第一章导论

1.操作系统：在计算机用户和硬件之间起代理作用的程序

目的：提供环境，以便用户能便捷高效的执行程序

目标有：执行用户程序，使解决用户问题更加容易；使计算机系统便于使用；以有效的方式使用计算机硬件

根本目的：执行用户程序并更容易解决用户问题

定义：在计算机上一直运行的程序，通常称为内核，从不同视角来看它既是资源分配器又是控制程序，有效利用硬件并管理用户程序的执行

2.计算机系统结构：硬件-提供基本计算资源；操作系统-控制和协调各种应用程序和用户之间的硬件使用；应用程序-定义使用系统资源解决用户计算问题的方式

计算系统：CPU和设备竞争内存，并发执行

每个设备控制器负责特定的设备类型；每个设备控制器都有一个本地缓冲区；每个设备控制器类型都有一个操作系统设备驱动程序来管理它；CPU将数据从主存移动到本地缓冲区，或从主存移动到本地缓冲区；

I/O从设备到控制器的本地缓冲区；设备控制器通过引起中断来通知CPU它已完成其操作

\*中断：中断的目的是提高计算系统的性能；中断通常通过包含所有服务例程地址的中断向量将控制转移到中断服务例程；陷阱和异常是由错误或用户请求引起的软件生成的中断；中断体系结构必须保存中断指令的地址；

中断分为软件中断和设备中断；操作系统是中断驱动的；

操作系统通过存储寄存器和程序计数器来保持CPU的状态

\*存储结构：

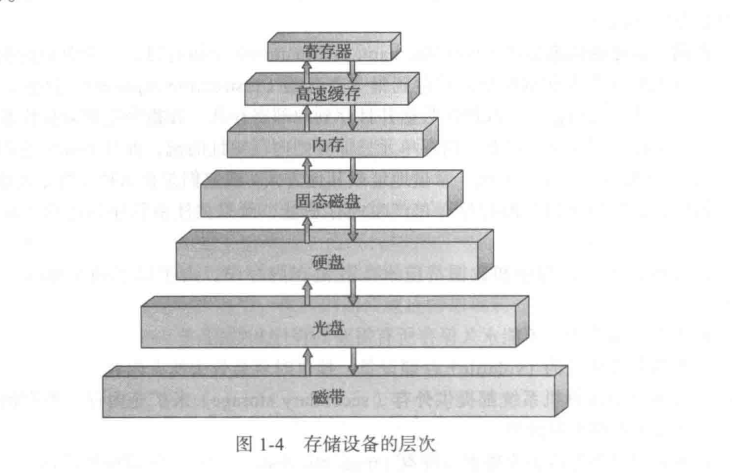
主存储器-只有CPU可以直接访问的大型存储介质：随机存取，典型的挥发性，通常为DRAM；

辅助存储器：主存储器的拓展，提供较大的非易失性存储容量（硬盘）

硬盘驱动器（HDD）：磁盘表面按逻辑划分为磁道，磁道细分为扇区，磁盘控制器确定设备与计算机间的逻辑交互

非易失性存储器设备

缓存-将信息复制到更快的存储系统中；主存可以看作辅存的缓存



\*局部性原理：时间局部性和空间局部性；处理器对指令和数据的内存引用往往会聚集；背处理器访问的内存往往是聚簇的；程序数据被组织成这样的结构，后续访问较低层次储存的百分比尽可能少于访问高层次

\*中断驱动IO缺点：I/O速度取决于处理器测试和服务于设备的速率；处理与I/O传输过程绑定即处理器需要执行一串指令以完成I/O

\*直接内存访问（DMA）：设备控制器可在本地缓存和内存之间传送整块数据，无需CPU干预，每个块生成一次中断；

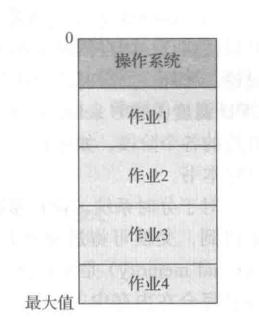
高速缓存：对操作系统不可见，执行一条指令访问一次内存

\*体系结构：单处理器，多处理器：对称多处理器（SMP）：对称处理每个处理器都参与完成操作系统所有任务，非对称处理就是每个处理器都有各自特定的任务，存在主从关系；集群系统：将多个CPU组合在一起，一般网络连接提供高可用性服务

3.操作系统的结构：操作系统最重要的一点是具有多道程序能力。

\*执行过程：引导程序-初始化系统、加载内核的简单代码（位于固件）；内核加载；启动系统守护进程（在内核外提供服务）；内核中断驱动

\*多道程序设计（批处理）：通过安排作业使得CPU总有一个执行作业，从而提高CPU的利用率；系统中所有作业的一个子集保存在内存中，选择一个作业并通过作业调度运行；当作业必须等待时，操作系统将切换到另一个作业；



\*多任务（分时）：批处理系统的逻辑拓展-CPU频繁切换作业，用户可以在作业运行时与作业交互，从而创建交互式计算；

更多特点：

响应时间小于1s；每个用户至少有一个程序在内存中执行过程；如果多个作业就绪同时运行进程调度；如果多个作业就绪同时运行进程调度；如果进程不适合内存，交换会将他们移入和移出内存；虚拟内存允许执行不完全在内存中的进程；

\*多道程序设计（批处理）-》最大化处理器利用率；多任务（分时）-》最短反馈时间

\*进程不等于程序

\*多道和分时联系与区别：

相同：都是微观上串行，宏观上并行的

不同：1进程调度方式不同：多道：只有收到暂停指令才会去运行下一个程序，分时系统中将处理器时间分为很短的时间片，时间片用完就切换，2：目的不同：多道解决人机矛盾即CPu，IO矛盾，提高系统有效性，分时系统是实现人机交互的系统。

\*操作系统本身是一种软件、代码

\*操作系统经常释放控制，必须依赖处理器才能重新获得控制

\*中断处理handler不能睡眠

不能调用schedule（）调度

不能释放信号或调用可能睡眠的操作

不能和用户地址空间交换数据

\*基本的操作系统：批处理操作系统，分时操作系统，实时操作系统

\*多道程序系统：资源利用率高，吞吐量大，平均周转时间周期长，但无交互能力。

\*分时系统：多路性：一台主机可以轮流给多个用户服务。

独立性：每个用户独立终端，独立操作互不干扰，及时性；交互性；对所有事情完全平等

\*实时系统：可靠性对时间要求很高；可用中断和响应；有优先级

\*双模式操作：允许操作系统保护自身和其他系统组件

用户态当中断、异常、系统调用时切换内核态，而当创建新进程，线程、中断、系统调用返回，进程线程上下文切换，信号时切回用户态。

操作系统引导式内核态-操作系统装载后用户态-中断或陷阱-内核态-控制权交还用户前-用户态

\*系统调用：提供用户程序与操作系统之间的接口

\*定时器：防止无限循环的计时器

…………后续将详细由每一章节作内容补充。

第二章 操作系统结构

1.给用户的接口：命令行 系统调用 图形界面

2.系统调用的处理过程：面向程序的编程接口，通过API访问，不直接使用

3.主要结构：简单结构，层次结构：低硬件-高用户，向下调用服务

\*中断、异常：

中断：来自硬件设备的处理请求

异步：产生原因与当前执行指令无关

处理机制：持续，对应用程序透明

异常：非法指令或者其他原因导致当前指令执行失败后的处理请求

同步：产生和当前执行的或试图执行的指令相关

处理机制：杀死或重新执行

系统调用：应用程序主动向操作系统发出服务请求

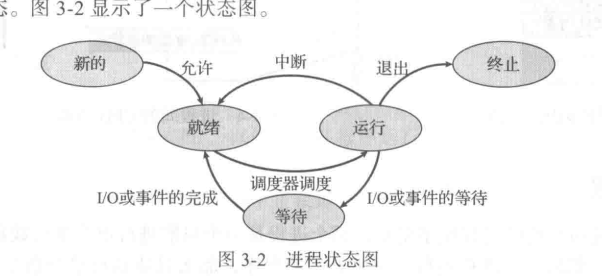
处理机制“等待与持续

第三章 进程

1.进程是一个实例程序的一次执行，它是活动的实体。进程在内存程序在外存，它是系统中的一个工作单元，是操作系统分配资源的最小单位

2.进程元素：程序代码，与代码关联的数据集

3进程的5状态模型：



一次只能有一个进程在一个处理器上运行，但许多进程可处于就绪态或等待态

4.进程控制块：PCB，包含进程相关信息，PCB在内核空间中，读取PCB涉及到用户空间到内核空间的切换。PCB是进程存在的标志

5.进程控制块：包含进程元素；可以中断进程的运行，稍后还可以恢复执行，由操作系统创建和管理；支持多进程并提供多重处理技术的关键工具。

\*进程控制块是操作系统最重要的数据结构；每个进程控制块都包含了操作系统所需进程的全部信息；困难的不是访问而是保护

6.进程创建：进程派生：当操作系统为另一个进程显式地请求创建一个进程时

父进程：创建的原始进程，子进程：新的进程

\*使用fork函数创建进程：

进程在执行过程中，可用通过系统调用来创建多个新进程，从而形成子进程和父进程的关系，进而形成进程树。

执行过程：通过系统调用创建新进程；新进程复制原进程的地址空间，父子两个进程都继续执行，位于系统调用fork后的指令；因为子进程的返回值不同，可用区别父子进程

Fork（）返回值：负值：创建子进程失败；零：返回到新创建的子进程；正值：返回父进程或调用者，包含子进程进程id。

\*exec函数族中只有execve是真正意义上的系统调用，其他都是包装的库函数；

函数族作业是根据指定的文件名找到可执行文件，用其取代进程内容，就是在调用进程内部执行一个可执行文件，其执行成功后不会返回。

7.进程销毁：任何一个计算系统都必须为进程提供表示其完成的方法；一个批处理作业中应该包含一个halt指令或其他操作系统显式服务调用来终止；对交互式应用程序，用户的行为将指出进程何时完成。

8.挂起的进程：交换需求:将内存中的一个进程的一部分或全部移到磁盘上；当内存中没有进程处于就绪状态时，操作系统将阻塞的进程交换到磁盘中，进入挂起队列。

\*挂起进程的特征：进程不能立即执行；进程可能是也可能没有等待一个事件；为阻止该进程执行，通过代理使其处于挂起状态，代理可以是进程本身也可以是父进程或操作系统；除非代理显式地命令系统进行状态轮换，否则无法从挂起状态中被释放。

\*进程控制结构：进程位置：进程必须包含一个或一组被执行的程序，一个进程应该至少有足够的内存空间来保存其程序与数据；程序的执行通常涉及用于跟踪过程调用和过程间参数传递的栈，进程属性：属性合集称为进程控制块，进程映像的位置取决于所采用的内存管理方案。

9.进程的控制：执行模式分为用户模式和系统模式，用户模式：非特权模式，用户程序运行在该模式下；系统模式：特权模式，也称控制模式和内核模式；内核模式指操作系统内核，是操作系统中重要的系统功能。

\*系统中断：

中断：与当前进程无关的某种外部事件相关：时钟中断，I/O中断，内存失效

\*\*时间片：进程中断前可以执行的最大时间段

陷阱：运行中进程产生的错误或异常

10.进程终止：进程执行的最后一条语句，使用exit（）系统调用请求操作系统删除它（将状态数据从子进程返回到父进程用wait（）），进程资源由操作系统释放

父进程可用abort（）系统调用终止子进程执行：当子进程超出分配的资源；不需要分配子进程任务；父进程正在退出（父进程终止子进程不能继续：级联终止所有子女孙辈都终止）；终止由操作系统启动。

父进程可以使用wait（）系统调用等待子进程的终止。该调用返回终止进程的状态信息和pid

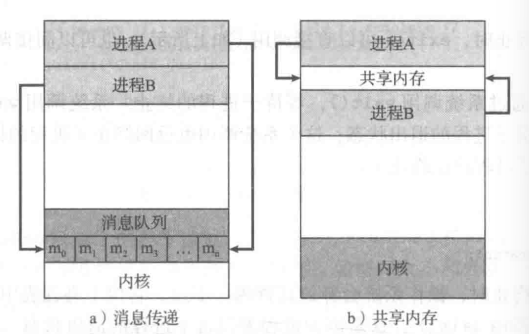
\*子进程终止时若父进程还未调用wait（），那此进程为僵尸

若父进程在未调用wait（）的情况下终止，那么该进程是孤儿进程

11.进程间的通信：系统内的过程可以是独立的也可以是协作的；协作进程可以影响或受其他进程的影响，包括共享数据，合作进程的原因：信息共享；计算加速；模块化；方便用户

协作进程需要进程间通信（IPC）

\*IPC的两种模型：共享内存，消息传递



\*多核处理器中每个核都有独立cache，消息传递优于共享内存，这时没有高速缓存的一致性问题

\*共享内存系统：希望通信的进程之间共享的内存区域，通信双方需要同时映射共享内存到自己的进程内；

通信由用户进程控制而不是由操作系统控制；

主要问题是提供一种机制，允许用户进程在访问共享内存时同步其操作

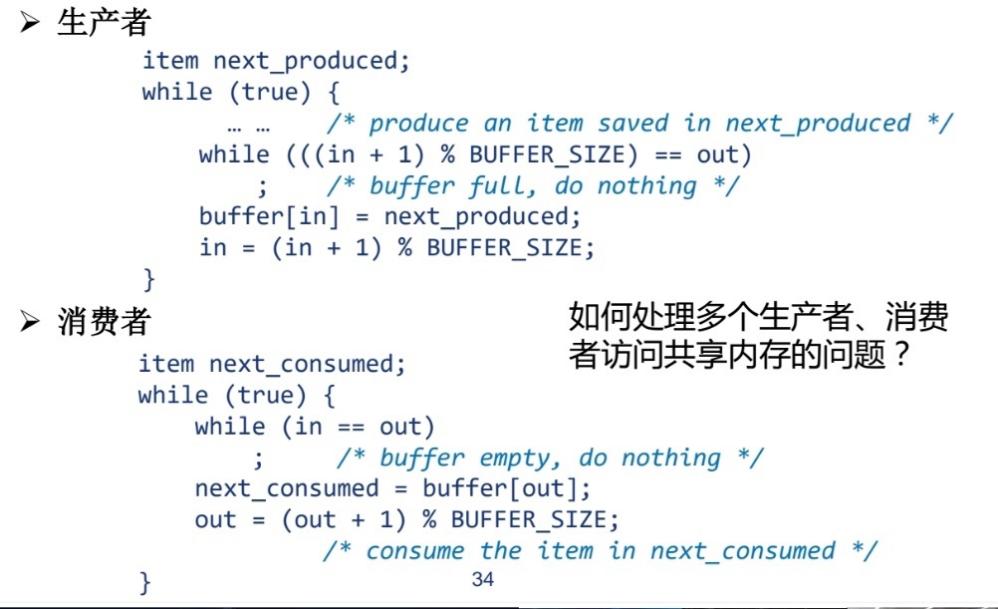
\*生产者消费者问题：（协作进程的范例

变体问题：无边界缓冲区对缓冲区的大小没有实际限制：生产者从不等待，如果没有要使用的缓冲区，消费者将等待

有界缓冲区假定存在固定的缓冲区大小：若所有缓冲区已满，生产者必须等待；若无可用，消费者等待

\*有界缓冲区-共享内存解决方案：

共享buffer的实现采用一个循环数组和两个逻辑指针：in和out。变量in指向缓冲区的下一个空位；变量out指向缓冲区的第一个满位，in==out空，in+1%buffer\_size==out为满。



\*IPC消息传递：进程之间相互通信，无需借助共享变量

操作：发送，接收

若P和Q希望通信：建立通信链接，通过上述操作交换信息。

直接通信：进程间必须明确命名：发送（p，消息）和接收（q，消息）

通信链路特性：自动建立，仅与一对通信进程相关联，每对之间只存在一个链接，链路可能单向但通常双向；

体现了寻址的对称性：发送和接收进程必须指定对方以便通信

间接通信：通过邮箱或者端口来发送和接收消息，

通信链路特性：仅当进程共享公共邮箱时才建立链接，链接可能与许多进程相关联，每对进程可以共享多个通信链路，链路可以是单向的或双向的

存在接收方不明时可以一次只允许一个进程执行接收操作等解决方式

\*\*同步：消息传递可以是阻塞的也可以是非阻塞的

阻塞被认为同步，非阻塞被认为异步

\*缓存：附加到链接的消息队列，不管通信是直接还是间接的，通信进程交换的信息总是驻留在临时队列中；以三种方式之一实现队列：零容量（无缓冲消息系统），有限容量，无限容量（自动缓冲消息系统）

\*套接字为通信的端点，只允许在通信线程间交换无结构字节流，客户机或服务器程序需要自己加上数据结构

12.管道：充当允许两个进程通信的管道

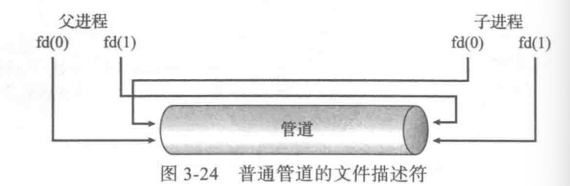
\*普通管道（匿名管道）：无法从创建它的进程外部访问，通常父进程创建管道使用其与子进程通信。

允许标准生产者-消费者风格的通信；生产者写入一端：写入端

使用者从读取端读取数据

单向管道





命名管道：可以在无父子关系时使用；通信为双向，多个进程可使用命名管道通信。

13.进程的上下文切换：一个进程切换到另一个进程运行，包含虚拟内存、栈、全局变量等用户空间的资源，还包括内核堆栈、寄存器等内核空间的资源。

第四章：线程

1. 线程：线程是CPU调度的基本单位，线程只包含运行时的状态，静态部分由进程提供，包括了执行所需的最小状态（主要是寄存器和栈）

\*一个进程可以包含多个线程，每个线程共享统一地址空间，在进程内允许并行

包含线程ID，程序计数器，寄存器和堆栈，与其他线程共享代码段、数据段和其他操作系统资源。

\*一个进程的多线程可以在不同处理器上同时执行：调度的基本单元变为线程，每个线程都有状态，上下文切换的单位变为线程，

每个线程拥有自己的栈，内核中也有为线程准备的内核栈。

\*进程与线程比较：  
\*\*进程的两个特点为：资源所有权，调度/执行：进程具有执行态和优先级，是可被操作系统调度和分派的实体。

概念上有差异：进程是……线程是……

调度上：

并发上：进程并发执行，同一进程内的多个线程可以并行

资源：进程拥有，线程一般没有

系统开销上：进程创建或撤销都需要分配或回收资源系统开销，线程的切换只需要保存和设置少量寄存器内容，不涉及储存器管理

\*多线程是指操作系统在单个进程内支持多个并发执行路径的能力。

\*类比理解进程与线程：我们注意到不同进程之间数据很难共享，进程间不会相互影响，但进程中的一个线程挂掉整个进程挂掉，进程可以拓展到多机，线程最多适合多核

进程使用的内存地址可以上锁，即一个线程使用共享内存时，其他线程必须等待它结束才能使用上锁内存

进程使用的内存地址可以是限定使用量

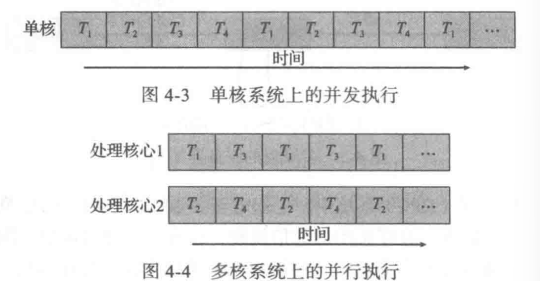
\*多线程的优点：响应性-部分线程阻塞仍可用继续执行

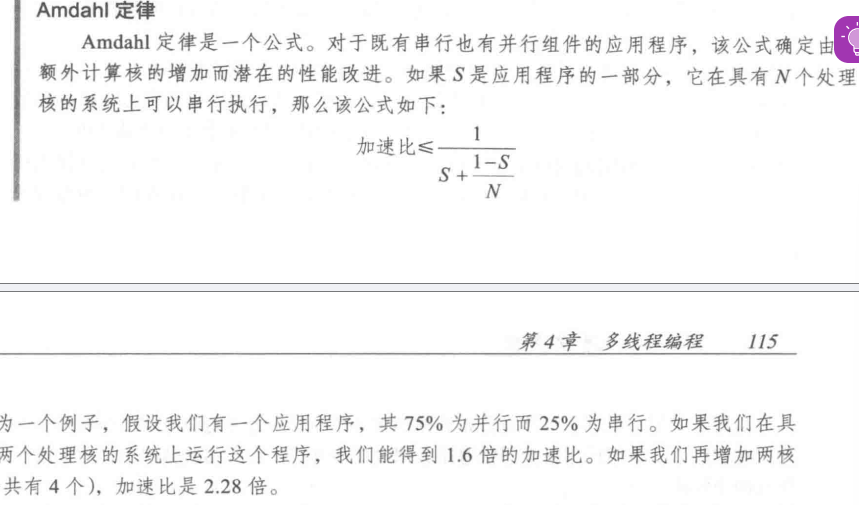
资源共享-允许一个应用程序在同一地址空间内有多个活跃线程；

经济；可拓展性-线程可以利用多核体系结构

多核编程：单核系统并发：快速的CPU调度支持多个任务，提供并行假象

多核系统：并行执行：同时执行多个任务：





N趋于五穷时，收敛于1/S

\*并发支持多个正在进行的任务，允许所有任务都能取得进展（即使是单处理器也有调度器来提供并发性）

\*并行意味着系统可以同时执行多个正在进行的任务

类型：数据并行，任务并行：每个线程执行唯一操作；一般混合使用

2.线程控制块：TCB

\*线程包括了用户空间的数据和资源，内核中的数据结构（TCB）

3.线程分为用户线程和内核线程：

用户级线程：管理线程的所有工作都由应用程序完成；内核意识不到线程的存在，调度单位仍然为进程

\*优势：切换不需要内核态；可以被特定应用程序调度；可以运行在任意系统

\*劣势：执行系统调用时，不仅会阻塞线程，也会阻塞进程中的所有的线程不能利用多处理技术，一个进程中只有一个线程可以执行。

内核线程：线程的管理工作由内核完成，应用级没有线程管理代码，只有一个内核线程设施的应用编程接口（API），其调度单位为线程。

\*优势：内核可以同时把同一个进程中的多个线程调度到多个处理器；进程中的一个线程阻塞，内核可以调度同一个进程中的另一个线程；内核线程自身也可以多线程。

\*劣势：切换线程时需要切换内核模式

混合方法：线程的创建完全在用户空间中完成；线程的调度和同步在应用程序中进行；多个用户线程会映射到一些内核线程上，在组合方法，同一应用程序中的多个线程可以在多个处理器上并行运行，并且阻塞系统调用不需要阻塞整个进程。

4.线程模型即用户线程与内核线程的对应关系：

多对一：线程管理由用户空间的线程库来完成，效率更高。但一个线程执行阻塞系统调用，那么整个进程将会阻塞；任一时间只有一个线程可以访问内核，所以多个线程不能并行运行在多处理核系统上。

一对一：在一个线程执行阻塞系统调用时，能够允许另一个线程继续执行，提供了更好的并发功能；允许多个线程并行运行在多处理器系统上，只是创建线程开销较大

多对多：(常用)开发人员可以创建任意多的用户线程，并且相应内核线程能在多处理器系统上并发执行，当一个线程执行阻塞系统调用时，内核可以调度另一个线程来执行。

隐式多线程：将多线程的创建与管理交给编译器和运行时库来完成

\*方法：线程池；Fork-Join；OpenMp；中央大调度；英特尔线程构建

线程池：在池中创建多个线程，在其中等待工作：使用现有线程处理请求通常比创建新线程快一点；允许将应用程序中的线程数限制到池的大小；将要执行的任务和创建任务的机制分离，可以使用不同的策略来运行任务

Fork—join：将大任务分解为多个小任务，交由多个工作线程运行，最后将小任务的计算结果汇总，得到大任务的结果。

5.多线程问题：

Exec（）通常正常工作-替换正在运行的进程，包括所有线程

\*取消仅在线程到达取消点时发送

第五章 CPU调度

1.CPU调度是多道程序操作系统的基础，通过在进程间切换CPU，操作系统可以更加高效

2.对于支持线程的操作系统，操作系统的实际调度的是内核级线程而非进程

长程调度：将已经进入系统的作业送入内存创建进程并加入准备队列

中程调度：将外存中被挂起的未完成进程调回内存，就绪挂起或阻塞挂起队列

短程调度：进程调度，处理器分配给一个就绪态进程执行，有抢占式与非抢占式。

调度算法：

FCFS：非抢占，对输入敏感，等待时间较长

SJF：最短剩余时间优先！！！只是非抢占式的不明显啦

抢占式的最短作业优先平均等待时间和平均周转时间降低，但是代价是增加了上下文切换的成本

\*抢占式SJF平均等待时间可以最小，短程调度中通常预测下次CPU执行长度，因为看不到

轮转调度：专门为分时系统设计，类似FCFS但增加抢占来切换进程，是抢占式调度的；

每个进程给定一个较小的时间单位作为时间片，时间片用完选择另一个进程调度执行；循环调度就绪队列中的进程

（如果就绪队列中有n个进程，时间片为q，则每个进程一次最多以q个时间单位的块获取1/n的CPU时间，没有进程等待超过n-1个时间单位）；

Q非常大==FCFS

Q非常小==很多上下文切换，开销高

\*轮转调度RR最小化响应时间

\*\*特点：时间片要远大于时间片切换的时间，让上下文切换的时间占时间片的一小部分，随着时间片大小增加，一组进程的平均周转时间不一定会改善

优先级调度：优先级编号与每个进程相关联

问题：饥饿-低优先级进程可能永远不会执行

解决方案：老化-随着时间推移，进程优先级增加

多级队列调度：优先级调度器，前台进程RR；后台进程FCFS

3.线程调度：用户级线程和内核级线程的区别；如果支持线程则调度线程

4.多处理器调度：

非对称多处理器

对称多处理器（SMP）：每个处理器自调度

所有线程位于公共就绪队列a中

每个处理器可能有自己的线程专用队列b

\*多线程多核处理器：如果一个线程发生内核暂停，切换到另一个线程

细粒度：切换线程，线程之间的切换很小；

粗粒度：线程一致在处理器上执行，切换成本高；

两个级别的调度：操作系统调度；指定每个核心如何运行那一个硬件线程

周转时间=等待时间+cpu运行时间

Cpu利用率=cpu运行时间/系统周转时间

5.实时CPU调度

硬实时系统：一个任务应在它的截止期限前完成

软实时系统：有一个相关的截止期限但不是强制性的，不保证会调度关键实时进程，只是保证其优先度高于非关键性进程

最小化延迟：

实时系统的驱动性质；事件发生时，系统应尽快响应和服务它

事件延迟：从事件发生到事件得到服务的这段事件

中断延迟：是从CPU收到中断到中断处理程序开始的时间，对实时系统至关重要的就是尽量减少中断延迟

调度延迟：从停止一个进程到启动另一个进程所需时间量，保持调度延迟尽可能低的技术是，提供抢占式内核

实时调度：支持抢占式、基于优先级的调度

对硬实时需要满足提供截止期限的能力，添加新的调度特征。

进程新的特点：周期性进程需要以固定时间间隔使用CPU

进程应该向调度器公布其截止时间

采用准入控制

6单调速率调度：

抢占式静态优先级，调度周期性任务，按周期的倒数来分配优先级，T短优先级高

错过截止时间的单调速率调度N个进程最坏情况下的CPU利用率n（2^1/n-1）；

最早截止期限优先（EDF）调度：

根据截止期限动态分配优先级，截止期限越早进程优先级越高，不要求进程周期执行，不要求CPU时间固定，进程在编程可用运行时给出截止时间，CPU利用率高：

一些杂七杂八：优先级最高的线程称为下半部内核，优先级不会更改

上半部内核优先级永远不改；实时用户保持开始时的优先级，不会因为使用资源而更改；分时用户会定期重新计算优先级，最低优先级范围称为空闲用户。

7.算法评价：确定性模型是一个分析评估类型，主要用于描述调度算法和提供例子

排队模型：

补充：多对一、多对多使用PCS进程竞争范围调度用户级线程，一对一模型只用SCS系统竞争范围决定调度那一个内核级线程到处理器上。

第六章 同步

1.多进程共享内存问题，即同步问题的根源

2.竞争条件：多个进程并发访问和操作同一数据且执行结果与访问顺序有关；

为了防止竞争条件，确保一次只有一个进程访问关键数据。

竞争条件在操作系统中普遍存在，竞争条件是不确定的

\*同步，直接相互制约：源于进程合作，如果某一进程不收到另一进程提供的必要信息就不能继续下去；对多个相关进程在执行次序上进行协调，使并发执行的进程间可以有效的共享资源和相互合作，使程序执行具有可在线性保持数据一致

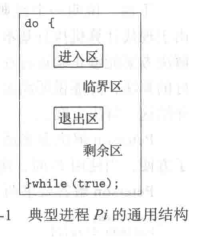
\*互斥，简介相互制约：源于资源共享，进程排外执行。

3.临界区问题：

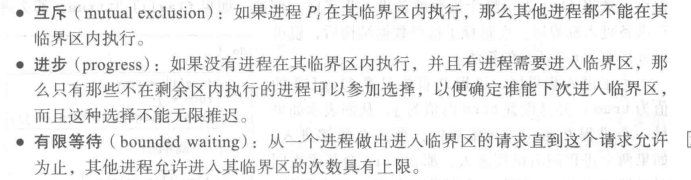
\*临界资源：一个时间段内只允许一个进程使用的资源称为临界资源，多个进程互斥访问。

考虑多进程系统

每个进程都有一段代码称为临界区，是并发进程中与共享资源有关的程序段。临界区中进程互斥。



临界区问题的解决方案要求:



有两种常用方法来处理临界区问题：抢占式内核：允许处于内核模式的进程被抢占，和非抢占式内核任一时间点只有一个进程处于内核模式。

4.硬件同步：

核心是中断禁用，缺点：耗时，系统时钟若依靠中断，会受影响。

特殊机器指令实现原子指令优势：

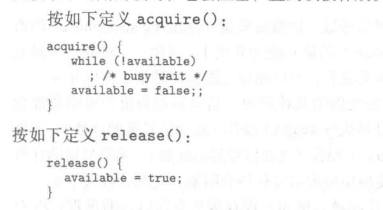
适用于单个处理器或共享内存的多个处理器上的任意数量的进程；简单且易于验证；支持多个临界区，每个临界区由自己的变量定义

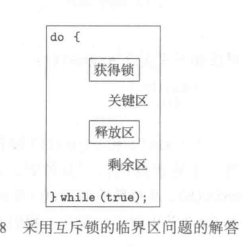
劣势：忙等待；饥饿；死锁；

\*互斥锁：表示锁是否可用的布尔变量

方式：获取一个锁，释放一个锁，其过程必须是原子的，通常使用硬件原子指令实现。又称自旋锁，通常用于多处理器系统。

进程等待锁时没有上下文切换，





\*严格备选：

在两个进程间严格交替（turn=0，process=0；turn=1，process=1）确定

劣势：效率不高

\*Peterson解决方案：

基于软件，适用于两个进程交错执行临界区与剩余区。

Turn和flag[2]全局：

Turn表示pi可以执行flag表示准备进临界区

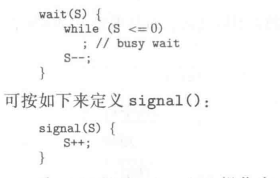


\*信号量：完成某种功能不能被分割，也不能被中断的操作序列。

\*\*互斥锁是一种外部强加的控制方案或者说全局控制，信号量是一种内部主动的控制方案，局部控制。

信号量S-整数变量：

Wait（）和signal（）即P（），V（）



信号量整数值的修改应该不可分割的执行，就是说当一个进程修改信号量值的时候没有其他进程可以同时修改同一信号量的值，wait中s的测试和修改也不能被中断。

\*\*一些重要结论：在进程递减信号量之前无法知道它是否会被阻塞；当两个进程并发运行的时候，无法知道那个进程将在单处理器系统上立即继续；

不知道是否有另一个进程正在等待，因此未阻塞的进程数可能为0或1；

计数信号量中的信号量初值=可用资源数量，进程需要资源-wait，释放资源-signal，信号量计数为0，资源都在使用中，信号量为负-绝对值为等待它的进程数。

\*计数信号量：整数值可以在不受限制的域内变化。

\*二元信号量可以实现互斥锁的功能，不同点在于锁主体不同。

\*每个信号量都有一个整数value和一个进程链表list，当一个进程必须等待信号量时，就被添加进进程链表，singnal从等待进程链表上取走一个进程，并加以唤醒。

\*强弱信号量：强信号量：FIFO，被阻塞时间最长的进程最先从队列释放

弱信号量：没有规定从队列中移出顺序的信号量

解决忙等待：挂起和唤醒。

5.死锁与饥饿：具有等待队列的信号量实现可能出现这样的情况，两个或多个进程无限等待一个事件，该事件只能由这些等待进程之一来产生---进程死锁

无限阻塞或饥饿：进程无限等待信号量

6.管程：一种高级程序设计语言结构，提供功能与信号量相同但易于控制，管程是由一个或多个进程、一个初始化序列和局部数据组成的软件模块（使用信号的管程）

属于抽象数据类型，提供一组由程序员定义的在管程中互斥的操作。管程中一次只能有一个进程处于活跃状态。

定义附加的同步机制，可以有条件结构来提供，对于条件变量，只能调用x.wait（）,x.signal()。

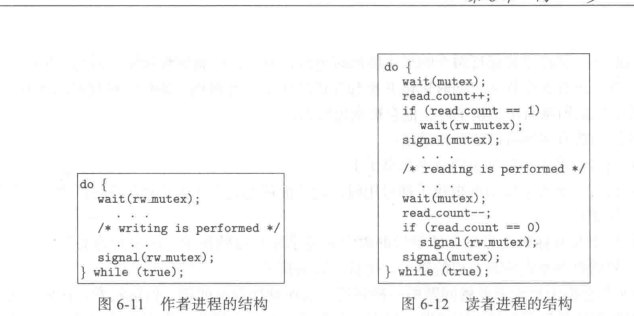
6.经典同步问题：

\*有界缓冲区问题：用信号量empty和full表示空和满的缓冲区数量

\*读者写者问题：

第一读者-写者问题：要求读者不应保持等待，除非作者已获得权限使用共享对象（作者可能饥饿）

读者进程共享：



第二读者写者问题：一旦作者就绪，作者会尽可能的执行（可能读者饥饿。

\*哲学家就餐问题：每只筷子编号用信号量表示

解决死锁办法：

最多允许4人上桌，只有左右筷子可用才可拿起；非对称的解决方案，管程解决方案

\*\*\*事务内存是一个内存读写操作，不涉及锁，所以不存在死锁。

\*在进行作业调度时，想要兼顾作业等待时间和计算时间，应该选取响应比高的算法。

\*系统出现死锁的根本原因：资源管理和进程推进顺序的不得当

第七章 死锁

1.死锁：当一组进程中的每个进程都在等待某个事件（资源）而这组进程中被阻塞的其他进程才可触发该事件时，则认为该组进程发生了死锁

\*死锁是永久性的

\*死锁问题并没有有效而通用的解决方案

\*死锁问题都涉及两个或者多个进程之间对资源需求的冲突。

2.死锁存在的必要条件：互斥、占有并等待：当一个进程等待其他进程时，继续占有已分配的资源、非抢占、循环等待：存在进程链，每个进程至少占有此链中下一进程所需的一个资源。

3.资源分配图：表征进程资源分配的有效工具，看是否有环，死锁必有环

4.处理死锁的方法：死锁预防和死锁避免

\*死锁预防：使死锁的四个必要条件之一无效（设备使用率低和系统吞吐率低）：

互斥-一般不这样作，因为我们必须保留不可共享的资源

占有等待：保证每当进程请求资源时，不会保持任何其他资源：

一种实现：当开始执行之前请求并分配其所有资源，另：仅当进程在未分配任何资源时才允许进程请求资源（资源利用率率低，饥饿）

无抢占：允许抢占

循环等待：破坏循环等待对所有资源进行完全排序，每个进程按递增顺序申请资源

\*一般我们破坏第四个条件：

循环等待：为每个资源分配唯一的编号，按顺序获得资源

5.死锁避免：（需要操作系统得到有关进程的申请资源和使用资源的额外信息）  
\*死锁预防的副作用是设备使用率低，系统吞吐率低；

死锁避免算法动态检查资源分配状态，以确保不存在循环等待条件

资源分配状态由可用和已分配资源的数量以及进程的最大需求定义。

\*安全状态：当进程请求可用资源时，系统必须决定立即分配是否使得系统处于安全状态；

只有存在一个安全序列，系统才处于安全状态，安全序列是指：对于每个进程Pi，Pi仍然可以申请的资源数小于当前可用资源加上所有进程Pj所占有的资源。

\*死锁避免的两种方法：资源分配图算法（单个资源）：添加需求边，环检测算法检查安全性。有环安全，无环非安全。

银行家算法：（多资源）进程已占用资源数与本次申请资源数之和不超过对资源的最大需求，且现有资源尚能满足需要的最大资源量：

\*写一个need序列

\*安全算法：

\*资源请求算法：

6.死锁检测算法：等待图（每种资源单个实例），当且仅当等待图中有一环，系统死锁

死锁避免的优势：无需死锁预防中的抢占和回滚进程，与死锁预防相比限制较少

限制：必须事先声明每一个进程请求的最大资源，进程执行顺序无同步要求限制；资源数量固定，占有资源时，进程不能退出

7.死锁恢复：进程终止：终止所有死锁进程，一次中止一个进程直到消除死锁循环

资源抢占：选择牺牲进程，回滚：回到某个安全状态，也可完全回滚，中止进程重新执行==重启，饥饿

\*\*可抢夺的资源分配策略可预防死锁，但只适用于主存和处理器。

第八章 内存管理策略

\*程序必须（从磁盘）放入内存并放入进程中才能运行。

\*主存储器和处理器内置寄存器是CPU可以直接访问的唯一储存器

存储单元只能看到：地址+读取请求；地址+数据和写入请求。

寄存器访问在一个时钟内完成

内存访问可能会占有许多周期，从而导致暂停（stall）

高速缓存位于主内存和CPU寄存器之间

不仅关心内存的访问速度，还要确保操作的正确。为此需要保护操作系统，而不被用户进程访问。这种保护应通过硬件来实现，操作系统不干预CPU对内存访问

1.内存保护：需要限制进程只能访问其地址空间中的那些地址

可以通过一堆基址寄存器和界限地址寄存器定义进程的可访问地址。（基地址寄存器含有最小的合法物理内存地址，界地址寄存器指定了范围大小）

地址空间提供这种保护，通过CPU硬件实现；

\*硬件地址保护：CPU必须检查在用户模式下生成的每个内存访问，以确保该用户的内存访问量在基地址和界限之间

\*加载基地址寄存器和界限寄存器的指令具有特权性，只能在内核模式下运行，也只有操作系统可以加载基地址寄存器和界限地址寄存器。

\*地址绑定：每一次绑定都将一个地址空间映射到另一个地址空间。

\*指令和数据绑定到内存的地址绑定可以发生在三个不同阶段：

编译时：如果内存位置已知，绝对代码

加载时：编译时内存地址位置：重定位代码，绑定会延迟到加载时才进行，若开始地址变化，那么只需要重新加载用户代码以合并更改的值

执行时间：进程在执行期间可以从一个内存段移动到另一个内存段，则绑定延迟到运行时\*\*需要硬件支持地址映射

2.逻辑地址空间和物理地址空间：逻辑地址空间绑定到另一单独物理地址空间的这一概念是正确内存管理的核心。

\*逻辑地址，即cpu生成的虚拟地址，物理地址，内存单元看到的地址，即加载到内存地址寄存器的地址

\*\*编译时和加载时地址绑定方案生成相同逻辑地址和物理地址；执行时地址绑定生成不同的逻辑地址和物理地址

\*逻辑地址空间是程序生成的所有逻辑地址的集合；物理地址空间是逻辑地址对应的所有物理地址的集合。

2.内存管理单元MMU：

运行时，将虚拟地址映射到物理地址的硬件设备

\*现在基址寄存器称为重定位寄存器

重定位寄存器中的值在发送到内存时加上用户进程生成的每个地址。用于保护用户进程彼此间不受影响，以防更改操作系统代码和数据

用户程序处理逻辑地址，不会看到真正的物理地址。

\*动态加载：

在调用程序前，不加载该程序，优点：只有一个程序被需要时才会被加载，不需要操作系统的特殊支持，只需要提供库来帮助实现动态加载

3.静态链接：加载程序将系统库和程序代码组合到二进制程序映像中，动态链接：对系统库的链接操作延迟到运行时，每个库程序的引用都有一小段代码，也叫存根，存根会用程序地址来替换自己

\*操作系统检查程序是否在进程的内存地址中

4.交换：进程必须在内存中以便执行，但其可以暂时从内存中交换到备份存储或磁盘，交换可能让所有进程的总物理地址空间超过真实物理空间。

\*标准交换：从内存到备份存储，一般是磁盘。一般禁止，只有当空闲时内存低于某个阈值时启用交换，交换一般与虚拟内存一起工作。

\*\*移动系统交换一般禁止任何交换

连续内存分配：

内存同时容纳操作系统和用户进程，内存作为有限资源必须有效配置，操作系统一般放在低内存，连续内存分配时：每个进程都包含在内存的单个连续部分中，与下一个进程的内存相连。

\*固定分区（分区的数量在系统生成阶段就已经确定，因而限制了系统中的活动进程的数量；由于分区是在系统生成阶段事先设置的，因而小作业不能有效利用分区空间）：每个分区只包含一个进程

\*可变分区：所有内存都可用于用户进程，因此可以作为一大块可用内存，称为孔

\*孔：可用内存块

\*操作系统使用表维护以下信息（1分配的分区

（2空闲分区（孔）

5.动态存储分配：从一组可用孔中选一个空闲孔的常用方法：

首次适应：分配首个足够大的孔；从头查找或从上次首次适应结束开始，一旦找到足够大的空闲孔则停下；

最优适应：分配足够大的孔；搜索整个列表除非已经是排好序的，选择最小的剩余孔

最差适应：分配最大的孔，搜索整个列表，产生最大的剩余孔，该孔可能比最优适应产生的较小剩余孔更为适用

\*在速度和存储利用率方面，首次适应和最优适应优于最差适应。首次适应和最优适应在利用空间方面相当，首次适应要更快

\*首次适应会使得前端产生很多小空闲区，有外部碎片

最优适应会剩下小孔

\*外部碎片：存在满足请求的总内存空间，但它不是连续的

50%规则：假定有N个可分配块，那么可能有0.5个块为外部碎片，1/3内存可能无法使用

\*\*解决方式：紧缩：移动内存内容，以便将所有空闲空间合并成一整块，但当重定位是静态的并在汇编和加载时进行就不能紧缩，重定位必须为动态的；另一解决方案：允许进程的逻辑地址空间不连续，基于分段和分页两种互补技术实现

\*内部碎片：分配的内存可能略大于请求的内存；此大小差异是分区内部的内存但是又不可以使用

6.分段：程序和其相关的数据划分到几个段：所有段的长度不一定相等；段有最大长度

\*逻辑地址由段号和偏移量两部分构成

\*分段类似于动态分区

需要将程序的所有分段都装入内存

一个程序占据多个分区但这些分区不要求连续

\*\*分段消除了内部碎片，但会存在外部碎片

\*程序产生二维地址<段号，偏移>映射到一维物理地址，这个地址是通过段表实现；

\*段表包含：段基地址，段界限

\*分段允许进程的物理地址空间是非连续的，避免了内部碎片

7.分页

分页可用避免外部碎片和紧缩，以及将不同大小内存卡匹配到交换空间的问题被大多数操作系统采用，但存在内部碎片

\*基本方法：将物理内存分为固定大小的块：称为帧或页帧，将逻辑地址页分为同样大小的块，称为页，页面，执行一个进程时，它的页从文件系统或备份存储加载到内存的可用帧，设置页表将逻辑地址转换为物理地址

备份存储区（磁盘同样拆分为多个界面）

\*CPU生成的地址：页码：页表的索引：每页基地址

页偏移：与基地址结合生成物理内存地址

\*页大小由硬件决定，若逻辑地址空间为2^m,页大小为2^n字节，那么m-n位代表页码，低n位表示页偏移

\*分页本身是一种动态的重定位

\*分页的重要作用是将程序员视图的内存和实际的物理内存进行清除分离，优点之一是可以共享公共代码

操作系统维护帧表

\*分页会增加上下文切换的时间，因为操作系统为每个进程维护一个页表的副本

\*页表保存在主存中（小的页表可采用寄存器），为每一个进程分配，加载和修改页表的指令是特权的

\*在该方案中，每个数据/指令访问都需要两次内存访问，速度减半

\*两次内存访问问题可用TLB转换表缓冲区（关联内存）来解，决

TLB是键-值存储，多层TLB设计用于性能加速

8.TLB（转化表缓冲区，快表）

地址转换（p，d）：

若p在关联寄存器中，则将帧号取出，否则，从内存页表中获取帧号

9.有效内存访问时间：命中率：在TLB中找到所需页码的次数百分比

有效内存访问时间=命中率\*访问内存时间+（1-命中率）\*2倍内存访问时间

10.内存保护

分页环境下的内存保护是通过与每个帧关联的保护位来实现的，通过将保护位与每个帧关联以指示是否允许只读或读写访问来实现内存保护

\*附加到页表中每个条目的有效-无效位

\*任何违规行为都会导致内核陷阱

共享页：

共享代码，私有代码和数据

11.页表结构：

分层页表

哈希页表：处理大于32位地址空间的常用方法，采用虚拟页码作为哈希值

散列页表：

倒排页表：对于每个真正的内存页或帧，倒置页表才有一个条目，每个条目上包含保存在真正内存位置上的页的虚拟地址。

第九章 虚拟内存管理

1.虚拟内存：将用户逻辑和物理内存分离

允许多个进程共享地址空间

逻辑地址空间可以比物理地址空间大得多

\*虚拟内存定义：

虚拟地址空间：进程如何存储在内存中的逻辑视图

通常从地址0开始连续地址直到空间结束

同时，物理内存组织在页帧中，分配给进程的物理帧可以不连续

MMU必须将逻辑页映射到物理页帧

\*实现方式\*请求分页，按需分段

\*请求调页：

可以在加载时将整个进程放入内存

或请求调页：仅在需要时才加载页面，页面只有在程序执行期间被请求才被加载，

其类似于具有交换的分页系统，不过是采用惰性交换器的哪一种

“调页程序只涉及进程的页面”

2.如果页已经驻留在内存中，与非请求调页没有区别；如果需要页但不是内存驻留页面时：需要检测页面并将其从磁盘加载到内存中：

\*如果进程试图访问那些尚未调入内存的页面（标记为无效位的页面），会产生缺页错误

3.处理缺页错误：

检查这个进程的内部表确定引用是否为有效内存访问；如果引用无效终止进程，如果有效：找到一个空闲帧，调度一个磁盘操作，将所需要的页面读到刚分配的帧，磁盘读取完成时，修改进程内部表和页表，以指示该页现在处于内存中。

重新启动被陷阱中断的指令

\*极端情况：启动进程时内存中没有页面，纯请求调页，将指令指针设置为进程的第一条指令

\*有效访问时间：若无缺页错误，有效访问时间=内存访问时间，如果存在缺页错误，设其概率为p

有效访问时间=（1-p）\*内存访问时间+p\*缺页错误时间

任何情况下，缺页错误的处理都有三个重要组成部分：

处理缺页错误中断

读入界面，

重启进程

\*空闲帧列表：当出现缺页错误时，操作系统必须将所需页面从辅助存储器带入主存

大多数操作系统都维护一个空闲帧列表-一个用于满足此类请求的空闲帧池

会在分配前将帧内容清零：按需零填充

当系统启动时候，所有可用内存放在空闲帧列表中

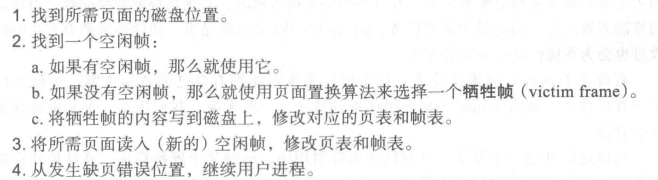
请求调页的性能优化：写时复制：允许父进程和子进程最初共享内存中的相同页面，如果任一进程修改了共享界面，则创建共享页面的副本

\*如果没有空闲帧：每个进程可能会突然试图使用其所有页面；内存不仅用于保存程序页面，内核，IO缓存也消耗大量内存

->页面置换-找到驻留在内存但没有在使用的页面，换出内存：同一个界面可能会多次进入内存

4.页面置换：

如果没有空闲帧，查找当前不在使用的一个帧，释放它（将其内容写到交换空间并修改页表和其他的表）以表示其不在内存。



加倍了缺页错误的处理时间，相应增加了有效访问时间

\*采用修改位（脏位）可减少这种开销

\*页面置换是请求调页的基础，它完成了逻辑内存和物理内存之间的分离。-在较小的物理内存上提供较大的虚拟内存

帧分配算法和页面置换算法的的设计是关键

\*帧分配算法：如果有多个进程在内存中，该如何为每个进程分配帧

为每个进程分配多少帧

\*页面替换算法：需要页面置换时，应该选择那些帧

评估页面置换算法：针对特点给的内存引用串运行某个置换算法并计算该字符串的缺页错误数

\*随帧数量的增加，缺页错误下降是我们期望的结果

5.页面置换算法：

FIFO先进先出：存在belady异常



性能不是很理想

\*\*Belady异常：对于某些置换算法随着分配帧数量的增加，缺页率错误可能增加

最优页面置换：置换在最长时间内不会使用的页面



最难实现，因为需要引用串的未来知识，但最低缺页错误率

最近最少使用算法LRU：替换最长时间内未使用的页面

将最后一次使用的时间与每页关联

需要硬件辅助来确定由上次使用时间定义的帧的顺序

\*\*实现：计数器：每个条目关联一个使用时间域；页码堆栈：页码被引用时放顶部，最少的总在下方，采用首尾双向链表实现，费时但不用搜索

近似LRU页面置换算法：引用位，额外引用位，第二次机会算法（时钟算法）采用循环队列，增强型第二次机会算法

\*\*基于技计数的页面置换：保留一个计数器，记录每页的引用次数，LFU最不经常使用算法，最常使用算法

\*\*页面缓冲算法：维护一个修改页面的列表，保留一个空闲帧并记录帧中页面分配情况

\*帧分配：帧的最小数

最小帧数由计算机架构定义，最大值未系统中的可用物理内存总帧数

\*固定分配：平均分配；比例分配-按进程大小分配

\*优先级分配

\*帧置换：全局置换-进程之间选，不过空不空；局部置换：从自己的一组分配帧里选

\*非均匀内存访问：具有明显不同内存访问时间的系统

\*系统抖动：缺页错误率过高导致的高频页面调度-抖动，如果一个进程调页时间多于执行，进程抖动

采用局部置换算法或者优先级置换算法可以限制

策略：提供足够多的帧

局部性模型

\*内存映射文件：允许一部分虚拟内存与文件进行逻辑关联，显著性能提高

第十讲 大容量存储

1.磁盘工作机制：移动磁头的磁盘装置

\*读取或写入时，磁头必须被定位在期待的磁道，从被期望的柱面和扇区开始

\*寻道时间：定位到期望柱面花费的时间

\*旋转延时：磁盘旋转到所要扇区的时间

\*平均旋转延迟时间=磁盘旋转一周时间的一半

\*固态硬盘SSD为非易失性存储无磁头不适用调度算法，DRAM为易失性存储

\*磁盘结构：现代磁盘驱动器可看作逻辑块的一维数组

逻辑块为最小传输单元，通常为512字节，逻辑块的一维数组按顺序映射到磁盘扇区：扇区0是最外层圆柱体上第一个轨迹的第一个扇区；磁头依次通过该磁道，由外向内

\*计算机访问磁盘存储的两种方式：I/O端口，主机连接存储；分布式文件系统的远程主机，网络连接存储

2.磁盘调度：

磁盘I/O的实际操作细节取决于以下几个因素：计算机系统；操作系统；I/O通道和磁盘控制器硬件特性；

操作系统负责高效的使用硬件—对于磁盘驱动器，这意味着具有快速的访问时间和较宽的磁盘带宽

最小化寻道时间：寻到时间是磁臂移动到目标扇区的柱面的时间

\*磁盘带宽是传输的总字节数除以从服务请求开始到最后传输完成之间的总时间

\*每当进程需要磁盘io操作，系统调用：

输入还是输出操作

磁盘地址

内存地址

传输的扇区数

\*如果所需的磁盘驱动器和控制器空闲，立即处理，忙则任何新服务请求都添加到磁盘驱动器的待处理请求队列

磁盘IO的请求可用来源：操作系统，系统进程，用户进程

\*操作系统维护每个磁盘或设备的请求队列

磁盘调度算法：

\*先来先服务：最公平的但不提供最快服务

\*SSTF调度：最短寻到时间优先：处理离当前磁头位置的最短寻道时间请求

本质上是一种SJF调度，会导致一些饥饿

\*优先级（PRI）：调度的控制不收磁盘管理软件的控制，目标不是优化磁盘利用率，短批作业和交互式作业具有更高优先级，提供良好响应时间，较长作业等待过长，数据库系统糟糕策略

\*SCAN调度：扫描算法：从一端开始，向另一端移动，需要知道其移动方向

\*C-SCAN调度：循环扫描，提供更均匀的的等待时间，其磁头到达另一端时，立刻返回磁盘开头，不处理任何回程上的请求，基本将柱面作为一个环链，将最后柱面连到首个环面

\*look调度

遵循scan和C\_scan但不到盘片外援只到达一个方向的最远请求为止。

选择调度算法：

SSTF比较常见

文件分配方式可以极大影响磁盘服务的性能；

目录和索引块的位置也能影响磁盘服务性能

Scan和C——scan对于磁盘负载较大的系统性能更好

3.磁盘管理：

\*磁盘格式化：

\*\*低级格式化或物理格式化：将磁盘划分为磁盘控制器可读写的扇区

\*\*要使用磁盘保存文件，操作系统仍然需要在磁盘上记录自己的数据结构

分区；逻辑格式化或创建文件系统

\*存储设备管理：

根分区包含操作系统，其他分区可以容纳其他操作系统、其他文件系统，也可以是原始分区：根分区启动时挂载

挂载时，检查文件系统一致性

引导块可以指向包含足够代码的引导卷或加载程序块，以了解如何从文件系统加载内核

\*引导块：为了开始运行计算机，必须有一个程序来运行

\*\*原始磁盘访问适用于希望自己进行数据管理的应用程序，操作系统不介入管理

\*坏块

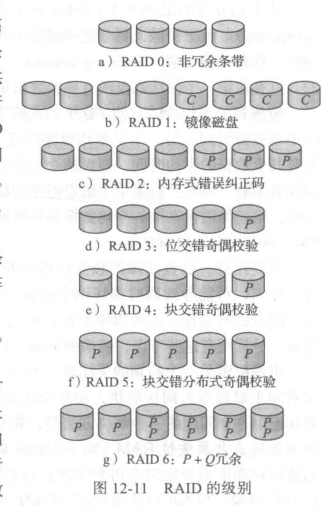
\*交换空间管理

\*磁盘连接

4.RAID磁盘冗余阵列技术

多种磁盘组织技术称为磁盘冗余阵列，通常用于处理性能和可靠性问题

镜像提高高可靠性，分条提供高数据传输率



RAID0块分条但无冗余

RAID1磁盘镜像

RAID2也成ECC，字节分散在磁盘上，实际并不使用

RAID5是最常见的奇偶校验RAID系统

略

第十章 文件系统

\*文件系统提供机制，对计算机操作系统与所有用户的数据与程序进行在线存储和访问

\*文件系统由两个不同的部分组成：文件集合：每个文件存储相关数据；目录结构，用于组织系统内的所有文件并提供文件信息。

\*文件系统位于设备上

1.概念：操作系统对存储设备的物理属性加以抽象，从而定义逻辑存储单位，即文件，是抽象数据类型，从用户角度来看，文件是逻辑外存的最小分配单元，数据通过文件才能写到外存

特点：长期存在；可共享的

2.文件操作：

创建文件：在文件系统中找到空间，在目录中创建新文件的条目

写文件：使用一个系统调用指定文件名称和要写入文件的信息，保留写指针，每次写操作，写指针更新

读文件：使用系统调用知名文件名称和需要文件的下一个块放在那里，保留读指针，指向下一次读取操作的文件位置

\*\*当前文件位置指针指向当前操作位置

重新定位文件（文件定位）：搜索目录以寻找适当的条目，并且将当前文件位置指针重新定位到给定值

删除文件：找到对应文件释放其文件空间

截断文件：删除内容保留属性，让文件重置为0，释放空间

\*操作系统有一个打开文件表，用以维护所有打开文件的信息

每个打开文件都有一些关联信息：文件指针，文件打开计数

文件磁盘位置，访问权限

\*文件锁：对于多个进程共享的文件很有用

类似读者写者锁：共享锁类似读者锁，独占锁类似写者锁

操作系统可提供强制或建议文件锁定机制

3.文件结构：

文件类型可以用于指示文件的内部结构

让操作系统支持多个文件结构会使其变得复杂

系统必须至少支持可执行文件结构，以便能加载和运行程序

内部文件结构：将多个逻辑记录包装到物理块中，逻辑记录大小、物理块大小和打包技术确定了每个物理块有多少逻辑记录，打包可以通过用户应用程序或操作系统来完成，不管如何，文件都可当作块的序列，所有基本IO功能都以块为单位运行

由于磁盘空间总是按块为单位分配，会产生内部碎片

\*访问方法：

\*\*文件存储信息，使用文件时，必须访问这些信息，将其读到计算机内存

顺序访问：文件信息按顺序加以处理，最常见

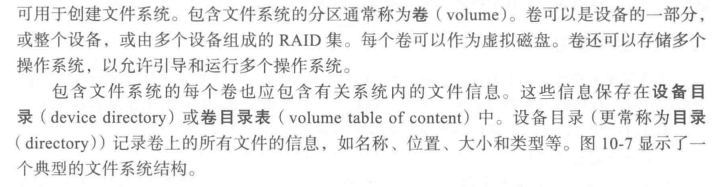
直接访问/相对访问：文件由固定长度的逻辑记录组成，以允许程序按任意顺序进行快速读取和写入记录（需要修改文件操作以包含块号作为参数）

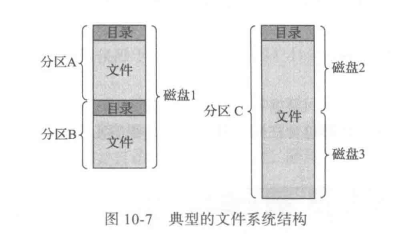
其他：建立在直接访问方法上，通常涉及创建文件索引

\*分区可用于限制但各文件系统的大小

\*硬盘的每个分区都可创建文件系统

包含文件系统的分区称为卷





5.目录：

目录可视为符号表，将文件名称转成目录条目

目录结构操作：

搜索文件

创建文件

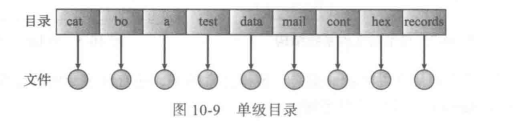
删除文件

遍历目录：遍历目录内的文件，及目录内每个文件的目录条目的内容

重命名文件：重命名文件也允许改变其在目录结构内的位置

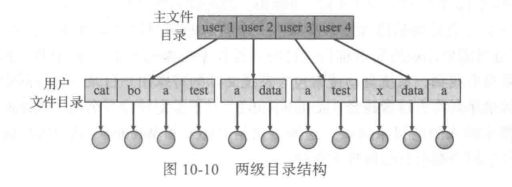
遍历文件系统：可能希望访问目录和目录结构内的每个文件，为了可靠性，定期备份文件系统到磁带上

\*单级目录

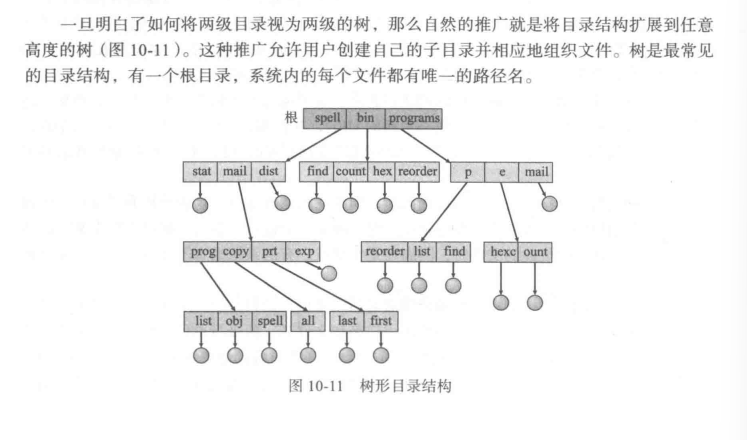


当文件数量增加，难以记住所有文件名称

\*两级目录:每个用户都有自己的UFD



\*树形目录：



每个文件有唯一路径名

每个进程有一个当前目录，包含对进程当前感兴趣的大多数文件

路径名：绝对路径：从根开始；

相对路径：从当前目录开始

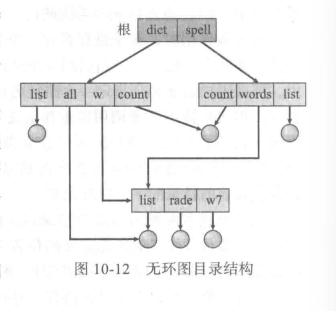
需要关注如何删除目录，目录为空才允许删和直接删，后者危险

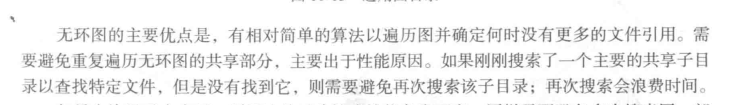
可以访问其他用户文件

\*无环图目录：

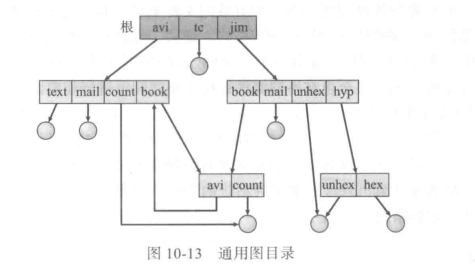
树结构禁止共享文件或目录，无环图允许目录共享子目录和文件

可以链接实现





通用图目录：



\*文件系统在系统进程之前·必须先安装

目录结构可以建构在多个卷上，这些卷必须先安装才能使其可以用于文件系统名称空间

第11章 文件系统实现

1.文件系统结构：

磁盘提供大多数外存，以维护文件系统

优势有：可以原地重写，可以直接访问它包含信息的任何块

\*为了提高IO效率，内存和磁盘间的IO传输以块为单位进行，每个块有一个或多个扇区

\*文件系统提供高效便捷的磁盘访问，以便允许轻松存储定位提取数据

结构：应用程序-逻辑文件系统-文件组织模块-基本文件系统-I/O控制-设备

\*I/O控制层包括设备驱动程序和中断处理程序在主内存以及磁盘系统之间传递信息

基本文件系统：只需向适当设备驱动程序发送通用命令，以读取和写入磁盘的物理块。

文件组织模块：将逻辑块地址转化成物理块地址，供基本的文件系统传输

逻辑文件系统:管理元数据信息，元数据包括文件系统的所有结构，但不包括实际数据，FCB文件控制块包含有关文件的信息

2.文件系统的实现：

引导控制块，有分区引导扇区NTFS，包含该卷引导操作系统所需信息，卷控制块包括卷的详细信息，NTFS中存贮在主控文件表里，目录结构用于组织文件，NTFS中存贮在主控文件表里；FCB中有一个唯一标识号与目录条目相关联

\*系统调用open（）和close（）

\*文件系统的实现需要多个磁盘和内存的结构

\*文件系统一般需要包含以下信息：如何启动存贮的操作系统，总块数，空闲块的数量和位置，目录结构及具体文件

\*\*内存中的信息用于管理文件系统并通过缓存来提升性能

内存中的安装表包含每个安装卷的有关信息，目录结构的缓存含有最近访问目录的信息

整个系统的打开文件表：包括每个打开文件的FCB的副本以及其他信息

每个进程的打开文件表：包括一个指向整个系统的打开文件表中适当条目的指针

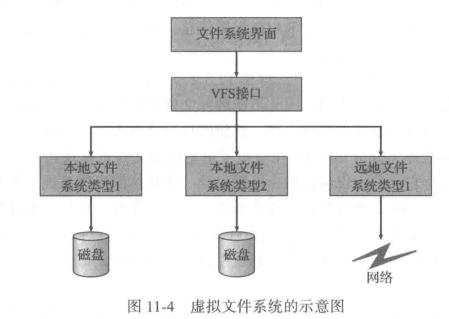
磁盘读出或写入时，缓冲区保存文件系统的块

3.分区与安装

一个磁盘可以有多个分区，一个卷可以跨越多个磁盘的分区，当没有合适文件系统时可以使用原始磁盘

\*根分区：包括操作系统内核和其他系统文件在启动时安装

4.虚拟文件系统：组成：文件系统接口，虚拟文件系统，实现文件系统或远程文件协议层

****

VFS接口：将文件系统的通用操作和实现分开

提供一种机制唯一表示网络上的文件

根据文件系统类型调用特定文件类型的操作以便处理本地请求，通过调用NFS协议程序来处理远程请求

5.目录实现：

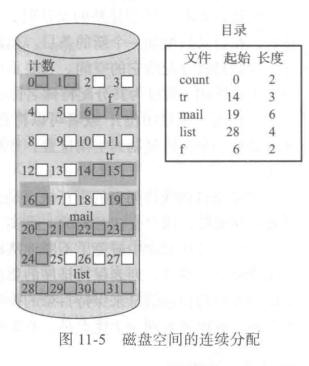
线性列表：采用文件名称和数据块指针的线性列表

哈希表：缺点，固定大小对哈希函数有依赖性

6.分配方法：

\*连续分配

每个文件在磁盘上占有一组连续的块，磁盘地址为磁盘定义了一个线性排序，用于访问连续分配文件的所需寻道数量最小；在确实需要寻道时所需寻道时间也最小



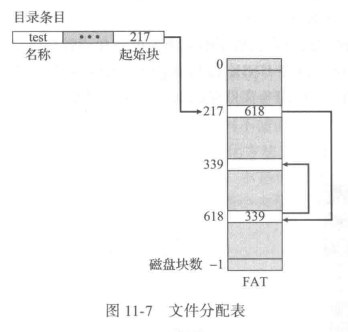
存在外部碎片以及难于为新文件找到空间

采用拓展的解决方案：最初分配一块连续空间，之后当数量不够时，会添加另一块连续空间

\*链接分配：每个文件时磁盘块的链表，磁盘块可能会散布在磁盘的任何地方，目录包括文件第一块和最后一块的指针

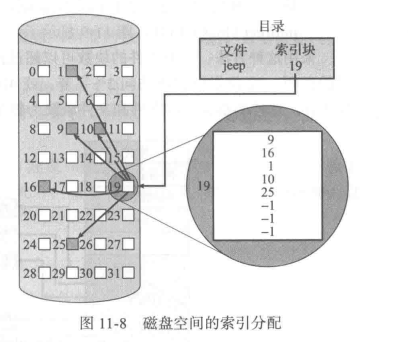
缺点：只能有效用于顺序访问文件，不支持之间访问，需要为指针分配额空间

其重要变种是文件分配表FAT用于MS-DOS



\*索引分配：解决了连续分配的外部碎片和大小声明问题，在没有fat前链接分配不能支持高效的直接访问

索引分配将所有指针放在一起，及索引块每个文件都有自己的索引块



创建文件时，索引块的所有指针为空

支持直接访问无外部碎片，但浪费空间，索引块指针的开销通常大于链接分配的指针开销

索引块大小尽可能小解决方案：链接方案，多级索引，组合方案

\*性能：最佳方法取决于文件访问类型

在创建时声明访问类型

索引分配更加复杂，其性能取决于索引的结构，文件的大小和所需块的位置

6.空闲空间管理：

系统维护一个空闲空间列表记录所有空闲磁盘空间，其按位图或位向量来实现

\*位向量：每块用一个位来表示，如果块空闲位为1，块分配位为0

查找磁盘上第一个空闲块和n个连续的空闲块时相对简单和高效

非零字对应块号码的计算：每个字的位数\*值为零的字数+第一个值为1的位的偏移

\*链表：将所有空闲磁盘块用链表链接，将指向第一空闲块的指针保存在磁盘的特殊位置上，同时也缓存道内存中，低效

\*组

\*计数

\*空间图

\*\*效率与性能：磁盘空间的有效使用很大程度上取决于磁盘分配和目录算法

影响因素：磁盘分配和目录算法；保存在文件目录条目中的数据类型，元数据结构的预分配或按需分配，固定大小或可变大小的数据结构

\*\*页面缓存使用虚拟内存技术和地址缓存页面而不是磁盘块，内存映射io使用页面缓存，通过文件系统的例程IO使用缓冲区（磁盘）缓存

统一缓冲区缓存使用相同页面来缓存内存映射页面和普通文件系统IO

第十三章：IO系统

\*计算机的两个主要操作是IO和处理

要素：端口，总线，设备控制器，IO设备

\*设备驱动程序为IO子系统提供了统一的设备访问接口

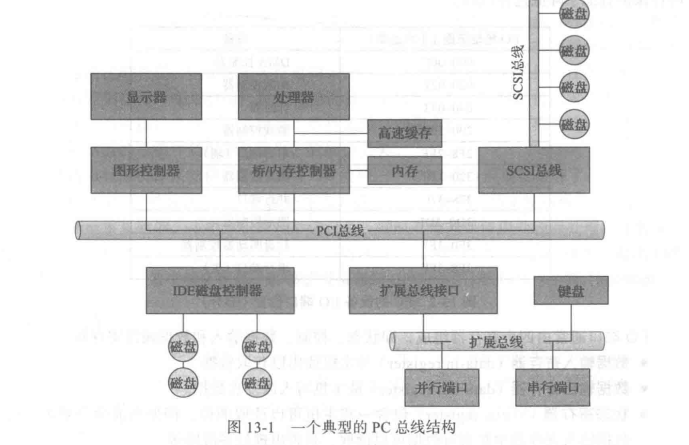
IO硬件：

端口：设备与计算机的通信经由的连接点

总线：一组线路和通过线路传输信息的严格定义的一个协议

控制器：操作端口、总线或设备的一组电子器件；IO指令控制和内存映射IO

IO端口通常由四个寄存器组成：状态，控制，数据输入输出



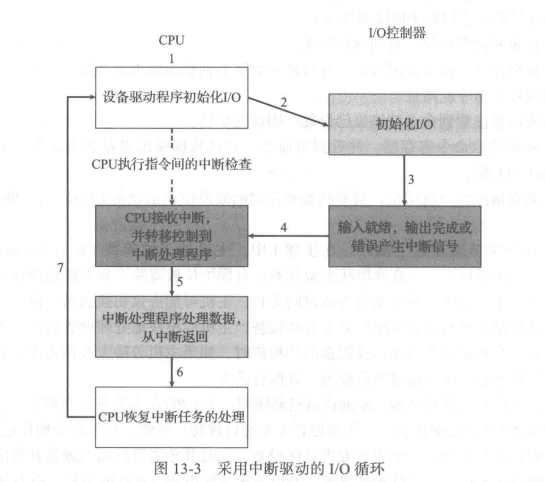
PCI总线将处理器内存子系统连到快速设备，而扩展总线连接相对较慢的设备，如键盘等

CPU与设备的通信方式：轮询，设备中断，DMA\*

轮询：传输主机重复从状态寄存器读取忙位直到该位清零可以在三个指令周期内发生，重复会低效

中断：让设备通知CPU硬件中断

CPU硬件有一条中断请求线



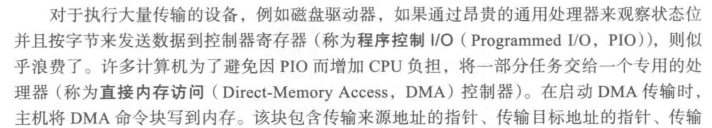
两条：可屏蔽和非屏蔽

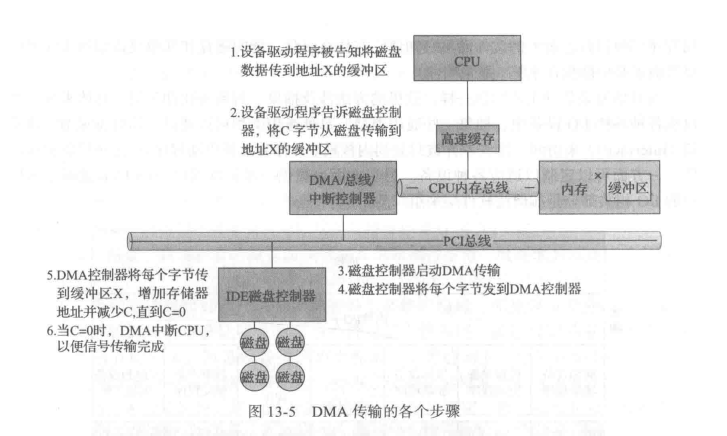
中断机制结束一个地址，也称中断向量表中的一个偏移量

多CPU系统可以并发处理中断

高性能系统要求高效的中断

DMA：直接内存访问

即，其绕过CPU，内存与IO直接传数据



直接虚拟内存访问要更高效

阻塞非阻塞异步系统调用

内核IO子系统：

IO子系统协调应用程序和内核其他部分可用的大量服务

流IO内部异步，用户进程与流头通信同步

\*缓存区是一块内存区域，用于保存在两个设备之间或者在设备和应用程序间传输的数据

\*缓存：保存数据副本的告诉内存区域