7交互的协调 设备管理

一.设备的概念与分类

1.1设备的分类

软件——运行，组织，管理，维护机电设备和物理机制的程序，系统软件，用户软件

硬件——

回顾部分：输入输出设备：

外部设备——输入输出设备，可以实现人机交互和机间通信等功能。

外设接口——包括三个寄存器，数据寄存器，状态寄存器，命令寄存器

信息交换——四种方式：直接交换，查询（轮询），中断，成组传送

编制方式：独立编址/统一编址

按用途分类

1. 人机交互设备——用于人和机器之间交互的设备
2. 协处理设备——用于给计算机追加额外计算能力的设备。
3. 通信设备——用于给计算机追加通信能力的设备，有可以分为联机设备和转接设备两个分类
4. 存储设备——用于给计算机追加存储器（一般为外存）的设备

按速度分类

1. 高速设备——数据传输速率高于或接近CPU处理能力的设备
2. 低速设备——数据传输速率低于CPU处理能力的设备

为什么数据传输速率的高低是以CPU处理能力来判别的?

因为CPU是整个计算机的核心，所有的数据都很可能要通过它。 如果数据来得比CPU处理得快，意味着设备要等待CPU；否则， CPU就要等待设备。这两种方案下，为获得最优综合性能，程序的设计方法是不同的

按数据传输单位分类

1. 块设备——以固定大小的巨型数据块为单位进行数据传输和处理设备，一般是外存，其IO需要被缓冲
2. 字符设备——以单个或数个字节为单位进行数据传输和处理的设备，一般是鼠标，键盘，显示器，打印机，IO不需要被缓冲，其读写灵活性类似内存
3. 其他设备——不能简单归入块设备或字符设备的设备。这些设备一般是网卡、 数据采集卡等， 其数据一般以存在时间先后顺序的数据包的形 式出现，其I/O需要非常复杂的缓冲方式。

按可共享性分类

1. 共享设备——可以被多个任务同时占用的设备。设备将以很小的等待间隔并 发响应多个请求，或以真正并行的方式响应多个请求。
2. 假脱机设备——多个任务无法真正同时占用该设备，但由于该设备具备请求队列，因此可将多个任务的请求并发提交到该队列，设备将以较长的等待间隔依次响应这些请求。

假脱机——为了缓和CPU的高速性与I/O设备低速性之间的矛盾，引入了**脱机输入、脱机输出**技术。该技术是利用**专门的外围控制机**，先将低速I/O设备上的数据传送到高速磁盘上，或者相反。

比如，打印机

3. 独占设备——一般情况下仅能被一个任务占用的设备。切换这些设备对应的 任务需要用户或应用程序主动干预。

按数据传输方式分类

1. 直接传输设备——随时准别好读写，且响应迅速的设备
2. 轮询传输设备
3. 中断传输设备
4. 直接内存访问（DMA）传输设备——数据量大或 IO操作频繁

按真实存在性分类

1. 物理设备——实际上存在的、有物理实体的设备。
2. 虚拟设备——仅在概念上存在的设备。它有着和真实设备一样的接口，并会 像真实设备那样对操作做出反应，但找不到一个物理实体和它 一一对应。又叫逻辑设备。

设备	按用途分类	按传输单位分类	按传输方式分类	按可共享性分类
显卡	协处理设备 人机交互设备	字符设备	DMA传输	共享设备
显示器	人机交互设备	字符设备	DMA传输	共享设备
键盘	人机交互设备	字符设备	中断传输	独占设备
鼠标	人机交互设备	字符设备	中断传输	独占设备
打印机	人机交互设备	字符设备	DMA传输	假脱机设备
转接器	通信设备	字符设备	DMA传输	共享设备
网卡	通信设备	其它设备	DMA传输	共享设备
硬盘	存储设备	块设备	DMA传输	共享设备
声卡	协处理设备 人机交互设备	字符设备	DMA传输	独占设备
实时时钟	?	?	?	?

1.2例子：字符设备

鼠标

性能目标 鼠标对输入的响应必须迅速，这样使用才有好的体验。

数据的内容 鼠标传输的数据至少包括鼠标的相对位移，以及按键的状态。

数据量的大小 数据量应该很小，就是一个坐标和几个按键的当前状态。

数据传输方向 鼠标一般是单纯的输入设备，因此数据是单向流动到计算机的。

数据传输方式 鼠标的数据量很小，且仅在操作时产生数据；另一方面，鼠标 对操作的响应必须迅速，因此使用中断传输较好。

键盘

键的类型 字符数字型，如A-Z、0-9等，每个按键对应一个字符。

扩展功能键，如F1-F12、Page Up等，每个按键产生一个动作。

组合控制键，如Shift、Ctrl、Alt等 ，改变其它按键的含义。

扫描码 键盘上的每个键都有一个单字节扫描码，据扫描码可确定操作 的是哪个键、是按下键还是释放键。扫描码的低7位是扫描码的 数字编码, 与键盘上的键一一对应；最高位表示键的操作状态， 当按下键时为0；当释放键时为1。扫描码是一个计算机系统内部 编码，不是该键对应的ASCII码！

键盘接口原理 键盘接口对按下键和释放键均向计算机发出中断申请，如果中 断响应条件满足, CPU转去执行键盘中断服务程序。

中断工作流程

（1）从键盘接口读取操作键的扫描码

（2）将扫描码转换成字符码；

大部分键的字符码为ASCII码。

无ASCII码键（如组合键Shift、Ctrl等）的字符码为0。

还有一些非ASCII码键产生一个指定的动作。

（3）将键的扫描码、字符码存放在键盘对应的字符设备中。

实时时钟

实时时钟的功能：读写时间，设置读闹钟

实时时钟的时间基准——晶体振荡器

定时器：带可编程计数上限的计数器。一组D触发器负责计数，另一组触发器则储存一个上限值。一个比较器负责比较这两组值，一 旦计数值达到上限值（**溢出**），就将计数值重置为0。

定时器中断：一旦定时器的输入时钟频率已知，那么其溢出的时间间隔（时 间片）也就决定了。我们可以设置定时器在每次溢出时**产生一 个中断报告给操作系统**，操作系统就能够**强制定期启动调度器。**

常见的系统定时器溢出周期在100μs-10ms之间

实时时钟：一个电子万年历，可以记录时间和日期，并具备闹钟功能。其往往还有一个单独的后备电池供电，用来在计算机主电源不通电时保持时间走时。

其内部是一系列互相串联的定时器。秒定时器的溢出连接到分 定时器的计数时钟输入端，如此继续下去，直到年定时器为止

实时时钟中断：实时时钟可以设置闹钟，一旦闹钟时间点到，便允许向操作系 统发送一个中断。该中断除了做日常事件提醒外还可以触发主板定时开关机。

上电流程

系统启动时，**从实时时钟读取当前时间**，然后系统便**自行维护时间的走动。**时间在POSIX操作系统中是用自1970年1月1日0时以来经过的秒数表达的，计算机每秒将这个变量递增一次，然后 再通过内置的软件万年历将其转化为日期和时间。

系统中的一切行为都以系统时间为基准。

问题一 随着系统的运行，系统时间和实时时钟时间产生差异怎么办?

以实时时钟为准，定期从实时时钟读取时间并覆盖当前系统时 间；也可以测量系统时间与实时时钟的流速差值，并对系统时 间进行校准。

问题二 接上问，若实时时钟时间与实际时间也不一致怎么办?

使用NTP等协议联网校准时间，更新系统时间与实时时钟时间。 现代操作系统在联网时往往会自行执行这一操作。

问题三 如果一个程序想延时一段时间t后再触发某个操作，怎么办?

解决方案一 允许直接编程系统定时器，修改其计数上限值。

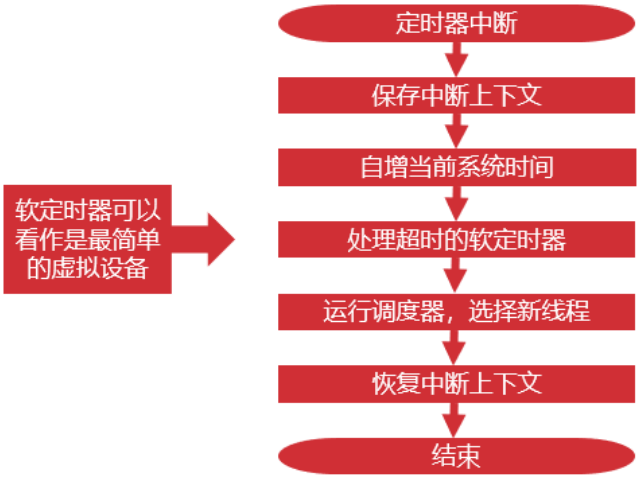
问题四 允许程序修改系统定时器导致其可以修改时间片长度，会破坏 线程之间的时间隔离。此外，如果有两个程序同时使用延时， 就没法用一个定时器应付了。

解决方案二 允许直接编程实时时钟的闹钟，并在闹铃时产生中断。

问题 实时时钟中断的分辨率很低，是秒级别的。此外，即便实时时 钟可以有多个闹钟槽位，但这些槽位的数量终究是有限的（一 般十个以内）。一旦使用定时功能的程序多起来就没法处理了。

能否找到一种方法，用一个定时器模拟出一批定时器?

软定时器　用系统定时器作为时基来模拟一系列的软件定时器。当程序需 要定时时间t时，操作系统内核将记录经过的系统定时器嘀嗒数， 待经过t个嘀嗒时就完成软定时器的定时。



问题 用每次时钟中断都必须检查哪些软定时器超时，这是非常频繁的操作。系统中的程序也可能频繁建立和删除定时器。使用什么数据结构来保证增删改查的性能?

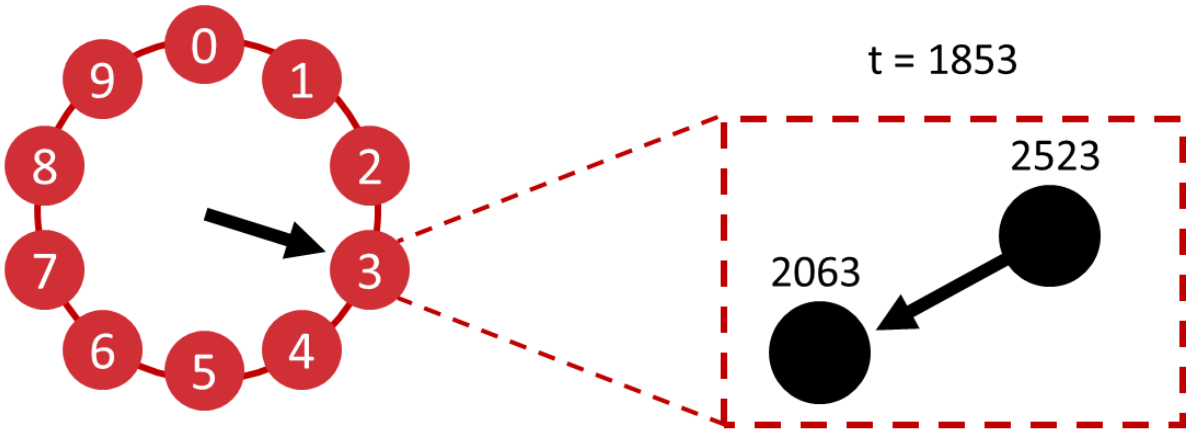
解决方案 和CFS调度器的选择一样，可以使用**红黑树**保存软定时器的超时时点。

由于每次到时间的只可能是距离当前时点最近的那个定时器， 因此只要查询红黑树最左侧的叶子结点即可确定有无软定时器 超时。如果最左侧的叶子结点对应的软定时器都未超时，则不需要查询其它结点。

问题 虽然红黑树的增删改查都保证是O(logn)，但n很多时候还是显得 太大，logn时间也很久。如何能让这个时间小一些?

定时器轮 将定时器按照其超时时间除以某固定值N的余数分成多组。每次检查定时器超时时，只要检查可能超时的那一组就可以了。

定时器轮还可以分成多层，层数越多、轮子越大，检查的时间 越短。因此，软定时器的超时处理可以非常迅速。



问题 软定时器的最高精度是多少？要怎么提高它？

用系统定时器作为时基，

软定时器的工作模式

软定时器有一次性（One-shot）和周期性（Periodic）两种工作模式。一 次性定时器到期后，系统会删除该定时器；周期性定时器到期后，系统 将依据其周期重新计算下次超时时间，并将定时器重新插入数据结构。

高精度定时器（HRTIMER）

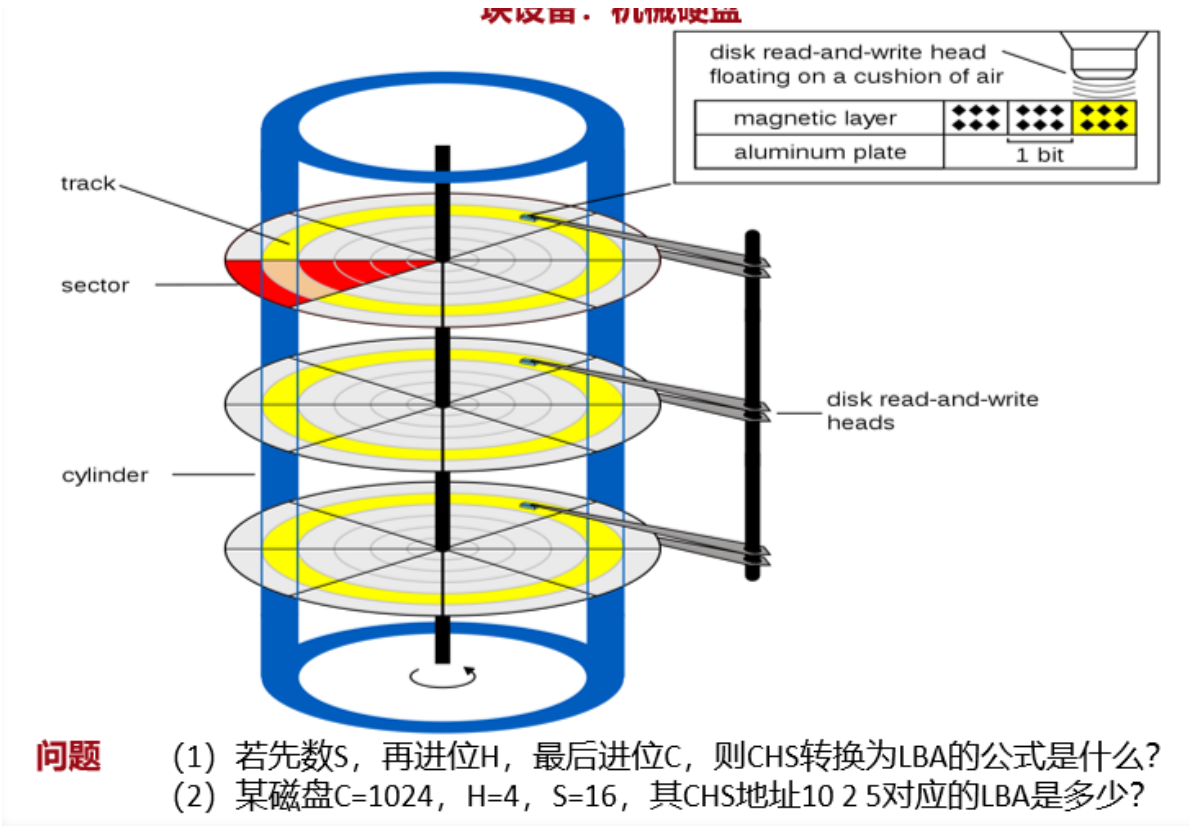
常规定时器是以系统定时器作为时基的，因此其最高精度就是一个时间片。在Linux中，常规定时器又称为低精度定时器，与jiffies变量相关，而 jiffies记录的则是内核启动后经过的时间片数量。

对网络通信、视频播放、数据采集等场合，这种精度是不够用的，因此 引入了高精度定时器HRTIMER。HRTIMER也是软定时器，但**它的底层硬件是具备高精度一次性倒计时能力。**每次内核处理完一个HRTIMER后，都会**将定时器硬件重编程为下一个软定时器的到期时间**，从而完成不依赖周期性中断的计时。

HRTIMER机制一旦启动，低精度定时器机制就不再使用。但内核中的 jiffies变量因为在很多地方用到，仍然需要维持。对此，内核的做法是， 声明一个周期性的HRTIMER，其到期时间正好是一个时间片，模拟原有的低精度定时器中断。

1.3例子：块设备

机械硬盘



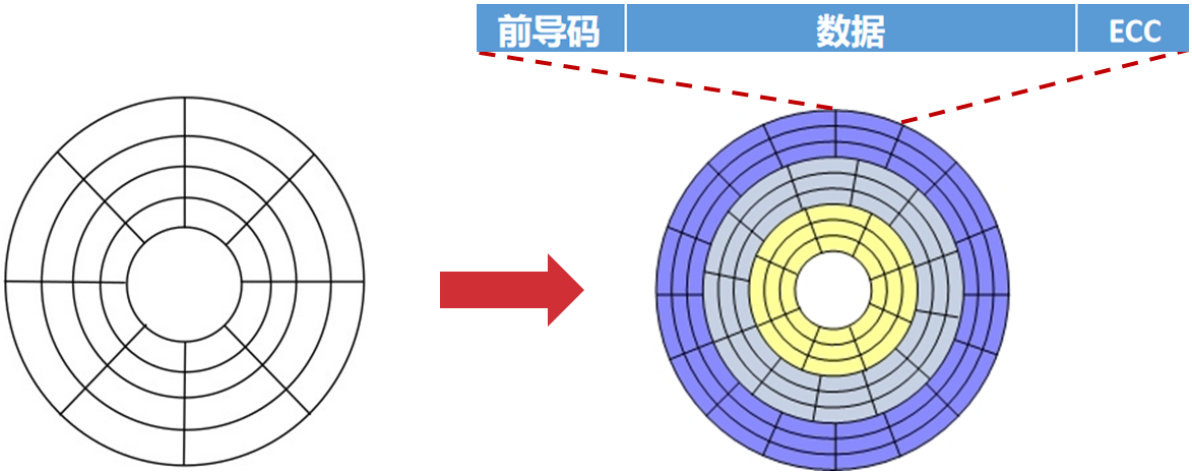
LBA与C/H/S 之间的转换:

设NS为每磁道扇区数，NH为磁头数，C、H、S分别表示磁盘的柱面、磁头和扇区编号，LBA表示逻辑扇区号，div为整除计算，mc为求余计算，则:

LBA=NH×NS×C+NS×H+S-1;

硬盘的接口 机械硬盘等外存的接口是文件系统，提供按文件名存取文件内容的功能。对于机械硬盘而言，这将会转化成为对具体的CHS地址的寻址，以最终定位到某个扇区

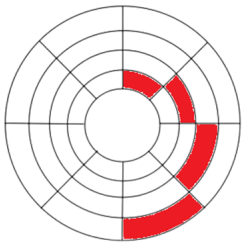
物理规格 硬盘制造时的物理规格，包括盘片的数量、盘片的磁道与扇区 划分、磁头的数量、盘片的旋转速度等。



几何规格 CHS表示法规定，每个盘片（Platter）上的每一圈都有相同数量 的磁道。在硬盘密度较小时，这不是一个问题，但在现代硬盘 中就会造成外圈磁介质的浪费。因此，现代硬盘的CHS表示法没有物理意义，仅仅是虚拟的编址；改用LBA表示法更合适。

低级格式化（低格） 将每个好扇区赋予LBA地址，并写入前导码和ECC纠错码等。还 需要屏蔽坏扇区。完成后，磁盘从物理介质变成一个可用设备。

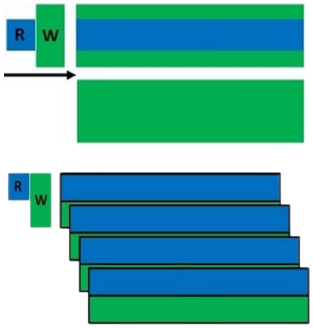
盘片斜进量 磁头本身只能做径向移动，要读取某个扇区只能等待它转过来。为了让读取完 一个磁道后读取另一个磁道的速度加快，每个磁道的1号扇区的起始角度是错开的。这样，上一圈转完读取完磁道，磁头稍作移动就正好等到下一个磁道的开始位置。



传统（Conventional）磁记录 vs 叠瓦（Shingled）磁记录

为保证读写可靠，传统磁盘的设计中写头总是比读头宽。这会导致盘片面积浪费，降低存储容量。为解决此问题，可将多个磁道叠加起来。使磁道之间的排列更紧密。

但这样并非没有代价：写一个磁道会覆盖下面的磁道。因此，每次写磁道都需要把会被覆盖的磁道读出来，然后再一并写回去。



读写单位 虽然磁盘的设备块是扇区，但操作系统并不会按照扇区来读写 硬盘，因为这个单位太小了，效率不高。操作系统实际使用的 读写单位是**簇（Cluster）**，它是一系列连续扇区的集合。

磁盘的共享 磁盘是一个共享设备，这意味着很多进程都会同时提交大量的写盘请求。磁盘的工作效率显著地比CPU低，那么如何处理这些 请求使I/O效率最高?

提示 磁道与磁道之间；簇与簇之间；簇内部

磁道 越靠近的磁道之间的访问延迟更低（磁头摆动幅度小；ms量 级），同时从内圈往外圈读访问延迟比从外圈往内圈读低（盘片斜进量，μs级）。

簇 同一个磁道，从前往后读快，从后往前读慢（扇区的排列顺序； μs级）。一个簇内部的写则最好一次写完。

主要方面 在调度磁盘I/O时，主要考虑磁头在磁道上的摆动耗时，因为这 个时间是以ms计算的，远远超过其它耗时。

排序策略：先来先服务（FCFS）

直接按请求的发起顺序来发起磁盘访问。

优点 简单。无需做任何顺序调整，直接提交请求即可。

缺点 磁盘的寻道时间可能过长。

排序策略：线性扫描

从磁盘头扫描到磁盘尾，在一次扫描中响应所有的请求。

优点 理论上生成的就是最优扫描顺序，磁头只在每个磁道处停留一次。

缺点 这是一个离线算法。已知所有请求时，才能将它们排序，然后一次性从 前向后满足它们。磁盘请求是应用程序运行过程中动态产生的，因此其次序不好预测（前面讲过类似的情况）。

排序策略：最短寻道时间优先（Shortest Seek Time First，SSTF）

在访问时先计算下次磁头移动的距离，选择最近的磁道进行访问。

优点 一定程度上减小了寻道时间。

缺点 有可能造成饥饿现象。什么情况会饥饿？怎么改进？

排序策略：电梯扫描算法（SCAN）

将队列中的请求排序，从头扫描到尾，然后再从头开始。在扫描到底之前，不做回溯，直到扫描完成后才一次性拉回来，像电梯一样。

优点 一定程度上减小了寻道时间，但不产生饥饿（为什么？）。

缓冲区buffer 缓冲区是一个临时的存储区，用以缓和通信双方I/O速度和数据传输单位上的差异，其基本特点是（可含有一定次序重排的） 先进先出队列。

磁盘返回的读数据都被提交到读数据缓冲区，等待操作系统拿走这些数据；提交到磁盘的读写请求都要在操作缓冲区中进行排队，等到合适的时候再操作磁盘。

在操作缓冲区中，同一个簇的读写请求会被合并，同一磁道的不同簇之间的读写请求会被按照簇的次序来排序。不同磁道之间的请求则按照某种策略排序。

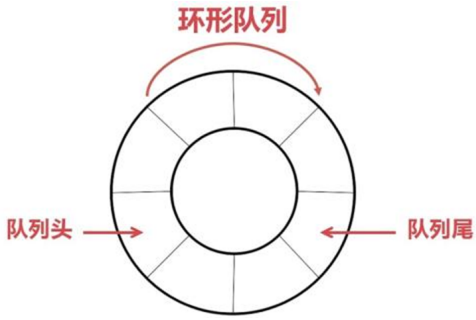
缓存cache 缓冲区的改进，增加了数据存留的功能，不再具备先进先出的特点。现代操作系统的磁盘缓冲实际上都是缓存，它们就像CPU 的缓存那样工作，含有一系列簇的副本；如果操作请求命中它们，就可以免去真正的磁盘读写。这种用来替代缓冲区的缓存习惯上叫做缓冲缓存（Buffer-Cache）。

缓冲区的类型

单缓冲 含有一个请求的缓冲区。

双缓冲 含有两个请求的缓冲区。并发能力更好。

环形多缓冲 队列中的多个请求组成一个环形，并有头尾两个指针。头指针负责写，尾指针负责读，如果头赶上尾则说明队列已满，如果尾赶上头则说 明队列已空。



缓冲池

多个缓冲区的集合，由内核各模块共享。程序需要的时候需要申请，用完释放回去，提供统一的接口并且集中管理。

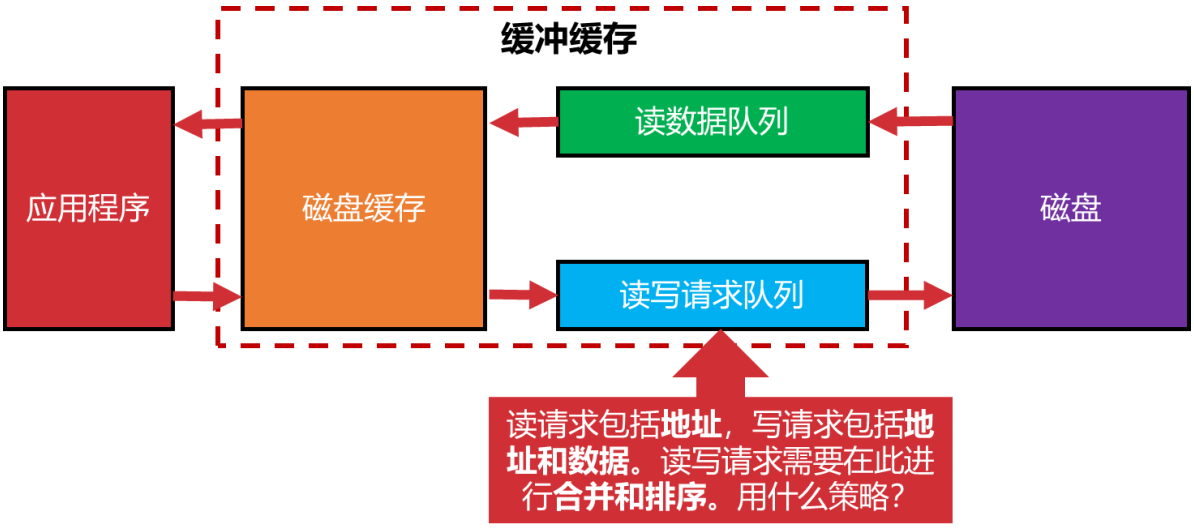
缓存的写策略

写透（Write-Through，WT）

对缓存发起的写会立刻被反映到存储器上。

写回（Write-Back，WB）

对缓存发起的写不会反映到存储器上，而是等待对应的缓存块 被淘汰后才将该块内容写回磁盘。



1.4例子：其他设备

网卡

问题 操作系统中，网卡（Network Interface Card，NIC）的接口应该是什么样?

从以下几个角度思考：

（1）期望达到的性能目的——网卡用来上网，它有两方面要求：一方面需要带宽大，另一方 面又需要延迟低。

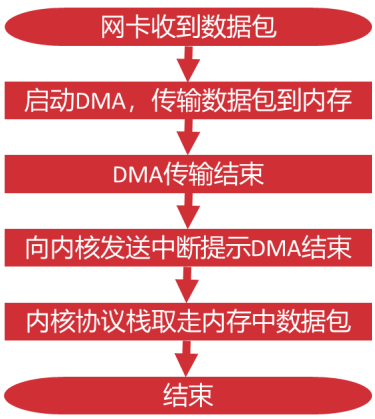
（2）数据的内容——网卡传输的是一系列网络数据包。数据包有一个最大大小 （Message Transfer Unit，MTU），约在1500字节左右（现代网 卡可允许更大的数据包），但却没有最小大小。

（3）数据量的大小——巨大的数据量。甚至比硬盘的数据量都还来得远要大，很多时 候是网卡在等待CPU，而非CPU在等待网卡

（4）数据传输的方向——双向传输，具备一个发送队列和一个接收队列。

（5）数据传输的方式——中断传输或DMA，DMA是主要数据搬运方式，中断仅做通知。

DMA工作流程



中断分发——在多喝处理器中，中断控制器库将设备的中断平均分配给每个CPU，这样就有足够的CPU来处理应付高中断的设备

中断节流——限制设备产生的中断率，也即在一定时间内产生的中断上限数目。对于网卡而言，我们可以将每个包产生一个中断改为每一组包产生一次中断，这样就降低中断对CPU的占用率，也叫中断裁决。

带超时时间的中断节流——报的数量达到一定值，或者第一个包到来后经过延迟时间，即产生一个中断。这样，在包多的时候可以发挥中断节流的优势，有不至于在包少的时候增加过多的延时。

网络协议栈——专门用来处理网卡数据流的协议栈，在网络中，每个包的延迟和可靠传输都是不确定的，因此在数据链路层和网络层之外还需要额外的传输层协议，他们要负责报的重排序，以及丢包重传，最终将一系列零散的数据包组成可以用的数据流送给各个应用程序。

带多收发队列的中断分发——每个CPU使用独立的收发队列，并且网卡通过IP封包的四元组（源、源端口、目的、目的端口）将其分发到不同的队列。这样一来解决了队列争用问题，使无锁成为可能；二来不同传输 层链接的包会去不同的队列中，方便CPU就地处理。此功能还可以结合NUMA，将属于某CPU的队列放在该CPU直接连接的内存中，最大限度地降低处理延迟。

二.设备与驱动程序

2.1驱动程序的概念

定义：直接控制设备的接口程序。

通过直接读写控制器中的寄存器来操作设备

一般运行在内核态，是内核的一部分

驱动程序是设备依赖的

驱动程序的目标是将设备的特性抽象掉，保留设备的共性，以保持内核的其他部分以及用用程序的设备独立的。

驱动程序的功能:
查找设备

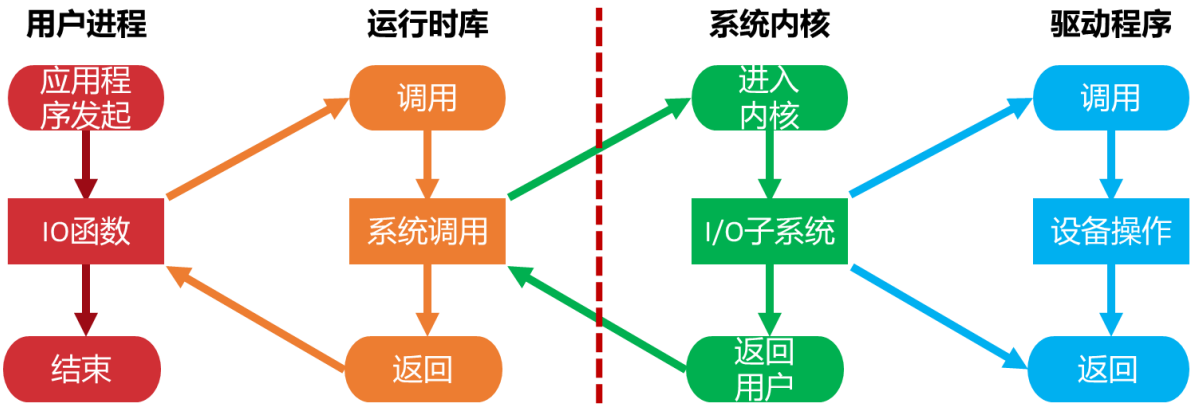
初始化设备

响应设备请求

响应软件请求

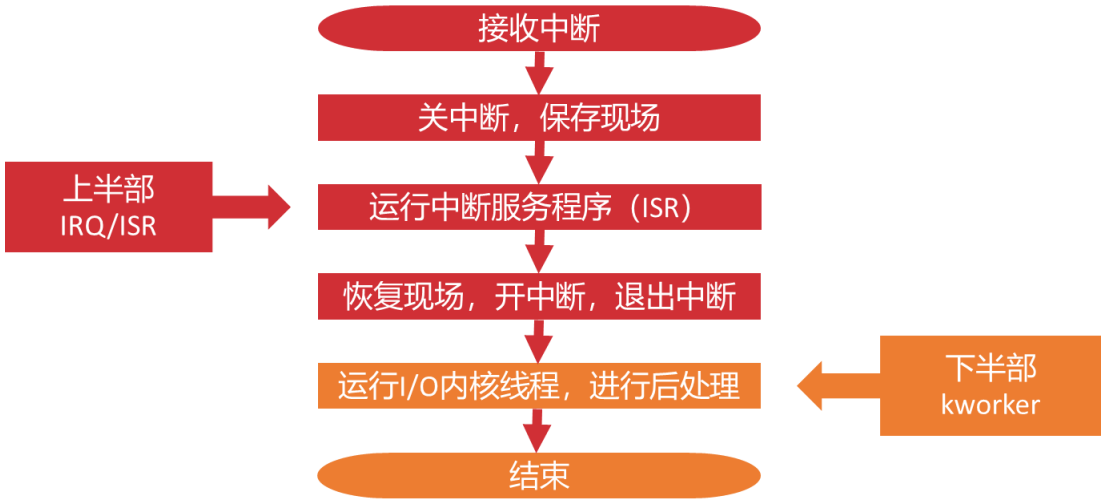
Linux70%都是驱动

设备操作的全景



阻塞：如果设备一时间无法立刻完成操作，就要阻塞发起操作的应用程序线程，直到设备完成操作，发起中断，才能解除该阻塞。

中断响应全景



上半部分：运行在中断上下文

下半部分：一个独立的内核线程

好处：如果对时延敏感，或不能被打断，放在上半部；其它都放在下半部。

驱动程序的基本组成

1. 初始化例程：初始化驱动程序自身
2. 设备识别例程：负责查找，匹配和初始化设备
3. 请求分发例程：将内核IO子系统的请求变化后发送个设备
4. 中断服务例程：响应中断请求
5. 下半部例程：处理中断（但是并非很要紧）

Linux驱动程序：*.ko内核模块

*.ko内核模块：可以在内核运行时被加载或卸除，使宏内核获得微内核那样的灵活性

内核模块运行在内核空间，内核模块之间可以互相访问，任一内核模块故障都会导致死机

编译：Linux的驱动都作为内核模块存在，M成为独立内核模块，Y编入内核

加载：insmod rmmod动态装载和卸除内核模块（sudo权限）

查询：lsmod

windows驱动程序：*.sys驱动文件

2.2驱动程序栈：USB

Unibversal Serial Bus通用串行总线

解决外设连接难，接口标准多的问题

硬件框架：USB设备——（USB桥片）——>PCIe总线——（南桥）——>CPU

热插拔

端口供电

便携接口

数据传输：基于数据包，主机端发起的（只有主机发起问询，设备端才能恢复数据，设备无法主动发起数据给主机）

四种传输方式：

类别	英文	描述
控制传输	Control Transfer	任何USB设备必须支持的传输方式，主要用来初始化和配置设备。
中断传输	Interrupt Transfer	适用于那些必须按照严格时点周期性地可靠传送小批量数据的设备，如鼠标。
等时传输	Isochronous Transfer	适用于那些必须按照严格时点周期性地不可靠传送大批量数据的设备，如摄像头。
批量传输	Bulk Transfer	适用于那些需要可靠传输大批量数据的设备，如U盘和移动硬盘。

“中断传输” 中断传输在USB协议层面并不是基于中断的，而是基于轮询实现。 当设备选择了中断传输时，主机会保证留出总线带宽，严格按 照指定的周期（如1ms）去轮询设备

描述符：

1. 设备描述符，主机会定期扫描 USB端口，一旦它发现有设备被插入（信号线电平变化），就发出控制 传输请求，要求读取设备的设备描述符。
2. 配置描述符：接下来，主机将请求配置描述符，查看设备 的可用工作配置，包括供电要求、功能选项等等。当每个配置描述符被读取时，设备会将该描述符的下属接口描述符和端 点描述符一并发回。USB主机将根据驱动程序和用户程序的输入，选择 一种配置，此后USB设备便工作在此配置下。
3. 接口描述符：描述一个配置中的一部分子功能。比如，一个带显示的手持终端就可以 有两个接口，一个接口负责显示，另一个接口负责按键控制。
4. 端点描述符：描述子功能中包含的一个通信信道（pipe）。比如，键盘可以有两个信 道，一个信道负责向上传送按键通断码，另一个信道则负责下发按键背 光颜色与强度。每个端点都是单方向的：要么是上行端点（IN，设备到主机），要么是 下行端点（OUT，主机到设备）。端点0是一个例外，它固定用来做控制 端点，且每个USB设备都有它；它可以进行双向传输

编写USB设备驱动，就是对设备进行配置，并且对端点进行数据传输。 USB子系统的编写者（或少数几个控制器厂商）已将USB控制器的驱动 写好，因此编写“设备驱动”仅需要调用内核级库函数，非常方便。

USB 驱动程序栈层级

如果要编写USB设备驱动，调动USB控制器驱动就行，不需要学习硬件知识，

完成USB设备 和USB控制器的解耦，成就USB的兼容性

USB设备类通用驱动程序

考虑到很多USB设备的功能都是雷同的，USB-IF准备了很多标准设备类（和子类），这些设备类的配置、接口、端点描述都是模板化的。只要在设备类描述符（XXX Class Descriptor）中提供必要信息，系统就会直接加载通用驱动。这就是所谓的**USB免驱动**；如果厂商生产的设备完全遵照（或部分接口遵照）一个设 备类的描述，那么在用户看来就等于“不需要任何驱动程序”。

考虑到这些驱动程序不属于任何一个厂商，操作系统开发商往 往会内置它们。这就是为何键盘鼠标总是不需要额外安装驱动

三.应用程序与设备

3.1接口的分类

按照接口对应的**设备**来分类

1. 人机交互设备接口
2. 显示设备接口——OpenGL，DirectX等图形学库，或者虚幻，Unity等三维引擎。提供大量的函数库来帮忙绘制界面
3. 网络设备接口——套接字
4. 存储设备接口——文件系统提供的文件操作

按照接口的**阻塞性**分类

1. 阻塞接口——当设备无法完成的时候，在接口上请求的线程将阻塞，直到设备返回数据
2. 非阻塞接口——当设备无法即时完成I/O操作或返回消息时，该接口将立即返回并将当前设备状态报告给调用线程。线程可以以合适的间隔轮询此接口，直到获取到数据。不一定需要操作系统介入。

问题一

（1）为什么需要非阻塞接口？一切接口都阻塞不好吗？

（2）阻塞接口和非阻塞接口哪个效率高？

（3）设备出故障，无法完成操作，导致线程永久阻塞怎么办？

答案 （1）试探是否有数据，或者轮番试探多个设备；（2）数据量 小的设备阻塞效率高；数据量大则反之。（3）增加超时返回。

问题二 如何不使用轮询非阻塞接口的方法，同时等待多个接口？

按照接口**同步性**分类

1. 同步接口——用同一个接口操作来发起I/O请求和接收I/O结果；当接口返回时， I/O结果必定已知，要么完成，要么失败。
2. 异步接口——用一个接口操作来发起I/O请求，并用回调函数来接收I/O结果； 发起请求的I/O接口操作返回时，请求可能还在处理中，I/O结果要等到回调函数被调用时才知道。

   回调函数callback：被操作系统挥着运行时环江调用而非被应用程序性主动调用的用户空间函数，类似中断向量，回调函数越短越好。

   要使用回调函数，需要

   （1）定义该回调函数，

   （2）将回调函数的函数指针和触发它的条件注册给系统，

   （3）系统将在满足条件时调用它，提醒应用程序某事件发生

3.2设备的共享

设备同时联机操作 Simultaneous Peripheral Operations On-Line，Spool

如果该设备同时只能执行一个程序的操作，但程序不关心也不 等待操作的执行完成才能继续进展，则各程序可以将请求提交 到队列，设备则从队列中依次拿出请求执行。相当于将异步I/O 操作转化成了一个同步I/O操作。这种设备就是之前介绍的假脱 机设备，这种操作也叫做假脱机操作。最常见的是打印机。

守护进程

在实际实现中，对于每个假脱机设备，我们都会启动专用进程

用来管理队列和操作I/O，而其他进程则把这个进程当成虚拟设 备并操作它。这个专用进程随着设备的启动而启动，随着设备 的关闭而关闭，因此叫做守护进程。

问题：如果程序需要独占设备，关心设备的操作何时完成，并需要等待它?

解决：加锁