含的，包含文件系统的各种实现设施，如超级块、节点区、数据区以及各种数据结构和文件类的操作函数。

VFS 实质上是一种存在于主存中的、支持多种类型文件系统的运行环境，其功能有：

VFS继承和保留了Unix文件系统的超级块，inode，数据块的结构。

(1) 记录所安装的文件系统类型;
(2) 建立设备与文件系统之间的联系;
(3) 实现面向文件的通用操作;
(4) 涉及特定文件系统的操作时,映射到具体的文件系统中去。

VFS 抽象层之所以能衔接各种不同的文件系统,是因为它定义了所有文件系统都支持的基本抽象接口和数据结构,同时文件系统也将自己的诸如"文件如何打开"、"目录如何定义"等概念在形式上与 VFS 的定义保持一致。对于像 FAT 和 NTFS 这类非 UNIX 风格的文件系统,必须经过封装,提供符合 VFS 概念的接口,比如,某文件系统不支持 inode 概念,它也必须在主存中装配 inode 结构体,如同其本身包含 inode 一样。这些装配和转换需要在使用现场引入特别处理,使得非 UNIX 文件系统能够兼容 UNIX 文件系统的使用规则,满足 VFS 的接口需求,这样一来,非 UNIX 文件系统便可与 VFS 共同工作,只是在性能上会产生少许影响。

6.5 Linux 文件系统

6.5.1 Linux 虚拟文件系统

Linux 最早采用 Minix 文件系统,20 世纪 90 年代初推出专为自己设计的优秀、高效的文件系统 Ext(Extended File System),目前常用的是 Ext2,最新升级版本是 Ext3。为了实现开放性,Ext 的设计有重大改进,引入虚拟文件系统(VFS),为用户提供统一、抽象的文件系统界面,以支持多种文件系统。迄今所能支持的文件系统有 Ext、Ext2、Ext3、FAT、Minix、UMSDOS(DOS 和 Linux 共存文件系统)、proc(Linux 进程信息文件系统)、ISO9660(CD-ROM 文件系统)、HPFS(OS/2 文件系统)、ROMFS(只读上存文件系统)、Xenix(Xenix 文件系统)、NFS(Sun 网络文件系统)、UFS(BSD 文件系统)、SYSV(UNIX System V)等。更进一步地,还支持多种文件系统的互操作。图 6.15 给出 VFS 和具体文件系统之间的关系。

Linux 虚拟文件系统采用面向对象设计思想,虚拟文件系统相当于面向对象系统中的抽象基类,它可派生不同的子类,以支持多种文件系统。但从效率考虑,内核纯粹使用 C 语言编程,故并未直接利用面向对象的语义。下面的讨论中使用术语"对象",实际上是一个结构体(struct),它所代表的确实是一个对象(object),虚拟文件系统由下列 4 个对象类组成。

(1) 超级块(super_block)对象:代表一个文件系统,存放已安装的文件系统信息。如果是基于磁盘的文件系统,此对象便对应于存放在磁盘上的文件系统控制块,每个文件系统都对应于一个超级块对象。

(2) 索引节点(inode)对象:代表一个文件,存放通用的文件信息。如果是基于磁盘的文件系统,此对象对应于存放在磁盘上的文件的 FCB,即每个文件的 inode 对象,而每个 inode 都有一个 inode 索引节点号,唯一标识某个文件系统中的指定文件。

Ext2扩展了传统的Unix文件系统，加入了Block Group的概念和机制。

关于Ext2比较好的技术介绍：
http://wiki.osdev.org/Ext2

};

6.5.4 Ext2 文件系统

Ext2(second extended file system)文件系统支持标准 UNIX 文件类型，包括普通文件、目录文件、特别文件和符号链接文件。Ext2 文件系统可管理特大磁盘分区，文件系统最大可达 4 TB。此外，还支持长文件名(最长 1 012 个字符)、可选的逻辑数据块大小，提供数据更新时同步写入磁盘和快速符号链接功能。

Ext2 文件系统的信息都保存在数据块中，数据块的长度相同，但不同 Ext2 文件系统中的数据块大小可以不同，其物理结构如图 6.18 所示。Ext2 所占用的磁盘除了引导块外，逻辑分区划分为块组，每个块组依次包括超级块、块组描述符表、块位示图、inode 位示图、inode 表以及数据块区，重复保存有关文件系统的关键信息及所存储的文件和目录信息。文件系统保存逻辑块号，由块设备驱动程序将逻辑块号转换成块设备的物理存储位置。引导块是磁盘上的第一个数据块，只有根文件系统才有引导程序放在这里，Linux 以 Ext2 作为其根文件系统。

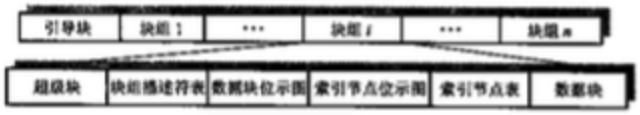


图 6.18 Ext2 文件系统结构

采用块组划分的目的：一是提高文件系统的可靠性，每个块组中都有管理信息的副本，当文件系统崩溃时可以容易地恢复；二是提高文件系统的性能，由于块组内的数据块靠近其 inode，文件 inode 靠近其目录 inode，从而将磁头定位时间缩减至最短，加快磁盘访问速度。

1. 超级块

Ext2 超级块用来描述目录和文件在磁盘上的静态分布情况，包括尺寸和结构，每个块组都有一个超级块，只有块组 1 的超级块才被读入主存工作，直至卸载，其他块组的超级块仅作为恢复备份。超级块主要包括：块组编号、块数量、块长度(1 KB～4 KB)、空闲块数量、inode 数量、空闲 inode 数量、第一个 inode 号、第一个数据块位置、每个块组中的块数、每个块组的 inode 数，及安装时间、最后一次写时间、安装信息、文件系统状态信息等内容。

2. 块组描述符

每个块组都有一个块组描述符，记录此块组的以下信息。

(1) 数据块位示图

表示数据块位示图占用的块号，此位示图反映块组中数据块的分配情况，在分配或释放数据块时需使用数据块位示图。

(2) inode 位示图

如何通过一个inode # 来定位属于中哪个块组，和在块组里是第几个inode记录：
block group = (inode - 1) /INODES_PER_GROUP
index = (inode - 1) % INODES_PER_GROUP

时需使用 inode 位示图。

(3) inode 表

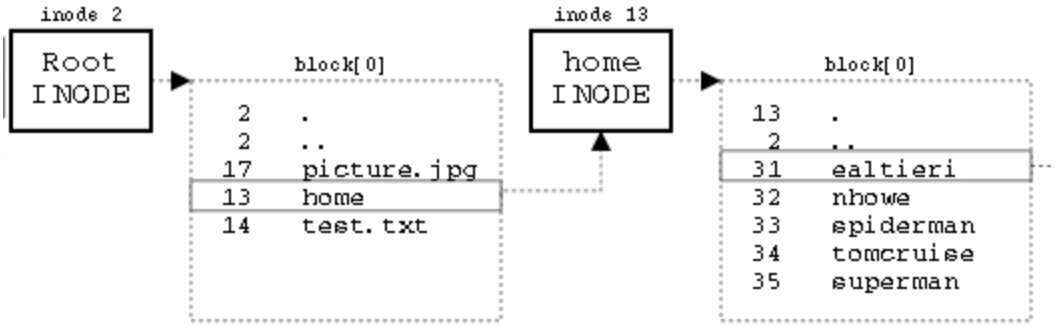
块组中 inode 所占用的数据块数，系统中的每个文件对应于一个 inode，每个 inode 都由一个数据结构来描述。

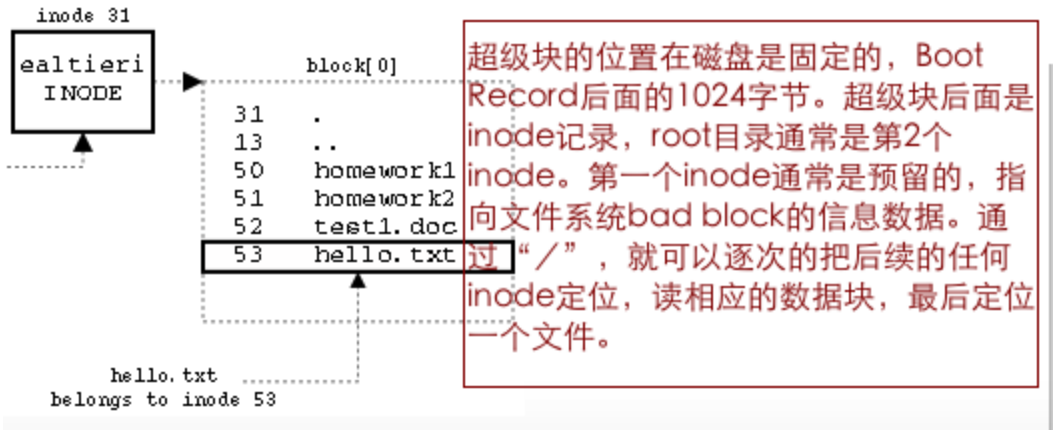
(4) 空闲块数，空闲 inode 数和已用数目。

一个文件系统中的所有块组描述符结构组成一个块组描述结构表，每个块组在其超级块之后都包含一个块组描述结构表的副本。实际上，Ext2 文件系统仅使用块组 1 中的块组描述结构表。

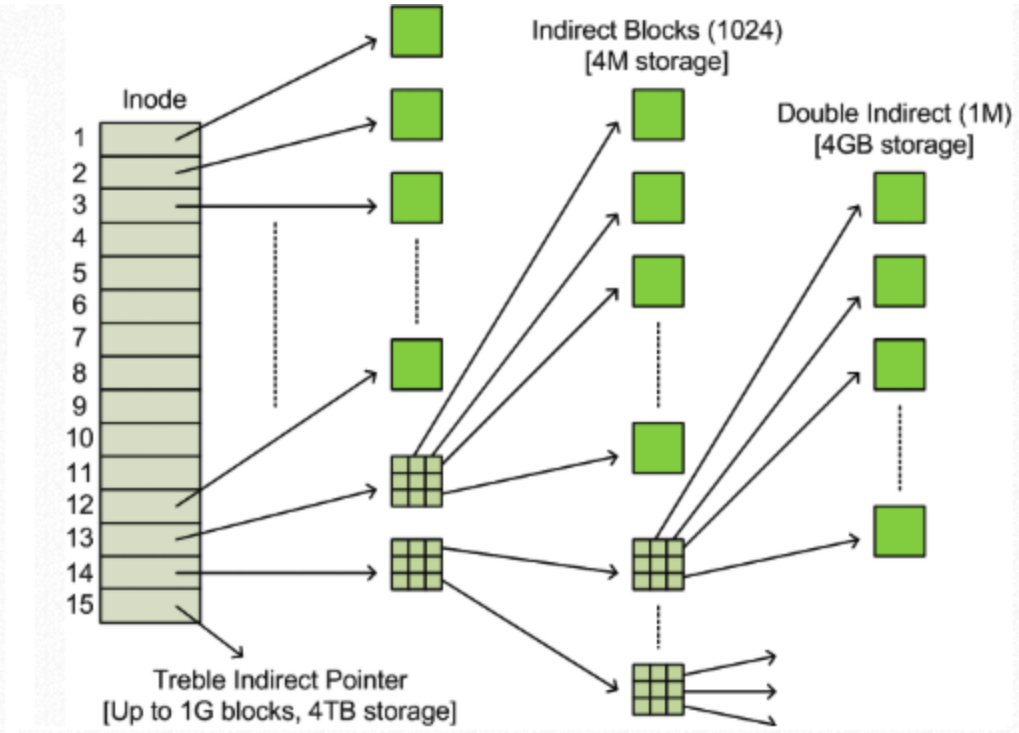
3. 索引节点

在 Ext2 中，每个文件都由一个 inode 来唯一描述，每个块组的 inode 集中存放在一个 inode 表中，每个 inode 有唯一的 inode 号。inode 位示图记录 inode 的分配情况，inode 起着文件控制块的作用，可用来对文件进行控制和管理。进一步可区分为磁盘 inode 和主存活动 inode。创建文件时就分给一个磁盘 inode，当文件被打开时，其对应的磁盘 inode 被复制到活动 inode 中；当文





超级块的位置在磁盘是固定的，Boot Record后面的1024字节。超级块后面是inode记录，root目录通常是第2个inode。第一个inode通常是预留的，指向文件系统bad block的信息数据。通过“/”，就可以逐次的把后续的任何inode定位，读相应的数据块，最后定位一个文件。



inode通常含有15个指针（32bit）指向数据块。其中12个是直接指向数据块。第13个指针是一次间接指针；第14个指针是二次间接指针，可以支持4GB的文件大小；第15个指针是三次间接指针，可以支持4TB的文件大小。

假设每个数据块是4K大小。

* 前12个指针能覆盖：4K * 12=48KB文件大小。

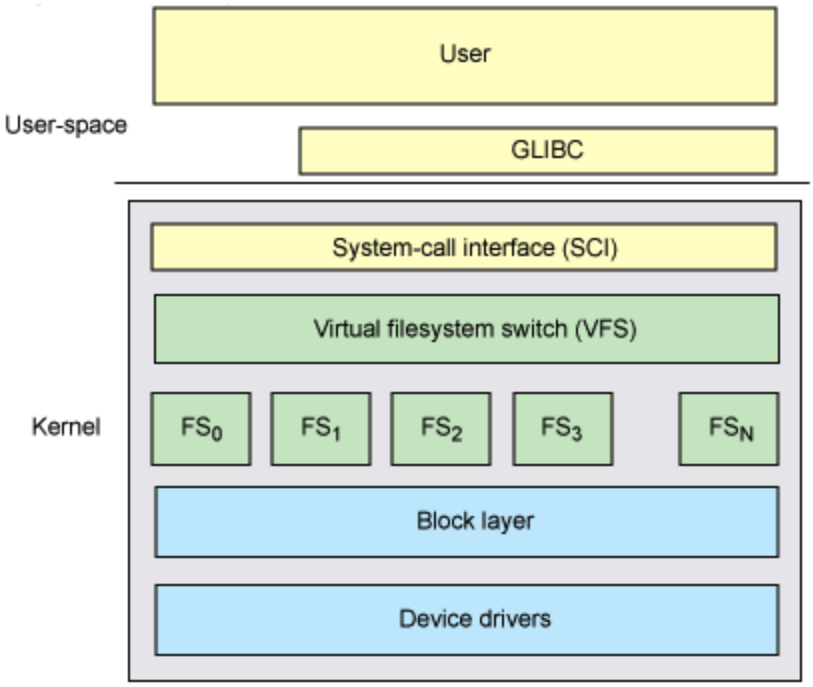
* 13th：指向一个4K的数据块。这个数据块全部被用做32bit（4Byte）的指针来用。每个指针指向一个4K的数据块。因此，能支持：（4K／4） * 4K = 4M文件大小。

* 14th：2级指针。（4K／4） * （4K／4） * 4K = 4G文件大小。

* 15th：3级指针。（4K／4） * （4K／4） * （4K／4） * 4K = 4T文件大小。

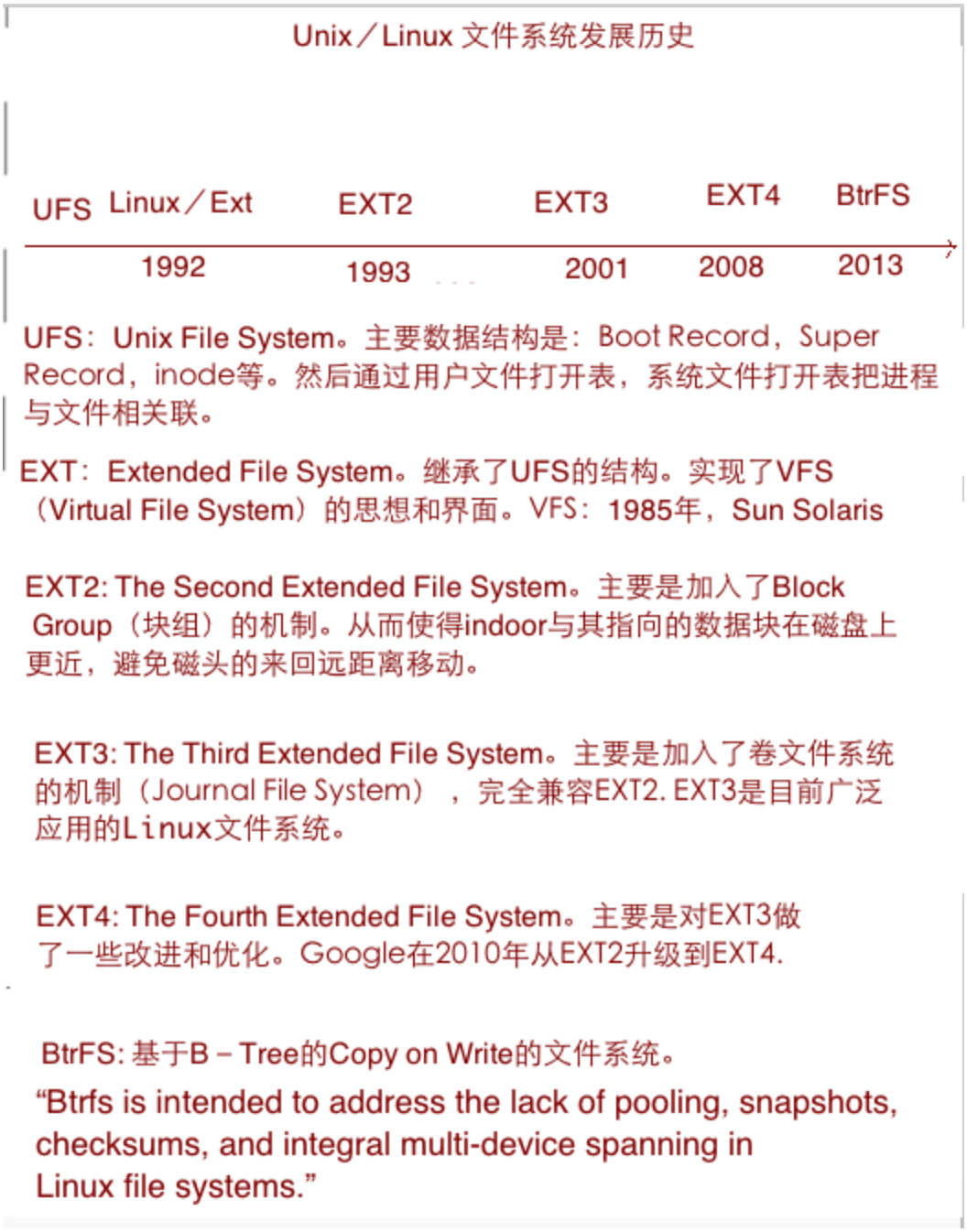
[注：每个指针是32位，所以是4Byte。因此，一个数据块，如果被全部用做指针，是1024个指针。]

本章小结



任何数据都是一个文件 的设计准则贯穿整个操作系统文件系统的发展。在VFS的虚拟层下，一个操作系统可以支持多种类型的文件系统实现，从而用户程序可以保持一个完整的存取和编程界面。操作系统通过用户打开文件表，系统打开文件表，Superblock，inode等重要数据结构，把文件与进程有机的联系起来。

文件数据块的定位，不管算法有多少相对的寻址，或者转换，一定是在源头有一个绝对的物理定位，从而这那个地位为出发点，可以读写任何一个数据块。这个出发点就是"\"目录，Metadata存放在第2个inode里。第2个inode在磁盘上的绝对偏移量可以通过boot（1024Bytes），Superblock（1024Bytes），inode（128Bytes 一个记录）来手工算出。inode这磁盘上是连续存放的，固定大小的记录。



[注释：]“系统来自演化”。Linux文件系统从最开始依托MINIX的简单文件系统，然后从早期Unxi File System（UFS）开始演变，历经EXT（Extended File System，开始支持VFS Interface），EXT2，EXT3，EXT4和BtrFS等。不断优化。其中EXT3是一重要的飞跃，支持卷（Journal）。

http://en.wikipedia.org/wiki/Common_Criteria

2005年，Common Criteria代替TCSEC

第七章

安全操作系统橘皮书（TCSEC）

定义5个OS的安全等级

操作系统的安全与保护

7.1 安全性概述

计算机安全所涉及的范围很广，至今尚无统一定义，其基本内容是对计算机系统的硬件、软件、数据加以保护，不会因偶然或恶意的原因而造成信息的破坏、更改和泄密，使计算机系统得以连续正常地运行。计算机安全既包括物理方面的，如对计算机环境、设施、设备、载体和人员，需要采取行政管理上的安全对策和措施，防止突发性或人为蓄意破坏；又包括逻辑方面的，针对计算机系统，特别是计算机软件及应用系统的安全和保护，严防信息被窃取和破坏。影响计算机系统安全性的因素有很多。首先，操作系统是一个并发系统，支持多用户共享一套计算机系统的资源，有资源共享就需要有资源保护，涉及各种安全性问题；其次，随着计算机网络的迅速发展，除了信息的存储和处理之外，还存在大量的数据传输操作，客户机要访问服务器，一台计算机要传输数据给另一台计算机，传输过程中对信息安全性的威胁极大，于是就需要有网络安全性，使得系统中信息的存取、处理和传输满足安全策略；另外，在信息系统中，越来越多的个人资料、商业机构和政府机关的机密信息被存放在数据库中，可被拥有它及依赖它的用户所使用，但不能被未授权者访问，这就提出了信息系统的基础——数据库安全性问题；最后，计算机安全中的一个特殊问题是计算机病毒，它会对系统资源和信息造成很大的危害和破坏，需要采取措施预防、发现和解除它。

计算机系统的安全性和可靠性是两个不同的概念。可靠性是指系统正常持续运行的程度，其目标是反故障；安全性是指不因人为疏漏或蓄谋作案而导致信息资源的泄漏、篡改和破坏，其目标是反泄密。可靠性是基础，安全性则更为复杂。鉴于计算机系统自身的脆弱性和安全模式的先天不足以及计算机犯罪现象的普遍存在，构造安全计算机信息系统绝非易事。一般来说，信息系统的安全模型涉及管理和实体安全性、网络通信安全性、软件系统安全性和数据库系统安全性。软件系统中最重要的是操作系统，由于它所处的特殊地位，计算机安全问题大都由操作系统来保证，所以，操作系统安全性是计算机系统安全性的基础。本章重点讨论操作系统安全性，主要内容包括：

单机安全，网络安全， 数据中心安全。

7.4 安全机制

操作系统是计算机硬件和用户之间的接口，不但要让用户通过此接口方便、高效地使用系统资源，而且还要保证计算机系统安全、可靠地运行，应提供机制和工具来实现安全策略所描述的安全问题。安全机制的主要目标是：根据安全策略对用户的操作进行控制，防止用户对系统资源的非法存取；标识系统中的用户并对其进行身份鉴别；监督系统运行的安全性；保证系统自身的安全性和完整性。为了实现这些目标，需要建立有效的安全机制。

7.4.1 硬件安全机制

优秀的硬件保护设施是实现高效、安全、可靠的操作系统的基础，计算机硬件安全的目标是

不支持虚拟主存的CPU的通用OS基本上不存在了。CPU在启动阶段可能运行做realmode下，之后通常会启动MMU的VM功能。

保证其自身的可靠性，并为操作系统提供基本安全设施。常用的方法有：主存保护、运行保护和I/O保护等。

隔离是重要的保护机制。

1. 主存保护

对于安全操作系统而言，存储保护是最基本的要求，存储保护机制保证指令只能访问程序中被授权访问的存储单元，保护单位可以是存储段、页面、字块或字，保护单位越小，保护精度越高，但其开销也越大。在多道程序系统中，由于同时存在系统程序和应用程序，在运行时，操作系统必须把它们相互隔离以便互不干扰。如果进程能够不经意地写入系统或其他进程的存储空间，将导致严重的后果。既要控制进程对自己的存储区域的访问，又要控制对其他进程主存区的访问，需要主存保护功能，即必须保护进程不受其他进程的干扰。需要解决两个问题：一是进程必须被禁闭在操作系统分配给它的主存区内，此类禁用的实施依赖于系统所提供的硬件保护机制及采用的存储管理方案；二是考虑进程对不同主存区的访问类型，在理想的情况下，对于给定的主存区，每个进程都应有自己的权限集合。

通过一些控制寄存器，做一些越界检查。

(1) 不支持虚拟主存的系统

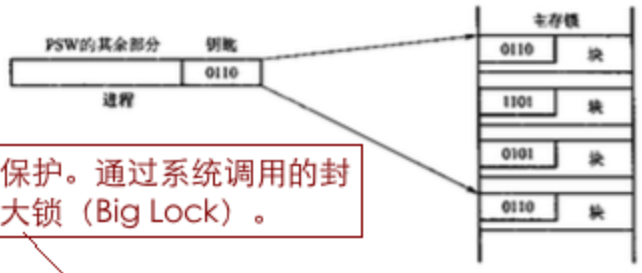
为了保证进程禁闭在自己的区域内，需要对所访问的每个物理地址进行有效性检查，以验证它仅对所分得的主存区进行引用。使用以下三种保护机制实施主存保护。

① 下界和上界寄存器法

对寄存器指定一个连续主存区的下界和上界地址，进程的代码和数据便存放在此区域中。对于给定的逻辑地址，先与下界寄存器的内容相加，计算出对应的物理地址，计算结果与上界寄存器的内容作比较，如果超出边界值，则发出越界中断，并终止进程的执行。下界寄存器又称重定位寄存器，用来实现动态地址绑定，如果进程的代码和数据移动到新的主存区，只需相应地修改上下界寄存器的值。

② 基址和限长寄存器法

基址和限长寄存器指定一个连续主存区的起始地址和大小，其保护原理与下界和上界寄存器法相同，只是用限长寄存器代替上界寄存器而已。



核心态空间的保护。通过系统调用的封装，类似一个大锁（Big Lock）。

图 7.3 钥匙和主存锁

存锁"，其余 1 位称为"取保护位"。每个进程分得与其他进程不同的钥匙值，此进程的所有主存块均被设置与钥匙值相同的主存锁。当进程运行时，操作系统把分给它的钥匙放入 PSW 的存储控制键字段，每当处理器访问主存或 DMA 执行一个页面 I/O 操作时，都把钥匙和锁进行匹配，以决定是否允许访问。若运行进程试图访问钥匙和锁不符的主存块时，便会产生保护中断。应用进程使用值 1～15；操作系统则使用值 0，被允许访问整个主存储器，且有权修改和设置钥匙和主存锁。"取保护位"指示"主存锁"是适用于"写"还是"读写"，钥匙与锁匹配时允许写操作。若"取保护位"为 1，则执行读操作时也要检查钥匙和锁是否匹配；若"取保护位"为 0，即使钥匙和锁不匹配，也允许读访问，从而实现信息共享。

(2) 支持虚拟存储器的系统

进程存储空间的隔离可通过虚拟存储器的方法来实现，虚拟地址空间分为内核空间和用户空间。在用户态只能访问用户空间，在核心态可以访问内核空间和用户空间。内核空间在每个进程的虚拟地址空间中都是固定的，系统中仅有一个内核在运行，所有进程均映射到单一的核心地址空间。

进程的地址空间通过地址转换机制映射到不同的物理页面上，分段、分页或段页式虚拟存储管理系统是通过段表、页表和段页表间接地访问虚拟主存的一段或一页。由于管理表对于进程是私有的，因此，通过在相关表项中设置保护信息，就可表示进程对所访问(私有或共享)的段或页面具有不同的访问权限，以便在每次硬件保护机制进行地址转换时执行必需的权限检查，如果不允许，则产生一个异常。

具体实现技术如下：在分段或段页式虚拟存储管理中，可以在段表中增加访问类型字段，扩展逻辑地址到物理地址的映射计算步骤，增加检查申请访问类型是否是对此段的有效访问，若否，则触发保护中断，以保证每个虚拟地址总会禁闭在分配给进程的地址空间之内。这种方式的灵活性很大，段内信息的保护与逻辑命名空间相关，而与物理生存无关，不同的段可规定不相同的访问类型。

分页式虚存管理中，只需在页表中增加访问类型字段，信息保护的原理完全一样。Windows 系统采用纯分页方式，通过不同的页面模式、页面类型和页面状态来完成页面安全性检查，具体

各种CPU 和／或OS中间的数据结构的介入，使得可以把相关的存取权限，身份验证等规则应用上去。达到一定的安全保护功能

http://en.wikipedia.org/wiki/Sandbox_(computer_security)

来说：

① 区分内核模式页面和用户模式页面：内核模式页面仅在核心态才可访问，且仅用于系统的数据结构，用户只能通过适当的内核函数对其访问；

② 区分页面类型：Win32 API 区分页面访问模式为不可访问、只读、读写、只能执行、执行和可读、执行和读写，其他模式对页面的访问都会造成违法；

③ 进程地址空间页面状态：有空闲、保留和提交三种，被记录在对应的页表项中。空闲页面是指尚未分配给进程的无效页面，任何访问企图都会产生无效页号错误；访问保留或提交页面可能会造成不同类型的缺页中断处理。

(3) 沙盒技术

在大多数操作系统中，进程所调用的函数会自动继承调用进程的所有访问权限，特别是可访问进程的整个虚拟主存。若函数是不可信的，如从因特网下载的程序(可能带有"特洛伊木马"病毒)，这种不受限制的访问是不允许的，因为会给系统造成严重的威胁。为了限制不可信程序造成潜在损害的范围，系统可限制访问权限为调用进程所具有的授权的一小部分特权，通常称这种缩小后的访问环境为"沙盒"。最重要的一点是，限制程序在小范围的主存区(主存沙盒)中运行，访问沙盒以外的数据及转向沙盒以外的代码的企图都会产生保护中断，并终止程序的执行。主存沙盒的一种变体是为每个程序提供两个单独的沙盒，一个沙盒用于存放代码，另一个沙盒用于存放数据。只允许程序在数据沙盒中读写，且只能在代码沙盒中执行，这样进一步缩小程序对主存的访问权限，从而实现更严格的主存保护。

Java 提供一个可定制的沙盒安全模型，保护用户免受恶意程序代码的攻击，Java 程序可在沙盒范围内执行任何操作，但不能超越范围。对于非可信 Java Applet，沙盒会限制其活动，如读写本地磁盘、连接网络主机、建立新的进程、下载相关代码等都被施以一定的限制，使其没有实施有害动作的机会。Java 沙盒的基本组件有：类下载器、类文件验证器和安全管理器等。

2. 运行保护

"运行保护"机制为进程的运行设置不同的保护域，安全操作系统很重要的一点是进行分层设计，而运行域正是一种基于保护环的层次等级式结构。运行域是进程的运行区域，最内层拥有最小环号的环具有最高特权，最外层拥有最大环号的环具有最低特权，一般的系统不少于 3～4 个环。

最简单的方式是设置两层保护域：核心域和用户域，它隔离操作系统程序和应用程序。保护系统程序或应用程序不受其他程序的破坏，核心域运行的程序处于系统模式(核心态)，用户域运行的程序处于用户模式(用户态)。如图 7.4 所示，运行在核心域下的程序比运行在用户域下的程序有更多的访问权限，以达到保护系统程序和其他应用程序的目的，这些权限包括可访问全部主存地址(包括 I/O 端口)和执行特权指令。

多层域结构中，最内层域存放操作系统程序；最靠近操作系统域的外层是存放受限制的系统程序(实用程序)域，如编辑程序、编译程序、汇编程序和数据库管理系统；最外层域存放应用程序。运行不同的程序时，最重要的安全措施是：

一个权限比较小的运行环境，可以测试没有验证过的程序，例如，下载的代码。

通过system call进入核心态。

例如，对cpu控制寄存器的操作，对有些cache指令的操作。

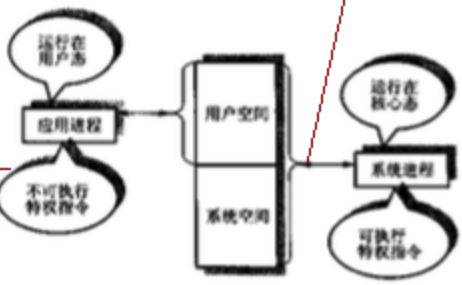


图 7.4 处理器模式扩展操作系统的访问权限

(1) 程序可以访问驻留在同域或外域中的数据，但不会调用外域中的程序；

(2) 程序可以调用驻留在同域或内域中的服务(函数)，但不能访问内域中的数据。

这样可以保证等级域机制中某域不被外层域侵入。进程隔离机制与等级域机制不同，进程可在任何时刻在任何域内运行，在运行期间还可从一个域转移到另一个域。当进程在某域内运行时，进程隔离机制将保护此进程免遭在相同域内运行的其他进程的破坏，即系统将隔离同一域内同时运行的多个进程。

VAX/VMS 操作系统利用处理器的 4 种模式构成保护域来加强对系统资源的保护和共享。处理器模式决定：指令执行特权，即处理器当前可执行的指令系统子集；随当前模式增减的存储访问权限，即当前指令可存取的虚拟主存的位置。

(1) 内核(kernel)态：执行操作系统内核，包括主存管理、中断处理、I/O 操作等。

(2) 执行(executive)态：执行操作系统的系统调用，如文件操作等。

(3) 监管(supervisor)态：执行操作系统其余的系统调用，如应答用户请求。

(4) 用户(user)态：执行应用程序，如编辑、编译、链接等实用程序和各种应用程序。

在较低特权状态下执行的进程常常需要调用在更高特权状态下执行的例程。例如，应用程序需要操作系统的某种服务，使用访管指令可获得这种调用，此指令会引起中断，从而将控制权转交给处于高特权状态下的例程。执行返回指令可通过正常或异常中断返回至断点。采用保护环时具有单向调用关系，如果在域 D_j 中欲调用 D_i 中的例程，则必然有 $i<j$。

3. I/O 保护

CPU 与计算机系统所连接的各种设备进行通信时，采用两条途径：文件映射 I/O 和 I/O 指令。在前一条途径中，对设备的访问控制与主存保护机制相似，仅有授权的系统进程，如设备驱动程序和存储管理程序才能实现文件到主存区的文件映射操作。

采用后一条途径时，必须让这些 I/O 指令具有特权，以保证 I/O 设备不会被应用进程直接访问。相反地，想使用设备的应用进程发出一个 I/O 系统调用，切换至核心态并把控制权转交给操作系统，系统的设备驱动程序代表发出调用的应用进程执行 I/O 操作。这些系统程序经过

通常是Memory Mapped I／O。
对I／O的存取按照内存的规则来保护。

[注释]：操作系统的安全等级规范最开始是NSA发布了一个橘皮书规范TCSEC，定义了5个安全等级。TCSEC(1983年 - 1999年）现在已经被CC（Common Criteria）和FIPS等安全规范逐步取代。CC定义了一个软件产品如果需要卖入到敏感部门所需要通过的相关需求。

http://en.wikipedia.org/wiki/Kerberos_(protocol)

用户登录系统时，需要输入用户名和密码供系统鉴别和标识其合法身份。用户名是标识，告诉计算机此用户是谁，而密码是确认证据，证明是正确的用户。当用户要求登录时，由守护进程getty接收用户所输入的用户名、密码并激活login，它根据/etc/passwd文件检查匹配，如果是合法用户，就为他启动shell进程。

3. Kerberos网络身份认证

Kerberos网络认证机制是由美国麻省理工学院在20世纪80年代的研究计划Athena中所开发出来的著名密钥分发和鉴别系统，适用于客户－服务器模式的网络系统，用户可以在一个不安全网络中的一台计算机上使用此协议来对另一台计算机进行认证访问。

在Kerberos中，假设客户机A上的一个进程欲通过网络通信来调用应用服务器B上的一个进程的服务。Kerberos提供身份认证服务(Authentication Service，AS)及票证颁发服务(Ticket Granting Service，TGS)，允许客户和应用服务器在一个特定的会话中传送验证消息到合作进程。首先，客户必须向KDC(Key Distribution Center，密钥分发中心)的AS服务器证明自己的身份，然后，才能接收分发给自己的票证，可通过口令完成这项工作。此后，客户会收到在整个会话期均有效的用于访问专用TGS服务器的票证，此专用服务器的任务是分发其他应用服务器的票证。图7.5给出Kerberos V.5中使用应用服务器B的客户机A必须遵守的协议。

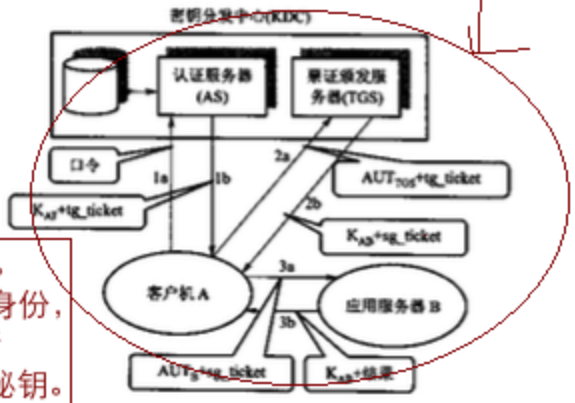


图7.5 Kerberos体系结构

1. KDC是可信的。
2. 通过逐步验证身份，逐步获得与B通话的权限和相应的秘钥。

步骤1：客户机A从AS处得到TGS所需要的许可票证。

1a. 登录时，客户机A向AS提供口令，以证明其合法身份，AS在数据库中验证A的访问权限，并生成相应的票证。

1b. AS向客户机A返回一条消息$E(\{K_{AT}, tg_ticket\}, K_A)$，其中，$K_{AT}$是A和TGS通信的会话密钥，tg_ticket是一个送往TGS的许可票证。此消息使用客户机A的密钥K_A进行加密，

用A的秘钥加密，把KAT（A与TGS通信的秘钥分发给A。

http://en.wikipedia.org/wiki/Symmetric-key_algorithm

建立从根本上不可能被未授权用户解密的加密算法，以提高信息系统及数据的安全性和保密性，防止信息被窃取和泄密。数据加密技术分为两类：一类是数据传输加密技术，其目的是对网络传输中的数据流加密，又分成链加密和端加密；另一类是数据存储加密技术，其目的是防止系统中所存储的数据泄密，又分成文件级（对单个文件）加密和驱动器级（对逻辑驱动器上的所有文件）加密。

加密技术可用于将明文转化为密文来保护暴露在未受保护的介质上的原文，定义加密函数（算法）和解密函数（算法），使用加密函数和加密密钥把明文变成密文的过程称为加密；使用解密函数和解密密钥将密文转化为原始明文的过程称为解密。明文（plain text）是指被加密的文本；密文（cipher text）是指加密后的文本；密钥是加解密算法中所使用的关键参数。

一般而言，密码系统依其应用范围可对信息提供下列功能。

（1）秘密性：解决信息泄漏问题，防止非法的信息接收者窃取信息；

（2）鉴别性：解决发送者的真实性问题，信息接收者能够确认信息来源，即此信息确实是由发送方传送而非他人伪造的；

（3）完整性：解决信息被修改或损坏的问题，信息接收者能验证信息未被修改，也不可能被假消息所替代；

（4）不可抵赖：发送者在事后不可能虚假地否认其所发送的信息。

2．基于密钥的算法分类

Kerberos是基于对称密码算法的认证系统。

（1）对称算法

对称算法又称传统密码算法，就是加密密钥能够从解密密钥中推算出来，反之亦然。在大多数对称算法中，加密/解密密钥是相同的，这种算法的安全性依赖于密钥，泄漏密钥就意味着任何人都能对信息进行加密和解密。对称算法分为两种：一是只对明文中的字位（或字节）运算的算法，称序列算法或序列密码；二是对明文中的一组字位运算，这些位组称为分组，相应的算法称为分组算法或分组密码。现代计算机密码的典型分组长度为64位，这一长度大到足以防止分析破译，但又小到足以方便使用。

对称技术的优点是具有很高的保密强度，可经受国家级破译力量的分析和攻击，但其密钥必须通过安全、可靠的途径来传递，密钥管理成为影响系统安全的关键性因素，这使其难以满足系统的开放性要求。

获得了2002年的Turing Award

（2）公开密钥算法

公开密钥算法又称非对称密码算法，其设计机理是：用做加密的密钥不同于用做解密的密钥，而且解密密钥不能根据加密密钥计算出来。之所以称其为公开密钥算法，是因为加密密钥可以公开，其他人能够用加密密钥来加密信息，但只有用相应的解密密钥才能解密信息，因此，加密密钥称为"公开密钥"，解密密钥称为"私有密钥"。

采用非对称密码体制的每位用户都有一对选定的密钥，一个可公开，一个由用户秘密保存。公开密钥算法的出现是现代密码学研究的一项重大突破，其主要优点是：适应开放性的使用环境，密钥管理简单，可以方便、安全地实现数字签名和验证，但其保密强度目前还远未达到对称密

http://en.wikipedia.org/wiki/Public-key_cryptography

http://en.wikipedia.org/wiki/Data_Encryption_Standard

码体制的水平，至今所发明的非对称密码体制绝大多数已被破译，剩下的几种也不能证明其完全不存在缺陷。

加密系统：加密算法可以公开；钥匙才是本质。

3. 计算机密码算法

(1) 数据加密标准

数据加密标准(Data Encryption Standard，DES)是通用的计算机加密算法，是美国标准，用于政府和商业应用。这是一种对称算法，加密/解密密钥是相同的。20 世纪 70 年代中期由美国 IBM 公司开发。尽管 DES 历经几十个年头，但在已知的公开文献中，仍然无法彻底地把 DES 破解，这种加密方法至今仍被公认是安全的。

由于 DES 的密钥只有 56 位，仅有 256 个不同的密钥，随着计算机运行速度的不断提高，容易采用尝试法破解 DES 密文。于是，3 次 DES 加密算法被提出来，它依次使用 3 个 56 位密钥进行 DES 处理，使密钥长度增加至 168 位，使用目前世界上运算速度最快的计算机穷尽尝试，将需要万亿年之久。

(2) 公开密钥算法 RSA

http://en.wikipedia.org/wiki/RSA_(cryptosystem)

RSA(Rivest，Shamir，Adleman)是最流行的公开密钥算法，已成为事实上的国际标准，能被用做加密和数字签名。DES 属于传统的加密算法，要求加解密的密钥是对称的，加密者必须用非常安全的方法把密钥送给解密者，如果通过计算机网络来传送密钥，则密钥就有泄密的可能。彻底解决这个问题的方法是不分配密钥，这就是公开密钥法。要求密钥是对称的，并不是一个必要条件，可以设计出算法，加密时用一个密钥，而解密时使用有联系的另一个密钥。另外，还可能设计出一个算法，即使知道加密算法和加密密钥也无法确定解密密钥。其基本技术如下：

① 网络中每个节点都产生一对密钥，用来对其所接收的消息进行加密和解密；

② 每个系统都把加密密钥放在公共文件中，这是公开密钥，另一个设置为私有的，称为私有密钥；

③ 若 A 要向 B 发送消息，它就用 B 的公开密钥加密消息；

④ 当 B 收到消息时，就用私钥进行解密。由于只有 B 知道其私钥，于是没有其他接收者可破解消息。公开密钥算法的主要缺点是算法复杂，开销大，效率低。

4. 数字签名

在金融和机要系统中，许多业务都要求签名，以备检查和核实，任何签名都不得伪造，也不可抵赖。公开密钥算法可用于数字签名，以代替传统的手工签名。

(1) 简单数字签名

① 发送者 A 使用私有解密密钥对明文进行加密，所形成的密文传送给接收者 B；

② B 利用 A 的公开加密密钥对所得密文进行解密，便得到明文。除了 A 之外，没有人具有解密密钥，因此，也只有 A 才能送出用他的解密密钥加密过的密文；

③ 如果 A 要对自己的行为抵赖，只需出示其解密密钥加密过的密文，使其无法抵赖。

(2) 保密数字签名

上述方法能解决数字签名的问题，但不能达到保密的目的，因为任何人都能接收密文，并可

Key Distribution是对称加密头疼的问题。
如何使得密码安全的给另一方。

http://en.wikipedia.org/wiki/Security-Enhanced_Linux

档经过处理(压缩、加密等)后保存,系统中有专门的备份程序,如 dump/restore、backup;网络备份程序有 rdump/restore、rcp、ftp、rdist 等。

2. 安全操作系统 SELinux

(1) 安全体系结构

2001 年,美国国家安全局(National Security Agency, NSA)发布安全增强 Linux——SELinux,系统的安全体系基于 Flask 结构,Flask 也是由 NSA 开发的安全体系结构。目前,已经在 Linux 内核的主要子系统中实现包括进程、文件和套接字等操作的强制控制机制。

SELinux 安全体系结构如图 7.15 所示,由两部分组成:策略(policy)和实施(enforcement),策略封装在内核于系统安全服务器中,实施由内核对象管理器具体执行,主要特点是通过安全判定与安全实施的分离实现安全策略的独立性。

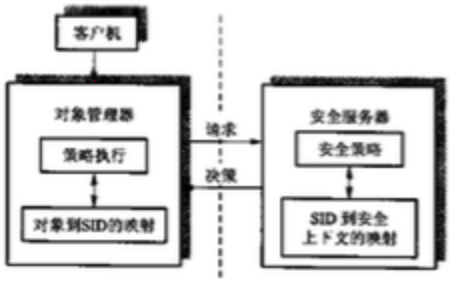


图 7.15 SELinux 安全体系结构

当需要对安全性进行判断时,向安全服务器提出请求,对象管理器只关心 SID。请求到达安全服务器后,实现与安全上下文的映射并进行计算。然后,将决策返回给对象管理器。系统中关于安全的请求和决策有以下三种情况。

① 标签确定(labeling decision):确定一个新的主体和客体采用何种安全标签(如创建主体或客体)。

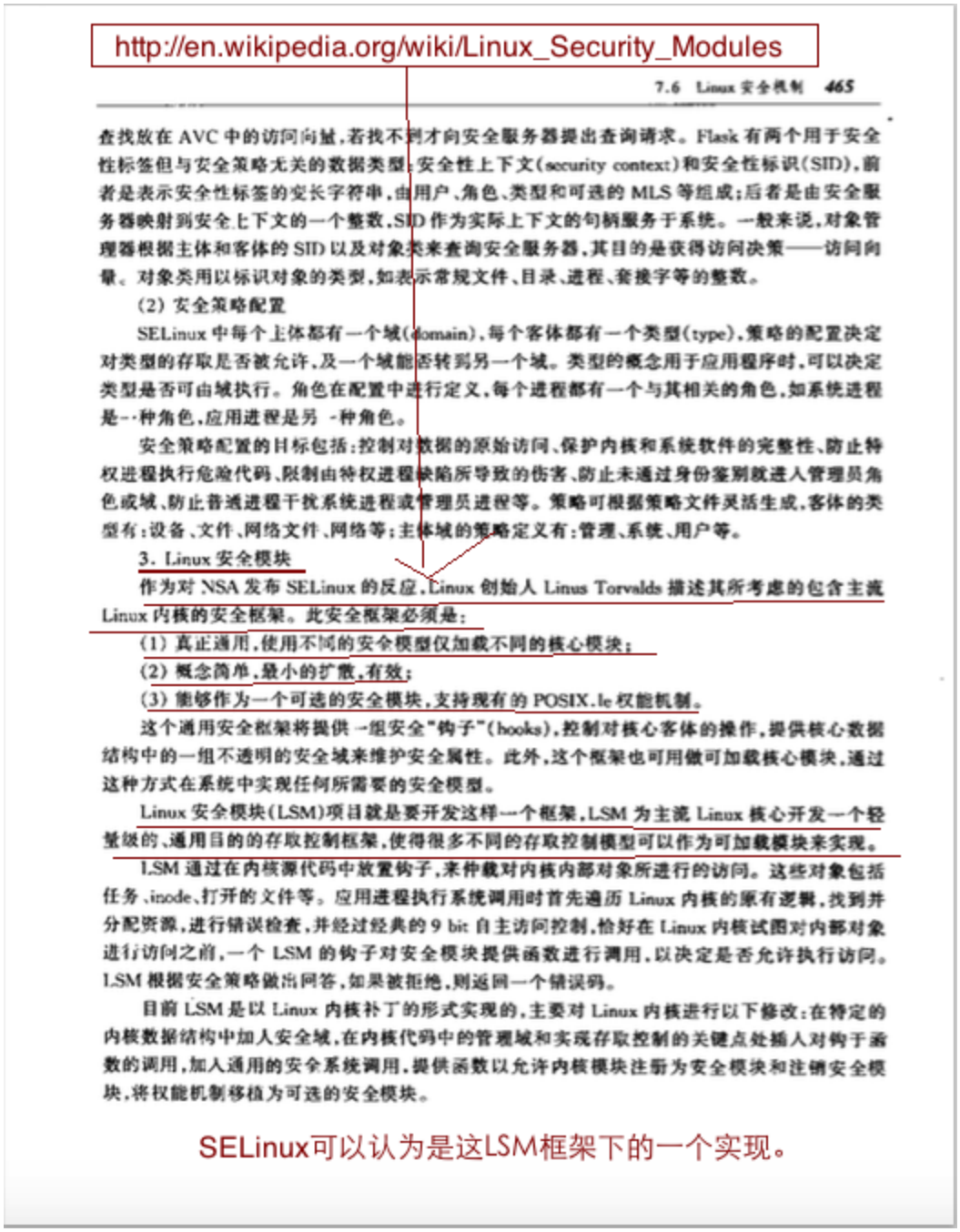
② 存取决策(access decision):确定主体是否能访问客体的某种服务(如文件读写)。

③ 多实例决策(polyinstantiation decision):描述一个特定的请求应该访问多实例化资源中的哪一个成员。例如,TCP 和 UDP 的端口空间资源是固定的,多实例支持用来在端口与 socket 绑定时,进入包的目的端口号存在于多个成员端口空间时,决定采用哪个端口空间。

SELinux 安全服务器实现一种混合型安全策略,包括类型实施(Type Enforcement, TE)、基于角色的访问控制(Role-Based Access Control, RBAC)和可选的多级别安全性(optional MultiLevel Security, MLS)。

Flask 结构还提供一个访问向量缓存(Access Vector Cache, AVC)模块,允许对象管理器缓存访问向量,减少整体性能损失,并实现对动态安全策略的支持。每次做安全性检查时,系统首先

SELinux已经在2.6的Linux的代码树里。



http://en.wikipedia.org/wiki/Linux_Security_Modules

查找放在 AVC 中的访问向量，若找不到才向安全服务器提出查询请求。Flask 有两个用于安全性标签但与安全策略无关的数据类型：安全性上下文(security context)和安全性标识(SID)，前者是表示安全性标签的变长字符串，由用户、角色、类型和可选的 MLS 等组成；后者是由安全服务器映射到安全上下文的一个整数，SID 作为实际上下文的句柄服务于系统。一般来说，对象管理器根据主体和客体的 SID 以及对象类来查询安全服务器，其目的是获得访问决策——访问向量。对象类用以标识对象的类型，如表示常规文件、目录、进程、套接字等的整数。

(2) 安全策略配置

SELinux 中每个主体都有一个域(domain)，每个客体都有一个类型(type)，策略的配置决定对类型的存取是否被允许，及一个域能否转到另一个域。类型的概念用于应用程序时，可以决定类型是否可由域执行。角色在配置中进行定义，每个进程都有一个与其相关的角色，如系统进程是一种角色，应用进程是另一种角色。

安全策略配置的目标包括：控制对数据的原始访问、保护内核和系统软件的完整性、防止特权进程执行危险代码、限制由特权进程缺陷所导致的伤害、防止未通过身份鉴别就进入管理员角色或域、防止普通进程干扰系统进程或管理员进程等。策略可根据策略文件灵活生成，客体的类型有：设备、文件、网络文件、网络等；主体域的策略定义有：管理、系统、用户等。

3. Linux 安全模块

作为对 NSA 发布 SELinux 的反应，Linux 创始人 Linus Torvalds 描述其所考虑的包含主流 Linux 内核的安全框架。此安全框架必须是：

(1) 真正通用，使用不同的安全模型仅加载不同的核心模块；

(2) 概念简单，最小的扩散，有效；

(3) 能够作为一个可选的安全模块，支持现有的 POSIX.1e 权能机制。

这个通用安全框架将提供一组安全"钩子"(hooks)，控制对核心客体的操作，提供核心数据结构中的一组不透明的安全域来维护安全属性。此外，这个框架也可用做可加载核心模块，通过这种方式在系统中实现任何所需要的安全模型。

Linux 安全模块(LSM)项目就是要开发这样一个框架，LSM 为主流 Linux 核心开发一个轻量级的、通用目的的存取控制框架，使得很多不同的存取控制模型可以作为可加载模块来实现。

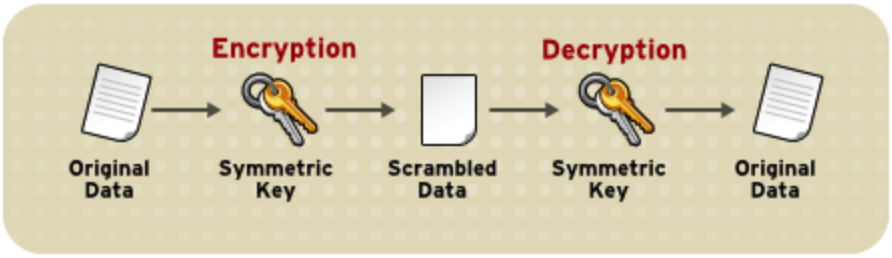
LSM 通过在内核源代码中放置钩子，来仲裁对内核内部对象所进行的访问。这些对象包括任务、inode、打开的文件等。应用进程执行系统调用时首先遍历 Linux 内核的原有逻辑，找到并分配资源，进行错误检查，并经过经典的 9 bit 自主访问控制，恰好在 Linux 内核试图对内部对象进行访问之前，一个 LSM 的钩子对安全模块提供函数进行调用，以决定是否允许执行访问。LSM 根据安全策略做出回答，如果被拒绝，则返回一个错误码。

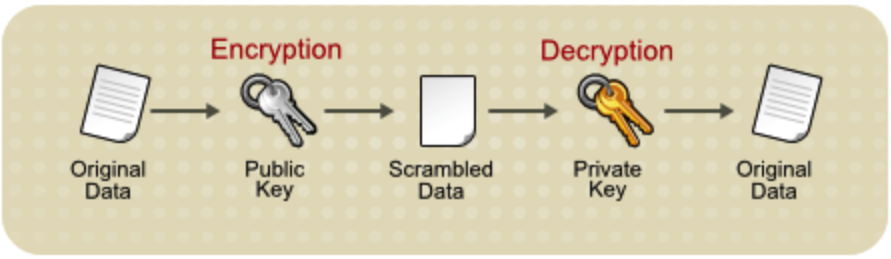
目前 LSM 是以 Linux 内核补丁的形式实现的，主要对 Linux 内核进行以下修改：在特定的内核数据结构中加入安全域，在内核代码中的管理域和实现存取控制的关键点处插入对钩子函数的调用，加入通用的安全系统调用，提供函数以允许内核模块注册为安全模块和注销安全模块，将权能机制移植为可选的安全模块。

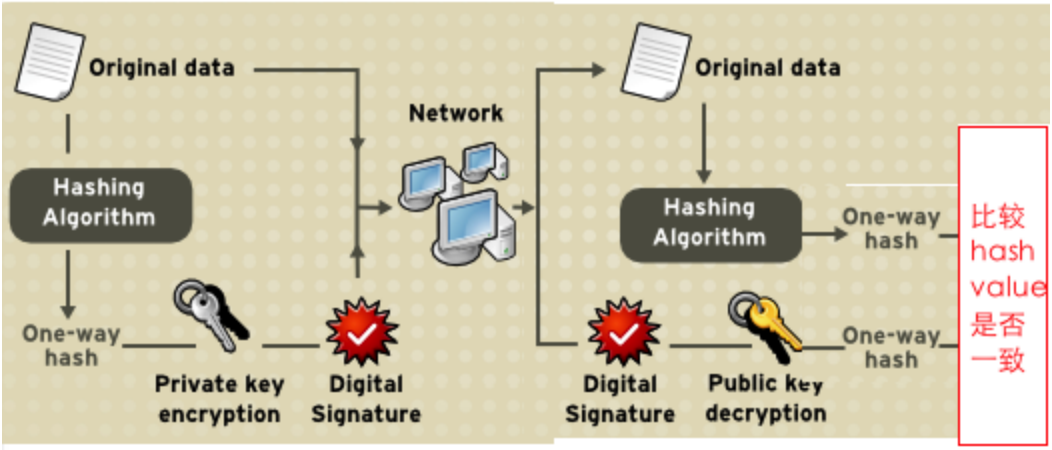
SELinux可以认为是这LSM框架下的一个实现。

[注释] ：LSM：Linux Security Modules（LSM）是Linux领域在可信计算方面的工作，其中NSA主导的SELinux（Security Linux）是基于LSM的一个分支，并已经在2.6进入主线。总体而言，安全的Linux的基本框架是在进程需要获得任何核心资源的时候，需要得到相应的权限和资格检查。

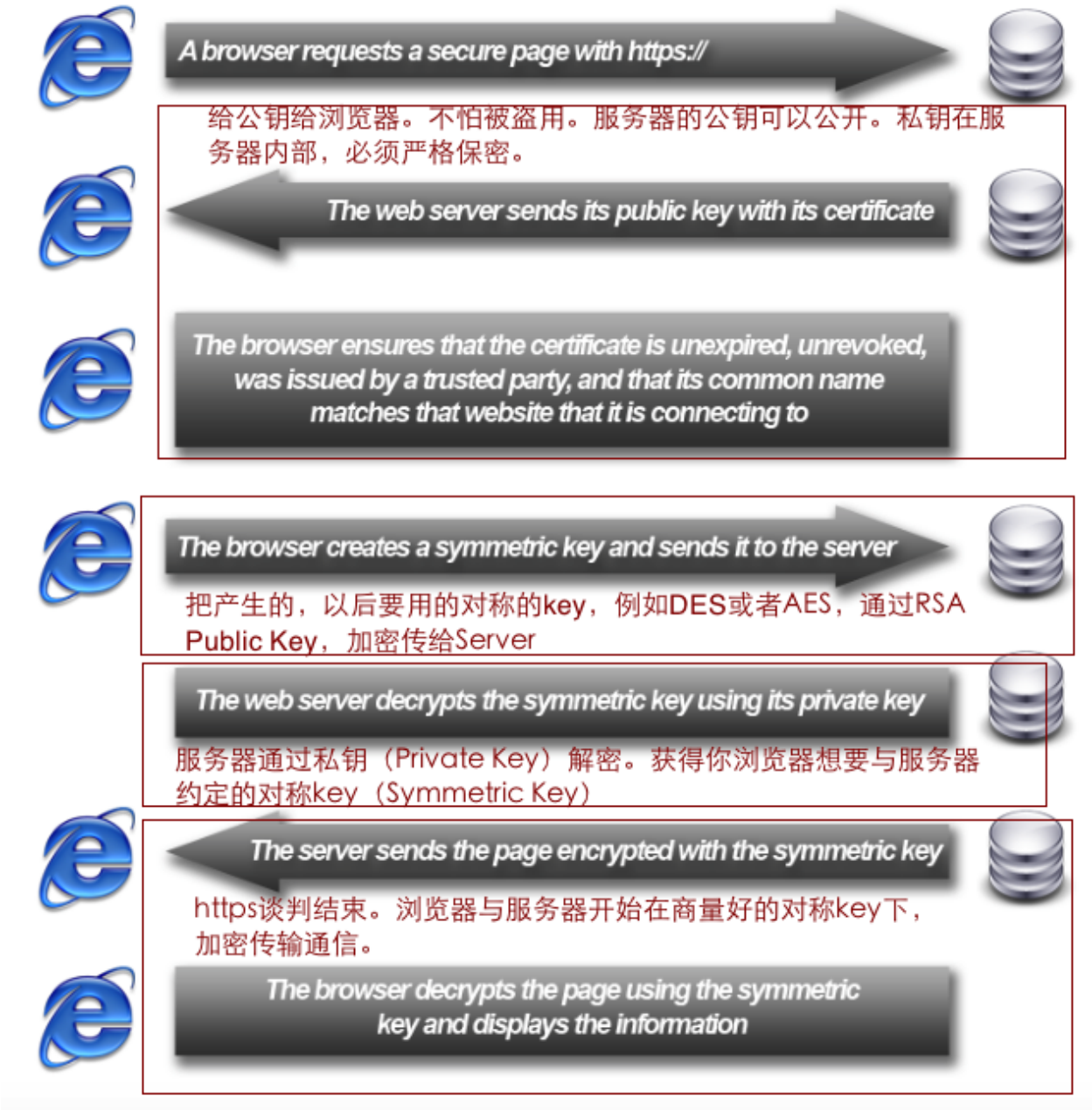
公钥和私钥是成对的，它们互相解密。公钥加密，私钥解密。
私钥数字签名，公钥验证。





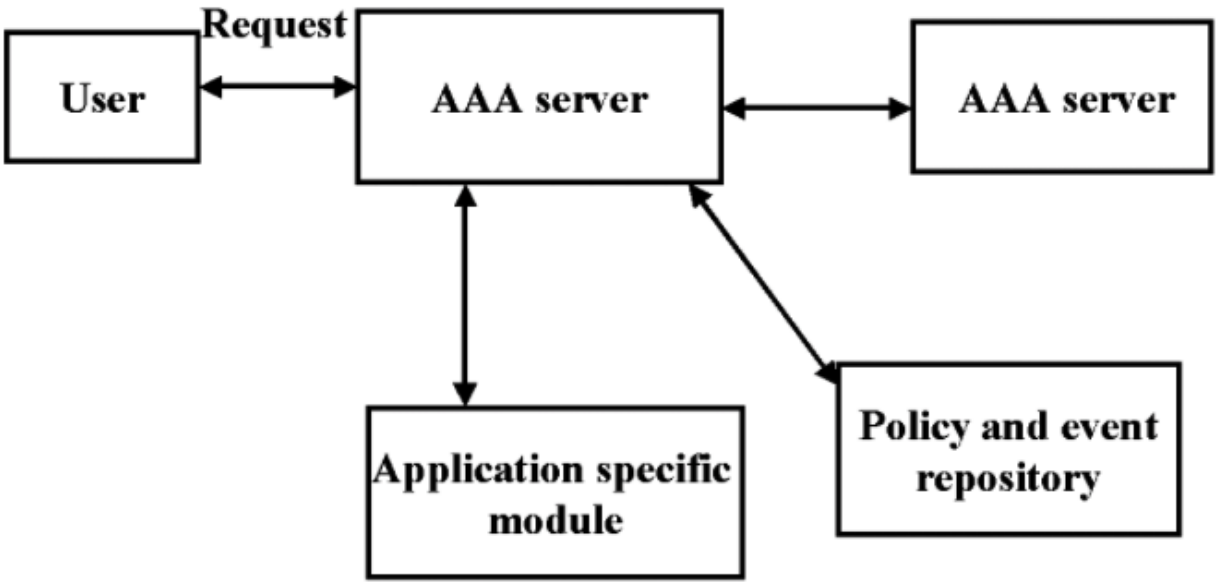


数字签名！ 拿私钥加密一个hash中间结果。 用公钥解密，



[注释] ：加密算法本身是一个学科，操作系统范畴接触一些概念即可。对称和非对称加密（aka，Public Key）算法各有优缺点。可以单独或者混合使用，例如SSL。基本流程如下：利用非对称加密算法，成功协商（交换）对称秘钥（浏览器与web服务器），然后在对称算法下交换数据。

3A（Authentication，Authorization，Accounting）



Authentication：　验证“你是谁”，or “你是否是你说的你是谁”

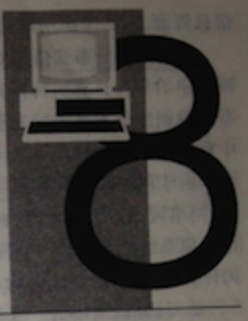
Authorization：　你能被赋予做什么，什么不能做。相关权限。

Accounting：　你在访问期间，做了些什么。

[注释] ： 在操作系统的安全控制中，3A是关于认证（Authentication）,授权（Authorization）和记录（Accounting）的统一安全监管范围。Authentication：你是谁，允许访问否？Authorization：你访问的权限和范围是什么？ Accounting：你访问期间，都干了些什么。

A distributed system is a software system in which components located on networked computers communicate and coordinate their actions by passing messages

第八章



Com

sharing

网络和分布式操作系统

Loosely Coupled；松散式的连接。
例如，通过以太网。

8.1 计算机网络概述

8.1.1 计算机网络的概念

1. 计算机网络的定义和功能

计算机网络是利用各种通信手段，把地理上分散的、具有自治功能的多个计算机系统互连起来，实现数据通信、资源共享、可互操作和协作处理的系统。

计算，存储。

一个计算机网络由三部分组成：

(1) 资源子网：含有若干台计算机，包括拥有资源的计算机(服务器)和请求资源的计算机(客户机)以及各种终端设备；

(2) 通信子网：由通信链路和通信处理机组成，如分组交换器、多路复用器、多路转换器，包组装与拆卸设备、网络控制中心和网关等，用于通信控制和处理；

(3) 通信协议：网络中主机之间、主机和通信子网或通信子网和通信子网各节点之间进行通信的规则。

TCP／IP

计算机网络系统是计算机和网络硬件、网络软件的组合体，除了上述组成部分之外，网络软件由网络操作系统、网络支持软件和网络应用软件组成。网络操作系统用于完成网络的资源分配、共享、管理和协调任务；网络支持软件用于支持数据通信、网络互连、维护管理及各种网络活动；网络应用软件在网络操作系统的基础上向用户提供各种网络应用。

通过计算机网络系统，用户可以进行：

(1) 数据通信：计算机联网后，使地理上分散的计算机之间能够相互通信、交换信息。例如，电子邮件已成为世人所接受的最快捷、经济、方便的通信方式。此外，网上电话、视频会议等许多通信方式也正在迅速发展。

(2) 资源共享：这是计算机网络的主要目的之一。在网络范围内，用户可以共享硬件、软件、

第一个internet msg：1969年10月29日晚上10:30. UCLA与SRI

信息资源，不必考虑用户或资源所在的地理位置。

(3) 支持分布式信息处理：在计算机网络的支撑下，许多大型信息处理问题可借助于分散在网络中的多台计算机协同完成，解决单台计算机无法或极难完成的信息处理任务，于是促进了分布式数据库管理系统的发展，使得分散存储于网络中不同计算机节点上的数据在使用时如同集中存储和处理一样。

由于网络中的计算机可互为备份，让同类资源分布在不同的计算机上，一旦某台机器出现故障，网络可通过不同的路由来访问这些资源，或故障机器的任务可由其他计算机承担，从而提高计算机系统的可靠性。当网络中的某些机器负荷过重时，也可以把一些任务分配给网络中的空闲计算机去完成，提高每台计算机的可用性和使用效率。

2. 计算机网络的产生和发展

计算机网络是计算机与通信技术相结合的产物，始于 20 世纪 60 年代，近 40 年来得到迅猛发展。最早的计算机网络是一台主机通过电话线连接若干远程终端，它是以单个主机为中心的星状网，这类结构称为第一代计算机网络。由美国麻省理工学院林肯实验室为空军所设计的 SAGE 半自动化地面防空系统被认为是计算机和通信技术相结合的先驱。

现代意义上的计算机网络是从 1969 年美国国防部高级研究计划署(DARPA)建成的 ARPANET 试验网和欧洲支持商业应用的 X.25 网开始的，也被称为第二代计算机网络。它以通信子网为中心，由许多主机和终端设备在通信子网的外围构成资源子网，通信子网不再仅仅依赖于类似电话通信的电路交换方式，更多地采用适合数据通信的分组交换方式，网络控制方式也由集中趋于分散，引入专门的通信控制处理器，大大降低了计算机网络的通信成本，这些都是现代计算机网络的典型特征。

计算机网络是一个非常复杂的系统，相互通信的计算机系统必须高度协调地工作。经过多年的研究，人们对网络技术、方式和理论的研究日趋成熟，为了促进网络产品的开发，许多计算机厂商制定自己的网络技术标准。如 IBM 公司的 SNA(System Network Architecture，系统网络体系结构)；DEC 公司的 DNA(Digital Network Architecture，数字网络体系结构)；Univac 公司的 DCA(Data Communication Architecture，数据通信体系结构)；Burroughs 公司的 BNA(Burroughs Network Architecture，Burroughs 网络体系结构)等，这些网络技术标准仅适用于一个公司，不同公司的产品很难互通，终于促成网络国际标准的制定。1977 年，国际标准化组织(ISO)成立"计算机与信息处理标准化委员会(TC97)"下属的"开放系统互连分技术委员会(SC16)"，在现有各种网络标准的基础上，制定"开放系统互连参考模型"(OSI-RM)作为网络的国际标准，推动网络技术进一步发展，迈向第三代计算机网络的时代。

20 世纪 80 年代中期以来，计算机网络领域最引人注目的事件是因特网的飞速发展，它仍然属于第三代计算机网络，有自己的网络体系结构，未完全使用 OSI-RM。进入 20 世纪 90 年代后，计算机网络的发展更加迅速，目前正在向宽带综合业务数字网(BISDN)的方向演变，这就是通常所说的第四代计算机网络。1993 年 9 月，美国政府提出国家信息基础设施计划(National Information Infrastructure，NII)，俗称"信息高速公路"，其重要内容之一就是建设一个覆盖全美

分层结构是网络系统最重要的一个概念和机制。

国的宽带综合业务数字网。

8.1.2 网络体系结构

1. 网络体系结构

计算机网络是一个复杂的大型系统，其实现过程中要解决很多技术问题，如支持不同通信介质、支持不同机器平台、支持多种通信规范、支持多种应用业务等，实现计算机网络时通常采用层次结构，将其分成若干层，每一层完成特定的功能，各层协调起来构成整个网络系统。网络体系是指完成计算机之间的通信协作，把互联的功能划分成有明确定义的层次，把网络的同层进程通信协议及邻层接口称为网络体系结构（network architecture）。具体地说，网络体系结构是关于计算机网络应设置几层、每层应提供哪些协议的精确定义，可见，它从层次结构和功能上来描述计算机网络结构，并不涉及每层硬件和软件的组成和实现问题。对于同样的网络体系结构，可以采用不同的方法设计出完全不同的硬件和软件，相应的层次提供完全相同的功能和接口。

2. 通信协议

在计算机网络中，为了保证数据通信双方能够正确、自动地进行通信，必须在信息传输顺序、信息格式和信息内容等方面制定一套规则和约定，这种规则和约定的集合就是网络协议（network protocol）。

由于网络体系结构是有层次的，因而，通信协议也被分为多个层次，每一层内还允许分成若干子层次，协议的各层次有高低之分。另外，通信协议应是可靠、有效的，否则会造成通信的混乱和中断。网络通信协议含有以下 3 个要素：

（1）语义：是指对构成协议的元素含义的解释。例如，在基本型数据链路控制协议中规定，数据报文中的第一个协议元素的语义表示所传输报文的报头开始，接着是报头；而第二个协议元素的语义表示正文的开始，接着是正文，等等。

（2）语法：规定将若干协议元素和数据结合起来表示一个完整内容时所应遵循的格式。例如，在传送数据报文时，协议元素和数据组合的次序为：报头开始符、报头；正文开始符、正文、正文结束符；最后是奇偶校验码。

（3）规则：规定事件的执行顺序和匹配速率，如通信双方进行发送和应答的次序。

综上所述，网络协议实质上是网络中互相通信的对等实体间交换信息所使用的一种语言。

3. 开放系统互连参考模型

为了实现计算机系统的互连，开放系统互连参考模型（Open Systems Interconnection Reference Model，OSI-RM）把网络通信功能划分成顺序的七层模型，即物理层、数据链路层、网络层、传输层、会话层、表示层、应用层，每层实现各自的功能，通过各层接口和功能的组合与其相邻层连接。各层的主要功能介绍如下。

（1）物理层：为通信提供物理链路，实现数据链路实体之间比特（bit）流的透明传输，处理与物理传输介质有关的机械、电气和信号的特性和接口，维持或拆除数据链路实体之间的物理连

Protocol。例如，建立一次TCP会话的握手过程。

接，并实现数据的发送和接收。

Link Layer

(2) 数据链路层：分为介质访问控制和数据链路控制子层，前者解决网络多用户竞争信道使用权的问题；后者负责建立、维持和释放数据链路的连接、数据传输时的流量控制、链路控制和差错控制，提供数据帧及无差错地传送以帧为单位的数据。

(3) 网络层：负责把数据从物理连接的一端送到另一端，其主要任务有：路由选择和中继，网络连接和多路复用，数据分段和组块，阻塞和流量控制。Network Layer

(4) 传输层：通过向会话层提供可靠的端到端的服务，使其上层与下三层隔离开来，提供进程间的通信机制，保证数据传输的可靠性，其具体功能有：地址映射、多路复用、数据分段与重装、组块与分块。

(5) 会话层：在两个相互通信的应用进程之间建立、组织和协调其交互，提供会话控制和传输同步控制。

(6) 表示层：处理数据的语法表示问题，实现信息转换，包括：格式表示、数据压缩、加密和解密等功能。

(7) 应用层：提供通用应用程序，包含各种应用协议，例如，文件传送协议 FTP、存取和管理协议 FTAP、虚拟终端协议 VTP、报文处理系统协议 MHS、作业传递及操作协议 JTM 等。

采用 OSI 七层模型的网络中，不同计算机上的应用进程在进行数据通信时，其信息流动过程如下：源计算机系统的应用进程把欲发送的数据传送至最高层，由其在用户数据前面加上这一层的控制信息，形成最高层数据单元后送至次高层；次高层又在数据单元前面加上这一层的控制信息，形成次高层的数据单元后，把它传送至其下一层。以下各层以此类推，信息按此方式逐层向下传送直至最低层，由于最低层实现比特流传送，故不必再添加控制信息。当比特流经过传输介质到达目标系统时，依次按由低向高的次序逐层向上传送，且在每一层都依照相应的控制信息完成指定操作，再把本层的控制信息去掉，剩余下来的数据单元向上一层传输，依次类推，当数据最后到达目标系统应用层时，再由应用层将用户数据交给目标计算机系统的接收进程，这样便结束一次通信过程。

4. TCP/IP 网络体系结构

De Facto：事实上的工业界标准

除了 OSI 之外，还流行其他网络体系结构，其中最广为人知的是 TCP/IP 协议，虽然此协议并非国际标准，但由于其简洁、实用，在计算机网络中占有很重要的地位，成为事实上的工业标准。

传输控制协议(Transmission Control Protocol，TCP)和网际协议(Internet Protocol，IP)是一个协议集，表示因特网所使用的一种网络体系结构，其主要特点是：适用于多种异构网络的互连，遵从可靠的端－端协议，与操作系统紧密结合，效率高且有较好的网络管理功能。

TCP/IP 技术是为容纳物理网络技术的多样化而设计的，是一个完整的传输协议，位于网络层 IP 之上，为进程提供高可靠的通信能力。TCP 提供基于连接的数据流管道传输，所采用的可靠性技术有：确认和超时重传、流量控制和拥塞控制等。IP 解决兼容性问题，由于各种网络技术的帧格式、地址格式等上层协议之可见因素差别很大，通过 IP 数据报和 IP 地址将它们统一起来，达到屏蔽低层细节，向上提供统一服务的目的。TCP/IP 协议的层次如图 8.1 所示。

通常不太纠结OSI模型。了解概念即可。

IP是一个Best Efforts的协议。不保证可靠传输。

TCP/IP 与 OSI 之间有不少差别，共有三层，顶层是应用层协议，相当于 OSI 的高三层；与 OSI 传输层相当的是中层传输控制协议 TCP 或 UDP；与 OSI 网络层相当的是底层网际协议 IP。TCP/IP 没有对更低层次做出规定，这是因为在设计时考虑到要与具体的物理传输介质无关。TCP/IP 协议集的主要内容和组成有：

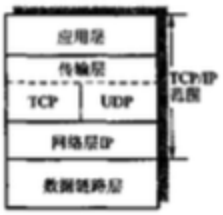


图 8.1 TCP/IP 协议的层次

(1) 应用层：对应于 OSI 的应用层、表示层和会话层。常用的协议有：简单邮件传送协议 SMTP，多用途因特网邮件扩充协议 MIME，域名系统 DNS，远程登录协议 Telnet，文件传送协议 FTP，网络基本 I/O 系统 NetBIOS，超文本传送协议 HTTP，等等。

(2) 传输层：对应于 OSI 的传输层。主要协议有：传输控制协议 TCP，提供可靠的数据传输服务；用户数据报协议 UDP，提供无连接、高效的数据报传送服务。

(3) 网络层：对应于 OSI 的网络层，提供主机间的数据传输能力。主要协议有：网际协议 IP，为传输层提供网际传输服务；网际控制报文协议 ICMP，允许主机或路由器报告有关 IP 服务状况；地址转换协议 ARP，把 IP 地址转换成网络物理地址；反向地址转换协议 RARP，将网络物理地址转换成 IP 地址。

(4) 数据链路层：提供 TCP/IP 与各种物理网络的接口，包括广域网和局域网、Ethernet、X.25、Token Ring 等。

5. IP 和 IPv6

http://en.wikipedia.org/wiki/IPv4

TCP/IP 协议体系结构的关键是 IP 协议，1995 年，开发互联网协议标准的因特网工程任务组(Internet Engineering Task Force，IETF)发布下一代互联网的规范，将其命名为 IPng(IP Next Generation)，此规范提出在新一代网络中用 IPv6 协议取代现有的 IPv4 协议。它增强了 IP 网络的许多功能，以适应更高网络速度及图像和视频应用所形成的混合数据流。推动新的 IPv6 协议发展的主要原因是对更大的网络地址空间的需求，现有的 IP 协议使用 32 位网络地址来表示目标地址和源主机地址，随着互联网呈爆炸式增长，大量网络连接到互联网，现有地址长度已经不能满足所有系统需要，IPv6 提供 128 位长的源和目的地址域。采用 IPv6 的网络将比现有网络具有更好的可扩展性、安全性和服务质量，最终，使用 TCP/IP 的所有系统都会从目前的 IP 体系

IPV4 报文头格式

Version	Hd Len	Type of Service	Total Length	
Identification			Flags	Fragment Offset
Time to Live		Protocol	Header Checksum	
Source Address				
Destination Address				
Options				Padding
Data Portion				

20 octe

Vari leng

http://en.wikipedia.org/wiki/Software-defined_networking

安全的数据传输，方便地实现网络资源共享，为网络用户提供各种服务。

典型的网络操作系统的特征有：硬件独立性，网络操作系统可以运行在不同的网络硬件上，可通过网桥或路由器与其他网络连接；多用户支持，能同时支持多名用户对网络的访问，应对信息资源提供安全和保护功能；支持网络实用程序及其管理功能，如系统备份、安全管理、容错和性能控制；多种客户端支持，如 Windows NT 网络操作系统可支持 OS/2、Windows 98、Windows for Workgroup、UNIX 等多种客户端，方便网络用户的使用；提供目录服务，以单一逻辑的方式让用户访问位于全世界范围内的所有网络服务和资源的技术；支持多种增值服务，如文件服务、打印服务、通信服务、数据库服务、万维网服务等；可操作性，这是网络工业的发展趋势，允许多种操作系统共享相同的网络电缆系统，且彼此可连通访问。网络操作系统分成以下三种类型：

1. 集中模式

集中式网络操作系统是由分时操作系统加上网络功能演变而成的，系统由一台主机和若干台与主机相连的终端构成，信息的处理和控制都是集中的。早期的 UNIX 分时系统是这类系统的典型例子。

2. 客户－服务器模式

Client／Server

网络中连接多台计算机，其中，一部分称为服务器，提供文件、打印、通信、数据库访问等功能，提供集中的资源管理和安全控制。另外一部分称为客户机，它向服务器请求服务，如文件下载和信息打印等。服务器的配置通常很高，运算能力强，有时还需要专职网络管理员来维护；客户机与集中式网络中的终端不同，它有独立的处理和计算能力，仅在需要某种服务时才向服务器发出请求。这一模式的特点是信息的处理和控制都是分布的，因而，又称分布式处理系统。NetWare 和 Windows NT 是这类操作系统的代表。

客户－服务器模式在逻辑上归入星状结构，以服务器为中心，与各客户之间采用点到点通信方式。当今两种主要的服务器应为：文件服务器和数据库服务器。无论是哪一种，客户机在请求服务器的服务时，双方需要多次交互：客户机发送请求包、服务器接收请求包、服务器回送响应包、客户机接收响应包。这种结构的优点是：数据分布存储、数据分布处理、应用编程较为方便。

3. 对等模式

让网络中的每台计算机同时具有客户机和服务器两种功能，既可向其他机器提供服务，又可向其他机器请求服务，网络中不存在中央控制手段。对等模式适用于工作组内几台计算机之间仅需提供简单的通信和资源共享的场合，也适用于把处理和控制分布到每台计算机的分布计算模式，NetWare Lite 和 Windows for Workgroup 是这类网络操作系统的代表。对等模式的优点是：平等性、可靠性和可扩展性较好。

8.2.2 网络操作系统实例

SDN是网络系统下一步发展的重要方向。通过集中的控制，从而能够最有效率的控制网络元素。 Software Defined Network

［注释］： 网络与分布式系统的界限比较模糊。共性是：都是基于松散式互联架构的拓扑，例如，基于以太网。 分布式系统比较强调资源（Resource）的透明性（Transparent），例如，一个提供的服务在哪个网络的节点上，对服务的请求者是透明的。

在学术范畴，透明性是分布式系统的一个重要的特征。在实际工程中，有时不会过分去追求100%的透明性，也不太可能。

8.3 分布式操作系统

8.3.1 分布式系统概述

分布式系统(distributed system)是由一组独立的计算机系统，经互连网络连接而形成的"单计算机系统"(single computer system)。通常满足以下条件：

(1) 系统中的任意两台计算机可通过系统的安全通信机制来交换信息；

(2) 系统中的资源为所有用户共享，且无须考虑资源的物理位置，即为用户提供对资源的透明访问；

(3) 系统中的若干台机器可互相协作来完成同一项任务，即一个程序可分布于多台计算机上并行运行，所以，分布式系统是一种特殊的计算机网络系统；

(4) 系统中的一个节点出错不影响其他节点运行，具有较好的容错性和稳健性。

分布式系统要让用户使用起来像"单计算机系统"，实现分布式系统以达到这一目标的技术称为透明性。透明的概念适用于分布式系统的各个方面：位置透明性，是指用户不知道包括硬件、软件及数据库等在内的系统资源所处的位置，资源名中也不应包含位置信息；迁移透明性，是指资源无须更名就可从一个节点自由地迁移到另一个节点；复制透明性，是指系统可以任意复制文件或资源的多个副本，而用户无须对此有所了解；并发透明性，是指用户不必也不会知道系统中同时还存在其他用户与其竞争使用资源；并行透明性，是指在分布式系统中解决大型应用时，可由系统(编译、操作系统)自动找出潜在的并行模块去分布执行，而不为用户所察觉，这是分布式系统追求的最高目标。

用于管理分布式系统的操作系统称为分布式操作系统，它具备以下 4 项基本功能。

(1) 进程通信：提供有力的通信手段，让运行在不同计算机上的进程之间可以交换数据；

(2) 资源共享：提供访问其他机器资源的功能，使得用户可以访问或使用位于其他机器上的资源；

(3) 并行计算：提供某种程序设计语言，使用户可以编写分布式程序，此程序可在系统中多个节点上并行运行；

(4) 网络管理：高效地控制和管理网络资源，对用户具有透明性，即使用分布式系统与传统单机系统相似。分布式计算机系统的优点是：坚定性强、易扩充、可靠性好、便于维护和效率较高。

为了实现分布式系统的透明性，至少需要以下机制：一是有单一全局性的进程通信机制，在任何一台机器上，进程都采用同一种方法与其他进程通信；二是有单一全局性的进程管理和安全保护机制，进程的创建、运行和撤销以及保护方式不会因机器不同而异；三是有单一全局性的资源管理机制，隐蔽资源的分布，控制资源的分配，例如，提供统一的文件系统，用户存取文件的方式和单机一致。

分布计算的目的：（1）. 资源透明性 or （2）. 并行计算。过去20年来，（2）占据主要发展方向，例如，Google的系统。

Messaging Passing和RPC，特别是RPC是实现Client／Server语义的重要技术手段。

分布式系统与网络系统的基础都是网络技术，在硬件连接、拓扑结构和通信控制等方面基本一样，都具有数据通信和资源共享功能，其主要区别在于：在网络系统中，用户在通信或资源共享时必须知道网络的构成、计算机及资源的位置，通常通过远程登录或让计算机直接相连来传输信息或进行资源共享；而在分布式系统中，用户在通信或资源共享时并不知道有多台计算机存在，其数据通信和资源共享如同在单计算机系统上一样。此外，互联的各计算机可互相协调工作、共同完成一项任务，可以把一个大型程序分布在多台计算机上并行运行。虽然分布式和网络操作系统都要管理通过网络联结的多台计算机，就像传统单机操作系统一样，对管辖的所有资源进行抽象和虚化，达到隐蔽硬件细节、提供接口、共享系统资源的目的，但它们的本质区别是呈现在用户面前的透明程度不同，分布式操作系统为用户提供的是一台具有分布处理功能的虚拟机；而网络操作系统为用户提供一组可互相通信的虚拟机。

8.3.2 分布式进程通信

分布式系统的主要特性之一是分布性，它源于具体应用的需求，应用程序可分布于分散的若干台计算机上运行；而通信则来源于分布性，因为机器的分散要求必须通过通信来实现进程的交互、合作和资源共享，可见分布式系统中的通信机制十分重要。

分布式系统中的底层进程通信有三种：消息传递机制、远程过程调用（Remote Procedure Call，RPC）和套接字（socket），它们都依赖于网络的数据传输功能。此外，为了解决互操作性，上层应用程序和下层通信软件之间增加一层标准的编程接口及协议——中间件（middleware），用来为分布式程序设计提供更好的支持和服务，目前已得到广泛的应用。

1. 消息传递机制 Messaging Passing

最简单的分布式消息传递模型称为客户－服务器模型，客户进程请求服务，如读入数据、打印文件，向服务器进程发送一条包含请求的消息，消息按照通信协议的规定来传递；服务器进程接收到消息后完成请求内容并返回结果。在设计分布式消息传递机制时，需要妥善解决目标进程寻址和通信原语的设计问题。

（1）目标进程寻址 例如，基于IP地址

客户向服务器发送消息，首先必须知道服务器的地址，这样消息才能到达此机器；其次，如果目标机器上有多个进程运行，那么消息到达机器后应该传递给哪个进程来处理呢？这就要解决目标进程的寻址问题。

第一种方法采用机器号和进程号寻址法：机器号用于使消息能正确地发送到目标机器上，进程号用来确定消息应传递给哪个进程。每台机器上的进程编号都从0开始，无须互相协调，因为进程 i 在机器1与机器2上不会发生混淆。这种方法的缺点是位置的不透明性，用户必须要知道服务器的地址。

第二种方法采用广播寻址法：让分布式系统中的进程在硕大且专用的地址空间中选择自己的标识号，如64位二进制整数空间内，两个进程选择同一数值的可能性极小。那么，发送消息时，发送到哪一台机器上呢？在支持广播通信的网络中，发送者广播一个特殊的定位包，包含目

name server，存放metadata（元数据），例如，地址信息。

标进程的地址。由于网络中的所有机器都能接收，由内核检查此进程是否在本机上。如果回答消息"我在这里"并给出机器号，发送消息的内核使用并记住这个地址，避免下次再广播。这种方法虽然具有位置透明性，但其缺点是广播通信将增加系统开销。

第三种方法采用名字服务器寻址法：系统中增加一台机器，由其提供机器名和机器地址的映射，称为名字服务器。客户机中存放 ASCII 码形式的服务器名，每次客户机运行并发送消息给一台服务器时，先发出请求消息给名字服务器，询问被请求服务器所在的机器号，有了这个地址，就可以直接发送消息。这种方法的缺点是要增加服务器及所发送的消息量。

(2) 同步和异步通信原语

分布式进程通信原语分为同步和异步两种，其基本形式为：

发送／接收 的配套

·Send(P,Message)　·Receive(Q,Buffer)

其中，P 表示接收进程，Q 表示发送进程，Message 表示要发送的消息，Buffer 则为接收者进程的信箱或缓冲区。在分布式系统中，进程标识实际上要由计算机地址和进程标识号组成。

同步发送时，发送进程要求接收进程做好接收消息的准备，在发送完消息后阻塞，并等待接收进程的回答。如果在预定时间内未收到回答，则认为消息已经丢失，应重新发送。同步发送消息的缺点是系统的并行性差，且无法利用广播消息功能，而广播功能在很多场合是很有用的。

异步发送时，发送进程把消息发送出去后，并不阻塞自己，而是继续运行。可见发送进程和消息传送可以并行工作，使系统的并行性能提高，异步通信的缺点是在消息被发送出去前，发送进程不能修改发送的消息缓冲区，它不知道传输何时进行，无法知道何时可以重新使用缓冲区。解决这个问题有两个方法：一是由发送原语把消息复制到系统缓冲区，允许发送进程继续工作；二是当消息被发送出去后，向发送进程发出中断并通知它缓冲区可用。

使用可以实现进程的同步（Sync功能／语义

与发送消息相类似，接收消息也分为阻塞型和非阻塞型，在此不再细述。此外，一个进程可能希望不加选择地向一组欲接收消息的进程中的任意一个进程传递消息，一种受限的广播形式叫做"多播"发送，可向一组命名进程中的每个成员传送消息。类似地，进程可以按照消息的到达顺序来接收多个发送进程中的任意一个进程所发送的消息。一种使用隐含命名的接收消息操作可用于这个目的。阻塞型的接收消息和发送消息是解决各种进程协作问题的有力机制；广播发送往往使用非阻塞形式；阻塞型的接收比非阻塞形式更为有用，如打印或文件服务器进程接收客户端的服务请求消息，接收消息使用隐含命名的阻塞形式，因为事先并不知道发送者是谁，应该接收来自一组客户端的任何一个客户端发来的消息。

(3) 缓冲和非缓冲原语

非缓冲原语不使用缓冲区，有一个指向特定进程的地址，一个 receive(addr,&m)调用告诉内核，此调用过程正在地址 addr 上监听，并准备接收一条发送到此地址上的消息，m 所指向的消息接收区将保存消息。当消息到达时，接收方内核把它复制到消息接收区，然后释放被阻塞的接收进程。

第一种方法是服务器方先调用 receive，客户端后调用 send。这种策略会工作得很好，但当 send 在 receive 之前被调用时，对于新来的消息，服务器内核如何知道哪个进程正在使用此地址？

Group Communication，Multicst

IP：Best Efforts的传送。 是不可靠的。可能丢失报文。

消息又被复制到哪里呢？答案是"不知道"。一种简单的策略是：忽略这条消息，等到客户端超时，这样服务器就可以在客户重传消息之前调用 receive。这种方法虽然容易实现，但可能导致客户端内核重传多次。如果多次重传失败，客户端内核会放弃发送。

第二种方法是让接收方内核保留所到达的每条消息一段时间，直到相应的 receive 被调用。每到达一条消息就启动一个计时器，若在相应的 receive 被调用之前计时器超时，则此消息被丢弃。从理论上说，保留消息一段时间可定义新的数据结构"信箱"，准备接收消息的进程请求内核创建信箱，并指明一个地址以便查找网络信息包，以此地址标识的消息都被放入信箱中。对 receive 的调用只是从信箱中读取消息，若无消息可读则阻塞进程，采用这一技术的原语称为缓冲原语。信箱可连接多个发送者和多个接收者，非遗憾，在分布式环境中，接收消息的代价太高，因为指向同一个信箱的接收者可能驻留在不同的节点上。遗常使用信箱的变种"端口"，端口仅与一个接收者相关联，从不同进程发来的指向同一个端口的消息都被送往一个与接收者相关联的位置。

(4) 可靠和非可靠通信原语

通常设想发送进程发送消息，服务器进程会接收到消息。在实际的通信过程中，消息可能会丢失，这样会影响消息传递模型的语义。解决这个问题的方法：一是尽可能保证传送，使用可靠的传输协议，附带完成检错、确认、重传、重排序，使消息正确到达目的地；二是发送进程所在机器的内核与接收进程所在机器的内核进行应答，仅当发送方收到接收方发来的确认消息后，发送方内核才释放客户进程。

RPC 构建了在网络环境下的函数调用语义。具备重大意义。

2. 远程过程调用 — RPC

发明者：Bruce Jay Nelson

远程过程调用是分布式系统中广泛采用的进程通信方法，它把单机环境下的过程调用拓展到分布式环境中，允许不同计算机上的进程使用简单的过程调用和返回结果方式进行交互，但这个过程调用是用来访问远程计算机所提供的服务，用户却感觉像在执行本地过程调用一样。在实现上，普通过程与远程过程不一样，后者在不同计算机上的地址空间中运行，它不能成为调用过程的一部分，相反地，必须有一个独立进程来执行被调用的过程，这个进程可在每次调用时动态地创建，也可作为专用服务进程静态地创建。

为了能以相同方式完成本地过程调用和远程过程调用，在客户－服务器上增设客户代理和服务器代理，当客户机上的进程需要调用服务器上的一个过程时，发出一条带参数的 RPC 命令给客户代理，它收到 RPC 命令后，便执行本次远程过程调用，把参数打成消息包，执行 send 原语请求内核把消息发送到服务器，通常客户代理调用 receive 原语阻塞自己直至服务器发来应答。消息到达服务器之后，远地内核把消息传送给服务器代理，接着，它拆包从消息中取出参数，然后，以一般方式创建或调用一个服务器进程，此进程执行相应的过程调用并以一般方式返回结果。当过程调用完毕后，服务器代理获得控制权，把结果打成消息包，再调用 send 原语请求内核把消息发回调用者。整个过程调用结束后，服务器代理回到 receive 状态，等待下一条消息。消息送回客户机后，内核找到消息并把它送给客户代理，客户代理检查并拆开消息包，取出结果返回给调用进程，调用进程获得控制权并得到本次过程调用的结果。图 8.2 说明远程过程调用的

http://en.wikipedia.org/wiki/Remote_procedure_call

http://en.wikipedia.org/wiki/Bruce_Jay_Nelson

stub：处理网络传送细节。通常通过RPC 工具生成。

流程，可把 RPC 的执行步骤总结如下。

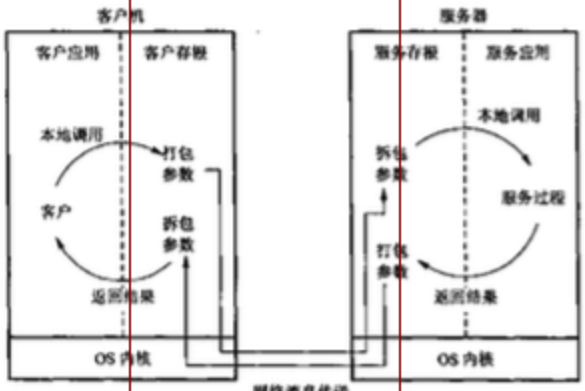


图 8.2 远程过程调用机制

(1) 客户进程以普通方式调用客户代理；
(2) 客户代理组织 RPC 消息并执行 send，激活内核程序；
(3) 本地内核通过网络把消息发送到远地内核；
(4) 远地内核把消息送给服务器代理；
(5) 服务器代理取出消息中的参数后调用服务器进程；
(6) 服务器进程执行完后把结果返回至服务器代理；
(7) 服务器代理将结果打包并激活内核；
(8) 服务器内核通过网络把消息发送至客户内核；
(9) 客户内核把消息交给客户代理；
(10) 客户代理从消息中取出结果返回给客户进程；
(11) 客户进程获得控制权并获得过程调用的结果。

上述 RPC 仅适用于同构型分布式系统，运行相同的操作系统，采用同种程序设计语言，有一致的参数表示方法；反之，对于异构型分布式系统来说，由于不同系统对数据的表示方式不同，导致客户机和服务器无法交互，故应该在 RPC 设施中增加参数表示的转换机制。例如，对于整数、浮点数、字符和字符串提供一个标准格式，这样，任何机器上的本地数据和标准表示均按规定作相应的转换。

3. 套接字

套接字是 UNIX BSD 首创的网络通信机制，提供进程通信端口，socket 是面向客户－服务器模式设计的低层通信机制。每个 TCP 及 UDP 的协议头都含有源端口和目的端口域，端口号指

socket 其实更是一个编程接口。可以归结为message passing。

socket可以stream（TCP），datagram（UDP），或者RAW（IP）打开。从而应用程序可以在不同的层面编程

通常也说RAW的模式说Layer 2的模式。

明两个相互独立的 TCP 单元使用者(应用程序)，同时，IP 包含源地址和目的地址域，IP 地址指明两个相互独立的主机系统，端口号和 IP 地址结合起来唯一地表示一个应用在因特网上的地址。

服务器端套接字通常保持 TCP 或 UDP 端口处于打开状态，等待未知时间的连接呼叫，客户端通过查找域名系统(DNS)数据库，确定所期望的服务器的套接字标识号，一旦连接建立，服务器端就将会话切换至其他的端口号，以释放主端口号并等待其他连接请求。

套接字定义应用程序编程接口 API，它是编写程序使用 TCP 或 UDP 进行通信的通用编程接口，套接字 API 使用协议、IP 地址、端口号来共同定义唯一的套接字。有两种类型的套接字 API：流套接字和数据报套接字。流套接字使用 TCP 协议，提供面向连接的可靠数据传输，在两个套接字之间所传送的所有数据包能够保证以发送时的顺序递交和到达目的端；数据报套接字使用 UDP 协议，它不像 TCP 那样提供面向连接的特性，传输可靠性得不到保证，也不能确保数据的顺序递交。Socket API 主要有：创建 socket()，指定本地地址 bind()，建立连接 connect()，愿意接收连接 listen()，接收连接 accept()，发送数据 send()、sendto()、sendmsg()、write()、writev()，接收数据 recv()、read()、readv()、recvfrom()、recvmsg()。

套接字是 TCP/IP 网络通信的基本构件之一，由于基于 TCP/IP 协议的应用一般采用客户－服务器模式，因此，在实际应用中，必须有客户端和服务器两个进程，且首先启动服务器进程，通过无连接协议或有连接协议服务的调用时序进行工作。服务器的工作方式还可分为两种类型：重复服务器和并发服务器，前者面向短时间内能处理完的客户请求，由服务器自己处理到来的请求；后者面向需要长时间才能处理完的客户请求，每接到一个客户请求，创建一个子进程响应它，自己退回等待态，监听新的客户请求，形成一个主服务器和多个从服务器并发执行的局面。

RPC是技术支柱。在Java里，通常叫做RMI。

4. 客户－服务器计算模型和中间件 Client／Server Middleware

(1) 客户－服务器计算模型

客户－服务器模型是一种基于局域网或广域网的分布式系统，例如，万维网基本上是一个客户－服务器系统，用户是客户机，Web 站点是服务器。客户－服务器模型的思想是：把操作系统看做一组协作进程，为用户提供各种服务，用户称为客户，协作进程称为服务器，客户和服务器通常运行相同的操作系统，且都作为用户进程运行，一台机器可以运行一个进程，也可以运行多个客户进程、多个服务器进程或两者的混合。为了避免面向连接协议所造成的大量开销，客户－服务器模型通常基于简单的、无连接的请求/应答协议，客户向服务器发送一条请求消息，要求提供某些服务(如查询信息)，服务器工作并返回所需要的数据或出错代码，请求和应答消息都要经过内核进行。

客户－服务器模型的优点是简洁性和高效率，此模型所用协议集比 OSI 的要小，因而效率高。如果所有机器都相同，那么只需要三层协议：物理层、数据链路层和请求应答层(相当于 OSI 的会话层)，前两层协议处理客户与服务器之间来往的数据分组，这是由硬件处理的，不需要路由选择，也不需要建立连接。在请求/应答层有相应的协议，它定义一组合法的请求和应答，由于结构简单，操作系统的通信机制只需提供两个系统调用：发送消息和接收消息，这两个系统调用可

例如，BEA，Oracle公司的许多为电子商务提供的后台系统。

通常说 3 Tier结构。

通过函数库加以封装。

传统的客户－服务器体系结构包括两层：客户层和服务器层。近年来，三层结构模型变得日益普遍，在这种结构中，应用软件分布在三种类型的机器上：用户层机器（客户）、中间层机器（应用服务）及后端服务器（数据资源）。用户层机器一般是瘦型客户机；中间层机器基本上是位于客户和很多后端数据库之间的网关，它能够转换协议，将对一种类型的数据库查询映像为另一种类型的数据库查询，另外，中间层机器能够融合来自不同数据源的结果，它还因介于两个层次之间而可充当桌面应用程序和后端应用程序之间的网关；中间层机器和后端服务器之间的交互也遵从客户－服务器模型，因此，中间层同时充当着客户机和服务器的角色。

此外，基于多层客户－服务器模型结构的应用框架使用得越来越广泛，Sun 公司的 J2EE 体系结构可用于构建复杂的 Web 应用，J2EE 采用四层结构：Web 浏览器、Web 服务器、应用服务器和数据资源服务器。用户通过 Web 页面提交请求和接收结果；Web 服务器负责接收来自用户的请求并转交给应用服务器，另一方面负责接收来自应用服务器的计算结果并组织成 Web 页面返回给客户端；应用服务器负责具体的逻辑运算，在需要时访问数据资源服务器以获得计算所需的数据资源；数据资源服务器只负责应用数据的保存和维护，且对外提供访问数据的接口。

(2) 中间件 Middleware

① 中间件的概念

客户－服务器计算模型的开发和使用对分布计算的标准化提出很高的要求，缺乏标准化使得实现集成的、多厂商的、企业范围的客户－服务器配置变得十分困难，因为客户－服务器方式的优势是与其模块化及将平台和应用程序混合、协调来提供商业解决方案的能力紧密相联的，这种"互操作性"问题必须得到很好的解决。

实现这一要求的一种方法是：在上层应用程序和下层通信软件及操作系统之间使用标准编程接口和协议，建立支持在异构网络、异构计算机和不同操作系统中访问系统资源的工具，这种标准化的接口、协议和工具被称为"中间件"。

目前已经有很多中间件软件包，有些简单而有些非常复杂，其共同特点是能隐蔽不同网络协议和操作系统的复杂性和不一致性。客户机和服务器厂商一般都提供很多流行的中间件软件包以供选择，这样，用户可以采取一种特定的中间件策略，然后从厂商那里集成设备来支持这种策略。

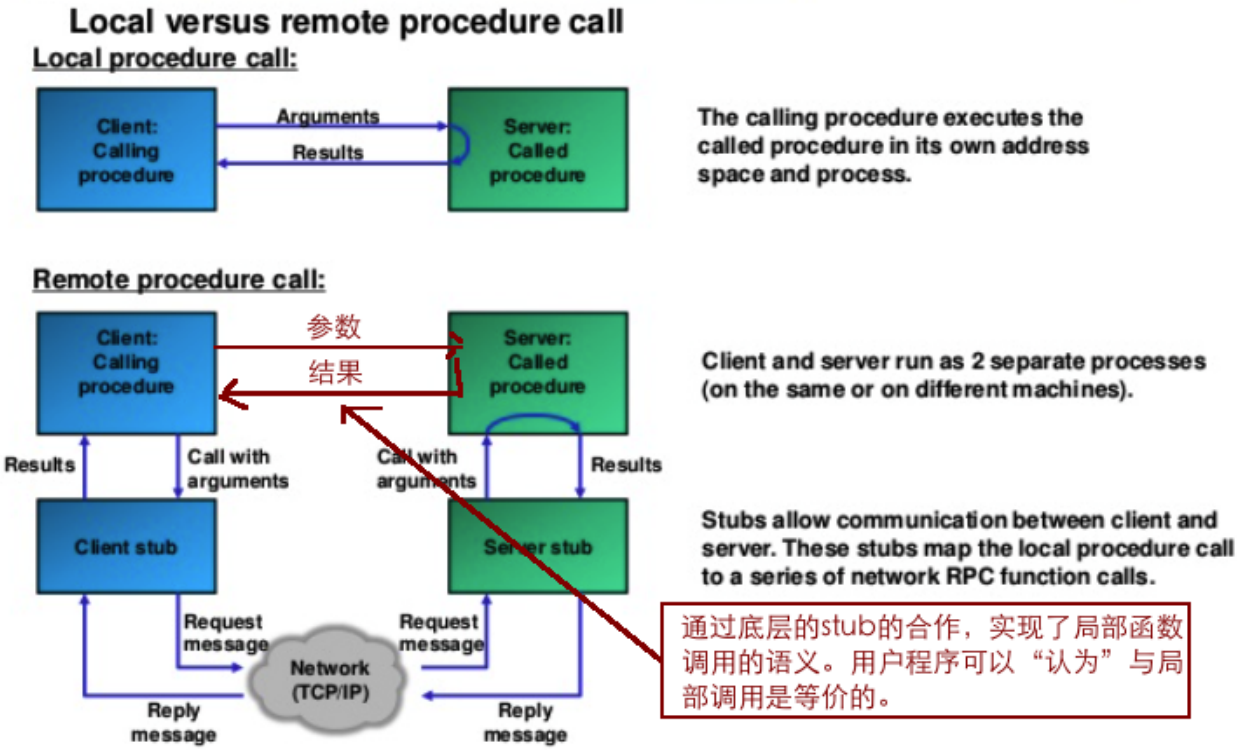
② 中间件体系结构

图 8.3 给出在客户－服务器体系结构中的中间件的作用，中间件的确切作用将取决于所使用的客户－服务器计算的类型，即应用程序的功能划分。中间件具有客户端组件和服务器端组件两个部分，其基本作用是使位于客户端的应用程序或用户能够访问服务器上的各种服务，同时无须考虑服务器之间的区别。例如，对于特定的应用领域，结构化查询语言 SQL 提供本地或远程用户访问关系数据库的标准访问方式，然而，很多数据库厂商都对 SQL 进行特定的扩展，这就使得众多产品存在差别，产生潜在的不兼容性问题。

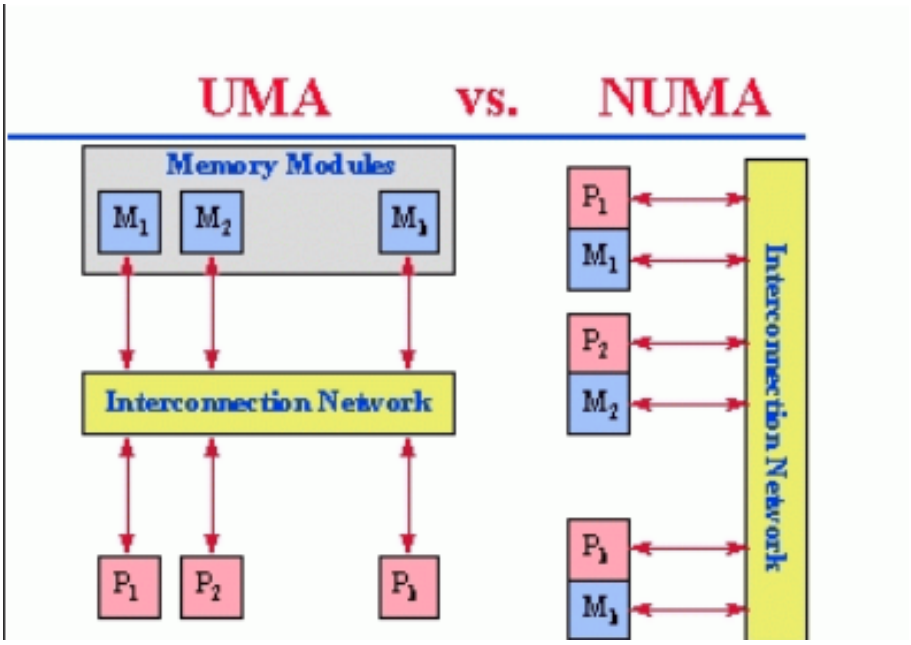
中间件提供一个软件层，其目的是隐蔽下层网络操作系统和平台的异构性，支持不兼容系统的统一访问，提供一致的分布式服务接口，把底层服务综合起来，以一种更透明、更高级的形式为

中间件的作用就是提供大量的软件模块，从而使得应用程序可以聚焦业务，不要去关心网络细节和一些底层处理。

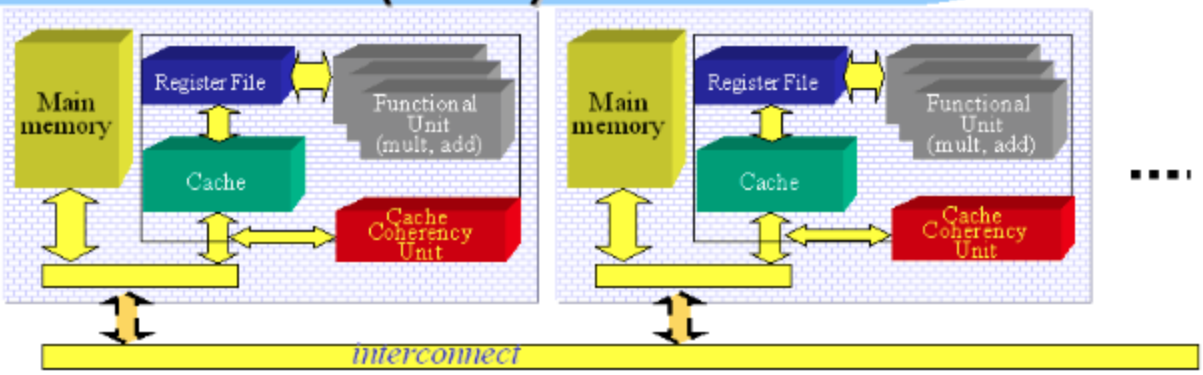
[注释]： 分布式系统有两个不能完全统一的目标：1.资源使用的透明性 2. 松散式的并行计算。过去20年来谷歌等互联网公司成功的把第2种模式良好的实践。而经典的追求资源透明性的系统并没有成熟和普及。“牺牲透明性，追求并行性”是现代分布式计算的一个折衷（Tradeoff）



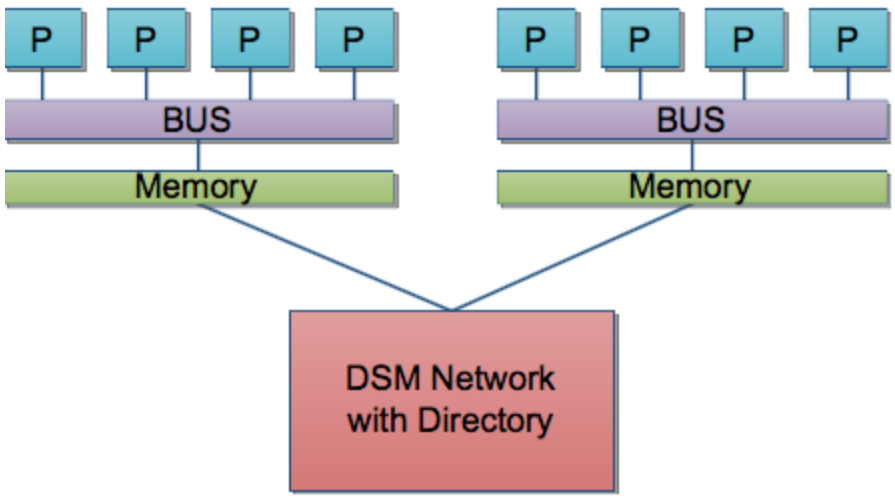
[注释]：RPC为分布式系统提供了重要的局部函数调用语义，使得应用不需要过分关注网络细节，例如网络编程。RPC变种有Java/RMI，OMG/CORBA和一些基于http承载的RPC, 例如XML-RPC。不管通信是基于L3的TCP，L7的HTTP，目标都一致：局部函数调用语义从而简化分布式编程模型。

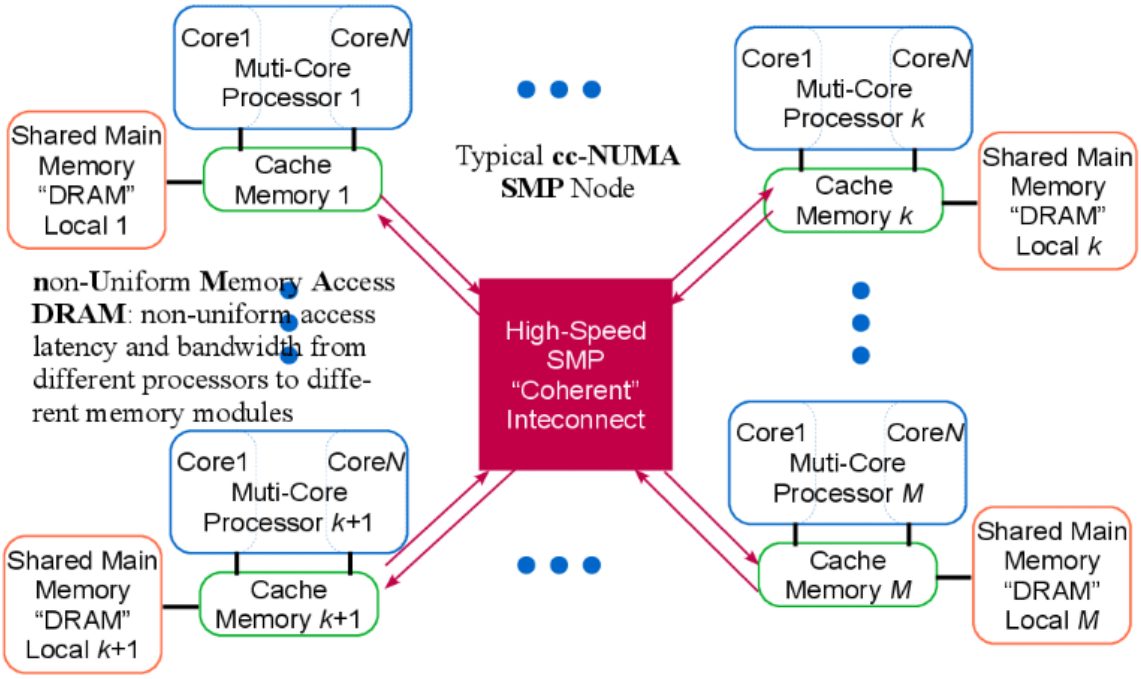


Distributed Shared Memory Architecture (DSM)



- **Local memory and independent path to memory as with the Multicomputer Architecture**
- **Memory of all nodes is organized as one logical "shared memory"**
- **Non-uniform memory access (NUMA):**
  - **"Local memory" access is faster than "remote memory" access**
- **Programming model is (almost) the same as for the Shared Memory Architecture**
  - **data distribution is available for optimization**
- **Scalability properties similar to the Multicomputer Architecture**





[注释] ：类似RPC，DSM是为了给应用程序提供一个类似局部访问内存的编程模型。通过一个数据一致性算法，保证各个CPU / 节点看见一个一致的内存空间。UMA：多个CPU直接能够通过内存总线访问内存。NUMA：通过互联网络（例如，infiniband）的拓扑。通常NUMA也缺省指ccNUMA。

Lamport 是2013年的图灵奖获得者。他的主要贡献是分布式系统理论，形式方法描述和消息机制。

种种进程同步方式不再适用。如果两个进程运行在一台机器上，它们就能共享信号量，并通过执行系统调用来访问并发生交互作用；如果进程运行在不同的机器上，这种方法就不起作用。采用完全分布管理方式时，每个资源由位于不同节点上的资源管理者共同管理，某个资源管理者在决定分配其所管理的资源之前，必须和其他资源管理者协商，对于计算机系统的共享资源都要采用这种管理模式。因此，必须设计一个算法，各资源管理者按此算法共同协商资源的分配，此算法应满足：资源分配的互斥性，不产生饥饿现象，且各资源管理者处于平等地位而无主控者。通常称这种算法为分布式同步算法，由同步算法所构成的机制称为分布式同步机制。

实现分布式进程同步比实现集中式进程同步要复杂得多，由于进程分散在不同的计算机上，只能根据本地可用信息做出决策，并通过网络通信联系。由于消息通过网络传递时会有一定的延迟，当资源管理者接收到不同机器上的进程同时发来的资源申请时，先接到的申请的提出时间，很可能晚于后接到的申请的提出时间，所以，必须要解决对不同计算机中发生的事件进行排序的问题。然后，再设计性能优越的分布式同步算法。

1．事件排序

分布式系统没有公共的时钟。

进程同步的实质是对多个进程在执行顺序上施加规定，为此，应对系统中所发生的事件进行排序。由于在分布式系统中，各台计算机无公共时钟，也无共享存储器，很难确定系统中所发生的两个事件的先后次序。1978 年，Lamport 提出不使用物理时钟而对分布式系统中所发生的事件进行排序的方法。如果两个进程是无关的，那么，它们的时钟根本无须同步，对于相交进程而言，通常并无必要让进程在绝对时间上完全一致，只要限定它们执行的先后次序就行。

同一个节点上的事件可以通过按照其节点上的时间先后顺序，定义其“－－>”关系。

定义关系“先发生”，将其表达为“a→b”，读作“a 在 b 之前发生”，意思是指系统中的所有进程认为事件 a 先于事件 b 发生。有三种情况产生“先发生”关系：

情况(1)：如果 a 和 b 是同一进程中的两个事件，且 a 发生在 b 之前，则 a→b 为“真”；

情况(2)：如果 a 是一个进程的发送消息事件，b 为另一个进程接收此消息事件，则 a→b 为“真”；

情况(3)：存在某个事件 c，若有 a→c 且 c→b，则 a→b 为“真”。 关系是传递的。

情况(1)说明同一进程中所发生的事件之间，可按发生时间的先后次序来确定“先发生”关系；情况(2)指出消息决不能在发送之前就已接收，因为传送消息的过程会有时间延迟，发送消息事件总是先于接收消息事件发生；情况(3)说明“先发生”关系具有传递性。

按照上述方法定义事件的“先发生”关系之后，同一进程中两个事件的先后关系被明确确定；不同进程中所发生事件间的先后关系，一部分可被确定，另一部分则不能确定。例如，有 3 个进程 P_1、P_2 和 P_3，它们分别发生以下事件：

事件 a：P_1 发送消息给 P_2；

事件 b：P_2 接收来自 P_1 的消息；

不能确定关系的事件，叫做并发事件。

事件 c：P_2 接收到 P_1 的消息后，发送消息给 P_3；

事件 d：P_3 接收来自 P_2 的消息。

显然，有 a→b→c→d。然而，假设 P_2 在事件 b 之前发生过某事件 f，如打印输出，尽管可以确定：

$$f \rightarrow b, f \rightarrow c, f \rightarrow d$$

Lamport的这个叫做“happen before”，因果序。例如，A－－>B。表示A发生在B之前。如果不存在"－－>"关系，两个事件认为是并发的（Concurrent）。

http://en.wikipedia.org/wiki/Lamport_timestamps

这些关系，但是 a 和 f 之间的先后关系却无法确定。如果两个事件 x 和 y 发生在不同的进程中，且这两个进程不交换信息，那么 x→y 和 y→x 都不成立，这两个事件称为并发事件。简单地说，无法确定这两个事件孰先孰后。

2. 逻辑时钟

Lamport 定义的逻辑时钟又称时间戳(timestamp)，是指能为系统中的所有活动赋予一个编号的机制，可利用整数计数器来实现。定义逻辑时钟的实质是把一个系统中的事件映射到正整数集合上的一个函数 C，并满足：若事件 a 先发生于事件 b，则 $C(a)<C(b)$，此处 $C(a)$和 $C(b)$分别是事件 a 和事件 b 所对应的逻辑时钟函数值，系统中的每个进程都拥有自己的逻辑时钟。

构造逻辑时钟函数的方法很多，任何满足上述映射关系的正整数函数都可作为逻辑时钟。下面是逻辑时钟函数的一种构造方法，定义在某系统集合上的逻辑时钟函数 C 如下。

(1) 对任一进程 P 中的非接收消息事件 e_j，若 e_j 是 P 的第 1 个事件，则

$$C(e_j)=1$$

若 e_j 是 P 的第 j 个事件，而第 $j-1$ 个事件是 e_{j-1}，则

$$C(e_j)=C(e_{j-1})+1$$

(2) 对于任一进程 P 中的接收消息事件 e_r，若 e_r 是 P 的第 1 个事件，则

$$C(e_r)=1+C(e_{r'})$$

此处，$e_{r'}$是进程 P′发送这条消息的事件；若 e_r 是 P 的第 r 个事件，而 P 的第 $r-1$ 个事件是 e_{r-1}，则

$$C(e_r)=1+\max[C(e_{r-1}),C(e_{r'})]$$

定义逻辑时钟后，可对一个系统中的所有事件人为地排出前后关系。对于由 n 个进程所组成的网络系统，逻辑时钟可用于对由消息传输所组成的事件进行排序，网络中的每个节点 i 都维护一个本地计数器 C_i，其功能相当于本地时钟。每次系统发送消息时，首先把时钟加 1，再发送一条消息，形式为(m,T_i,i)(其中，m 为消息内容，T_i 为此消息的逻辑时钟，i 为节点编号)。分布式系统中的进程可以拥有自己的逻辑时钟(各个节点上的本地时钟)，而这些时钟并非同步运行，可能出现这种情况：一个进程发送消息中所含的逻辑时钟大于接收进程收到此消息时它所具有的逻辑时钟，由于发送消息事件必定出现在接收消息事件之前，故需要调整接收进程的逻辑时钟。所以，接收进程 j 按照上述规则(2)应将其时钟设置为当前值和到达的逻辑时钟值这两者取最大值再加 1。由于逻辑时钟只能向前走，即为单调递增，所以校正逻辑时钟值是加至少为 1 的正数。在每个节点上，事件的排序由下列规则确定：对于来自站点 i 的消息 x 和来自站点 j 的消息 y，若下列条件之一成立，则说事件 x 先发生于事件 y。

(1) $T_i<T_j$；　(2) $T_i=T_j$ & & $i<j$

与每条消息有关的时间是附加在消息上的时间戳，时间的顺序是通过上述两条规则确定的。

现在来讨论事件排序规则的原理，看看 Lamport 提出的时钟同步算法如何校正系统中发生事件的逻辑时间。如图 8.5 所示，考虑有 3 个进程并发工作，它们运行在不同的机器上，每台机

逻辑时钟是一个偏序是因果序。系统中可以存在没有关系的并发事件

使得接受事件的逻辑时钟必须大于发送事件。从而避免乱序，和保证在一个全局的整数集合上，有一个统一的映射。

必须调整P2在接收事件C的时候，时钟要大于P3的60.否则，违反了因果关系。

器都带有自己的时钟，且按照自己的速度计时。在进程 P_0 中，时钟滴答为 6；而此时在进程 P_1 中，时钟滴答为 8；而在进程 P_2 中，时钟滴答为 10；每个时钟都按照恒定的速率计时，但不同机器上的晶体振荡存在差别，造成各自的时钟速率不一样。

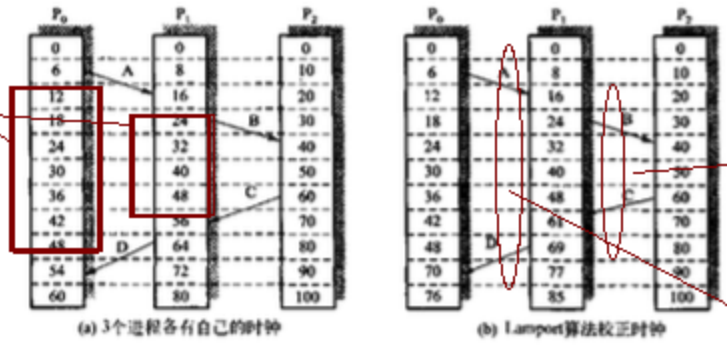


图 8.5 应用 Lamport 算法的示例

在时钟滴答 6，进程 P_0 发送消息 A 给进程 P_1，这条消息花多长时间到达目的地，取决于基于哪一个时钟来计算。如果进程 P_1 接收到这条消息是在时钟滴答 16，而同时消息 A 所携带的时钟滴答为 6，那么，进程 P_1 可认为此消息在路上花费 10 个时钟滴答，这是完全可能的时间值。同样道理，消息 B 从进程 P_1 传送到进程 P_2 花费 16 个时钟滴答，这也是完全可能的时间值。

现在来看消息 C，它从进程 P_2 传送到进程 P_1，开始发送时的逻辑时钟为 60，而消息到达时的时钟滴答为 56；类似地，消息 D 从进程 P_1 传送到进程 P_0，开始传送时的时钟滴答为 64，而消息到达时的时钟滴答为 54。显然这些时间值是不可能的，要对其加以校正。

Lamport 的时钟校正方法是从"先发生"关系直接得来的。因为消息 C 被发送时的时钟滴答为 60，当它到达时，其时钟滴答值应为 61 或大于 61。因此，让每条消息都携带其发送者的时钟所确定的发送时间，当消息到达目的地时，若接收者的时钟当时的指示值先于消息的发送时间，接收者的时钟值就应校正为发送时间加 1 之后的值。从图 8.5(b)可以看出，消息 C 到达的时间现在是 61；同样，消息 D 到达的时间是 70。

增加一个附加条件，Lamport 算法就能满足系统中全局性时间的需要，为所有事件确定全局顺序关系。所增加的条件是：在两个事件之间，时钟至少需要滴答一次。如果一个进程在很短的时间内快速发送或接收两条消息，那么，必须调整时钟，使时钟在这两个事件之间至少要前进一个滴答。

在某些情况下，上面的算法需要满足条件：任何两个事件都不会恰巧在完全相同的时刻发生。为了满足这个关系，对具有相同时间戳的两条消息则通过其所在的站点编号来排序。这种规定能够避免各通信进程的不同时钟之间的重合问题。

属于并发事件。没有直接的因果关系。谁大，谁小，没关系。不影响系统的稳定性。可以并发。

Lamport的序其实是表达了：因果关系。
“后发生的事件是因为前面事件造成的”

对分布式系统资源的集中式管理是趋势。例如，最近几年开始兴起的SDN。

(3) 令牌环算法

Suzuki 在 1982 年提出的用于分布式系统的另一种同步算法是令牌环算法，为了实现进程互斥，所有进程构成一个逻辑环(logical ring)，环中的每个进程都有前驱和后继，这样的逻辑环可由软件实现，并不意味着任何物理拓扑结构。系统中设置一个象征存取权力的令牌(token)，它是一种特定格式的报文，不断地在进程所组成的逻辑环中循环。

利用令牌实现进程互斥的过程如下：令牌在初始化后，被逻辑环中的任意一个进程获得，这样令牌开始绕环移动，它从进程 K 传递给它的下一个进程 $K+1$，可按照点到点方式进行传递。当进程从其前驱那里得到令牌时，它就检查所欲进入的临界区。如果临界区是开放的，则此进程进入临界区，访问共享资源；当此进程退出临界区时，便把令牌传送给其后继，不允许使用同一张令牌进入第二个临界区。当进程得到前驱传递来的令牌，又不想进入临界区，就把令牌往后传，这样一来，如果没有进程想进入临界区，令牌就会在逻辑环中高速循环运行。这种算法能实现进程互斥是显而易见的，由于只有一张令牌，任何时刻只有一个进程拥有令牌，所以只有一个进程可以进入临界区。由于令牌以固定的顺序移动，存在着前驱优先于后继的关系，也不会出现饥饿现象，一旦某个进程想进入临界区，最坏的情况是等待所有其他进程进入后再退出临界区所耗费时间的总和。

8.3.5 分布式系统中的死锁

1. 死锁类型

在网络和分布式系统中，死锁的防止、避免和解除等方法与单处理器系统相似，但其难度和复杂度要大得多。在分布式环境下，由于进程和资源的分布性，竞争资源的诸进程来自不同的节点，拥有共享资源的每个节点，通常只知道本节点中的资源使用情况，因而，检测来自不同节点中的进程在竞争共享资源时是否会产生死锁，显然是很困难的。分布式系统中的死锁分为两类：资源死锁和通信死锁。资源死锁是因为一组进程竞争系统中可重复使用的资源，如打印机、磁带机以及存储器等而引起的，进程的推进顺序如果不当，很容易引发系统死锁。通信死锁是指：在不同节点中的进程，为发送和接收报文而竞争缓冲区，出现既不能发送，又不能接收的僵持状态。

2. 分布式死锁检测与预防

客观地存在两种检测方法。

(1) 集中式死锁检测

分布式系统中的每台计算机都有一张进程资源图，用以描述进程及其占有资源的状况。一台中心计算机上拥有一张整个系统的进程资源图，当检测进程检测到环路时，就中止一个进程以解决死锁。检测进程必须适时地获得从各个节点发送来的更新信息，可采用以下方法解决更新问题：一是每当进程资源图中添加或删除一条弧时，就应将相应的变动信息发送给检测进程；二是每个进程周期性地把自己从上次更新之后新添加或删除的弧的信息发送给检测进程；三是检测进程在需要的时候主动请求更新信息。

SDN：Software Defined Network。通过一个（逻辑上）集中的控制器，来管理松散的网络和分布式系统。

工程实践中，往往是静态分配比较实用。显示的事先划分好。

④ 每个进程应将资源的分配情况通知所有进程。

由此可见，为了实现分布式系统环境下的死锁检测，通信开销相当大，而且还有可能出现假死锁，因而，在实际应用中，主要还是采用死锁预防方法。

为了防止在网络中出现死锁，可采取破坏产生死锁的4个必要条件之一的方法来实现。为了防止发生资源死锁，第一种方法可以采用静态分配方法，即所有进程在运行之前，一次性地申请其所需要的全部网络资源，这样，进程在运行过程中不会再提出资源申请，破坏"占用和等待"条件。如果网络系统无法满足进程的所有资源请求，索性连一个资源也不分配给此进程，这样也能预防死锁。第二种方法是按序分配，把网络中可供共享的网络资源进行排序，同时要求所有进程对网络资源的请求严格按照资源号从小到大的顺序提出，这样可以防止在资源分配图中出现循环等待事件。第三种方法主要解决报文组装、存储和转发造成缓冲区溢出而产生的死锁问题，为了避免发生组装型死锁，源节点的发送进程在发送报文之前，应先向目标节点申请一份报文所需要的全部缓冲区，如果目标节点无缓冲区，索性一个也不分配，让发送进程等待。为了避免存储转发型死锁，可为每条链路上的进程配置一定数量的缓冲区，且不允许其他链路上的进程使用，或者当节点使用公共缓冲池时，系统限制每个进程只能使用一定数量的缓冲区，而留出足够大的后备缓冲空间。

通过一个集中仲裁对象，分发ticket。

8.3.6　分布式文件系统

1. 分布式文件系统概述

分布式文件系统是分布式系统的重要组成部分，是允许通过网络来实现互连的、使不同机器上的用户共享文件的一种文件系统，其任务也是存储和读取信息，许多功能都与传统的文件系统相同。分布式文件系统不是一个分布式操作系统，而是一个相对独立的软件系统，它被集成到分布式操作系统中，并为其提供远程访问服务。分布式文件系统具有以下特点。

(1) 网络透明性

客户访问远程文件服务器上的文件的操作如同访问本机文件的操作一样。

(2) 位置透明性

客户通过文件名访问文件，但并不知道此文件在网络中的具体位置；同理，文件的物理位置若改变，但只要文件名不变，客户仍然可以对其进行访问。

2. 分布式文件系统的组成

分布式文件系统由两部分组成：运行在服务器上的分布式文件系统软件和运行在客户机上的分布式文件系统软件，两部分程序代码在运行过程中都要与本机操作系统的文件系统紧密配合，共同起作用。由于现代操作系统都支持多种类型的文件系统，因此，本机上的文件系统均是虚拟文件系统，它支持多个具体的文件系统，分布式文件系统将通过虚拟文件系统与本机文件系统交互作用。例如，Sun公司的网络文件系统NFS由以下几部分组成：网络文件系统协议、远程过程调用协议、扩展数据表达、网络文件系统服务器代码、网络文件系统客户机代码、安装协议、服务器监听进程、服务器安装进程、客户机I/O进程和网络锁定管理器及状态监视器。

例如，Google的GFS，Hadoop中的HDFS。都是分布式文件系统。

http://en.wikipedia.org/wiki/Google_File_System

http://en.wikipedia.org/wiki/Live_migration

移到其他系统中运行，提高系统的可用性。

2. 迁移的实现

为了实现进程迁移，在分布式系统中必须建立相应的进程迁移机制，主要负责解决：由谁来发动进程迁移；如何进行进程迁移；如何处理未完成的信号和消息等问题。

进程迁移的发动取决于进程迁移机制的目标，如果目标是平衡负载，则由系统中的监视模块负责在适当时刻进行进程迁移。在分布式系统中配置系统负荷监视模块，设定其中一个节点上的是主模块，主模块定时地与各个系统的监视模块交互有关系统负荷情况的信息。一旦发现有些系统忙碌，而有些系统空闲时，主模块便启动进程迁移，向负荷沉重的系统发出命令，让其将若干进程迁移到负荷轻的系统中去。当然，对于用户而言是透明的，进程迁移工作均由系统完成。类似地，如果进程迁移是为了其他目标，则分布式系统中的其他相应的部分成为进程迁移的发动者。

在进程进行迁移时，应撤销系统中的已迁移进程，在目标系统中建立一个相同的新进程，因为这是进程的迁移而不是进程的复制。进程迁移时，所迁移的是进程映像，包括进程控制块、程序、数据和栈，此外，被迁移进程与其他进程之间的关联应作相应的修改。

进程迁移的过程并不复杂，但是需要一定的通信开销，困难在于进程地址空间和已经打开的文件。由于现代操作系统均采用虚拟存储技术，对于进程地址空间可使用以下两种方法：一是传送整个地址空间，把进程的所有映像全部从源系统传递到目标系统，这种方法简单，但当地址空间很大且进程只需用到一部分程序和数据时，会造成资源浪费。二是仅传送主存中已修改的那部分地址空间，若程序运行时还需要附加的虚存空间部分信息，则可通过请求方式予以传送。这样，所传送的数据量是最少的，但源系统中仍然需要保存被迁移进程的数据及相关信息，源系统并未从对此进程的管理中解脱出来。三是预先复制，进程继续在原节点上运行，而地址空间被复制到目标节点上，由于原节点上的某些地址空间内容又被修改过，所以，需要有两次迁移，这种方法能够减少进程被冻结的时间。如果被迁移的进程已打开系统中的某些文件，可采用两种方法来处理，一是将已打开的文件随进程一起迁移，这里存在的问题是：进程有可能仅作临时迁移，返回时才需要访问文件；二是暂时不迁移文件，仅当迁移后的进程又提出对文件的访问要求时，再进行迁移。如果文件被多个分布式进程所共享，则需要维护对文件的分布式访问，而不必迁移。

在一个进程由源系统向目标系统迁移期间，可能会有其他进程继续向源系统中已迁移的进程发来消息或信号，这时应如何处理？一种可行的方法是在源系统中提供一种机制，用于暂时保存这类信息，还需要保存被迁移进程所在目标系统的新地址，当被迁移进程已在目标系统中被建成新进程后，源系统便可将接收到的相关信息转发至目标系统。

3. 进程迁移举例

IBM 公司的 AIX 是一种分布式 UNIX 操作系统，它提供一种实用的进程迁移机制。进程迁移的步骤如下：

(1) 当进程决定迁移自身时，先选择一个目标机，发送一条远程执行任务的消息，此消息传送进程映像及打开文件的部分信息；

进程迁移直到虚拟化技术发展之后，才逐步实用化，例如，vm的迁移。

当前，网络计算的主要几个潮流是：

1. 数据中心和跨数据中心的L2/L3 Non Blocking交换。特别是层3的Overlay。

2. SDN：软件定义网络。通过集中的控制，指导网络，特别是策略的下放，路由的收敛。

分布式系统方面，几个主要的动向：

1. OpenStack为主流的云控制和管理平台。

2. 分布式文件系统，例如Hadoop／HDFS和相应的变种／优化产品。

3. 分布式存储服务。例如，通过各个单机节点的局部存储，通过软件虚拟一个整体的巨大的存储系统。

8.6 本章小结

计算机网络是计算机与通信技术相结合的产物，随着微型计算机性能的不断提高和网络技术的飞速发展，很容易通过高速网络把许多计算机连接起来构成一个大型计算机系统。仅管理单台计算机的操作系统称为集中式操作系统，而不仅管理自身所在机器还具有联网通信及相关功能的操作系统称为网络或分布式操作系统。

网络操作系统是基于松散耦合的计算机上的松散耦合软件，配置网络操作系统是为了管理网络中的共享资源，实现网络用户的通信和方便用户对网络的使用，可把它看做网络用户和网络系统之间的接口。计算机网络系统中除了硬件，最重要的是要配置网络操作系统，它除了具备通常集中式操作系统所应有的基本功能外，还应增加联网功能，支持网络体系结构和各种网络通信

[注释] ：分布式系统：1. 不存在一个全局的时钟。2:任何一个时刻不存在一个节点，能知道那个时刻的全局系统状态。原因是：消息传递有时延。Lamport (2013年图灵奖) 逻辑时钟算法精髓是：通过收发消息这天然的同步点，把无序的分布式事件，映射到整数集合上，从而自然排序。