码农求职小助手：计算机操作系统面试高频题

笔记本： 7-计算机操作系统
创建时间： 2019/9/7 21:42 **更新时间：** 2019/9/7 21:42
作者： pc941206@163.com

更多资料请关注微信公众号：**码农求职小助手**

概述

1、基本特征

1.1、 并发

并发是指宏观上在一段时间内能同时运行多个程序，而并行则指同一时刻能运行多个指令。

并行需要硬件支持，如多流水线、多核处理器或者分布式计算系统。

操作系统通过引入进程和线程，使得程序能够并发运行。

1.2、共享

共享是指系统中的资源可以被多个并发进程共同使用。

有两种共享方式：互斥共享和同时共享。

互斥共享的资源称为临界资源，例如打印机等，在同一时间只允许一个进程访问，需要用同步机制来实现对临界资源的访问。

1.3、虚拟

虚拟技术把一个物理实体转换为多个逻辑实体。

主要有两种虚拟技术：时分复用技术和空分复用技术。

多个进程能在同一个处理器上并发执行使用了时分复用技术，让每个进程轮流占有处理器，每次只执行一小个时间片并快速切换。

虚拟内存使用了空分复用技术，它将物理内存抽象为地址空间，每个进程都有各自的地址空间。地址空间的页被映射到物理内存，地址空间的页并不需要全部在物理内存中，当使用到一个没有在物理内存的页时，执行页面置换算法，将该页置换到内存中。

1.4、异步

异步指进程不是一次性执行完毕，而是走走停停，以不可知的速度向前推进。

2、基本功能

2.1、进程管理

进程控制、进程同步、进程通信、死锁处理、处理机调度等。

2.2、内存管理

内存分配、地址映射、内存保护与共享、虚拟内存等。

2.3、文件管理

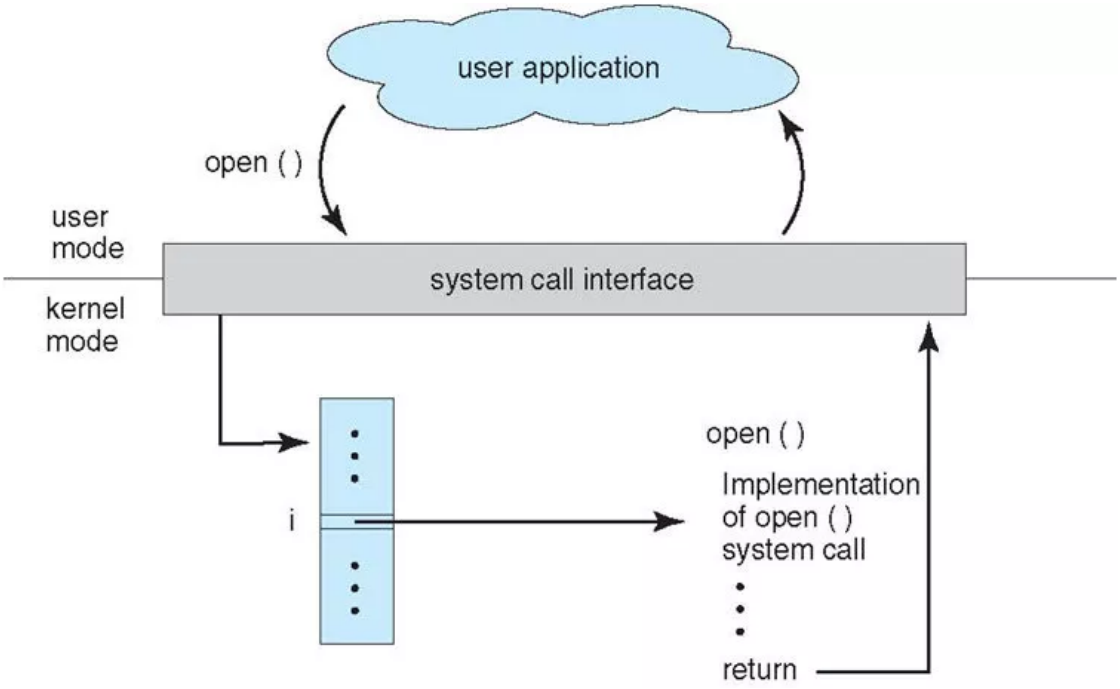
文件存储空间的管理、目录管理、文件读写管理和保护等。

2.4、设备管理

完成用户的 I/O 请求，方便用户使用各种设备，并提高设备的利用率。主要包括缓冲管理、设备分配、设备处理、虚拟设备等。

3、系统调用

如果一个进程在用户态需要使用内核态的功能，就进行系统调用从而陷入内核，由操作系统代为完成。



Linux 的系统调用主要有以下这些：

Task	Commands
进程控制	fork(); exit(); wait();
进程通信	pipe(); shmget(); mmap();
文件操作	open(); read(); write();
设备操作	ioctl(); read(); write();
信息维护	getpid(); alarm(); sleep();
安全	chmod(); umask(); chown();

4、大内核和微内核

4.1、大内核

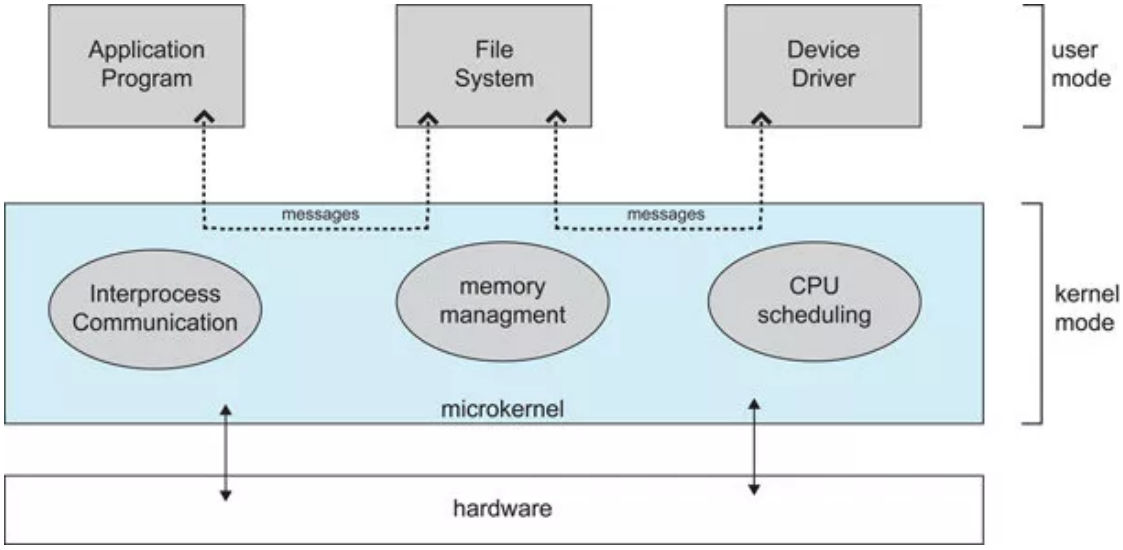
大内核是将操作系统功能作为一个紧密结合的整体放到内核。由于各模块共享信息，因此有很高的性能。

4.2、微内核

由于操作系统不断复杂，因此将一部分操作系统功能移出内核，从而降低内核的复杂性。移出的部分根据分层的原则划分成若干服务，相互独立。

在微内核结构下，操作系统被划分成小的、定义良好的模块，只有微内核这一个模块运行在内核态，其余模块运行在用户态。

因为需要频繁地在用户态和核心态之间进行切换，所以会有一定的性能损失。



5、中断分类

5.1、外中断

由 CPU 执行指令以外的事件引起，如 I/O 完成中断，表示设备输入/输出处理已经完成，处理器能够发送下一个输入/输出请求。此外还有时钟中断、控制台中断等。

5.2、异常

由 CPU 执行指令的内部事件引起，如非法操作码、地址越界、算术溢出等。

5.3、陷入在用户程序中使用系统调用。

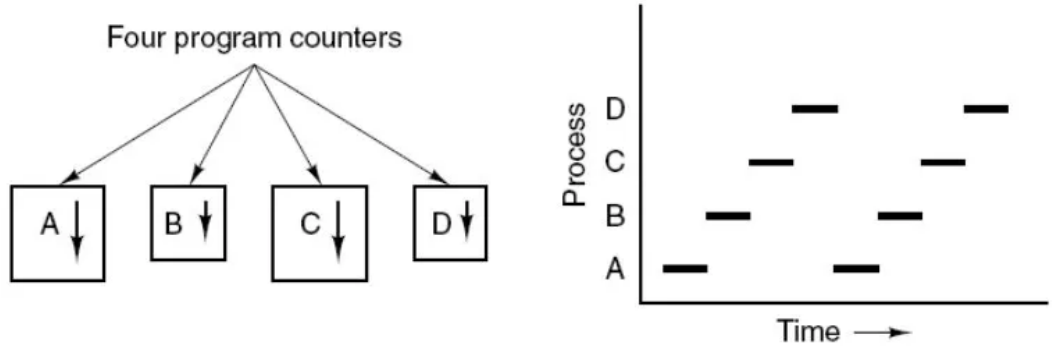
一、进程与线程

进程

进程是资源分配的基本单位。

进程控制块 (Process Control Block, PCB) 描述进程的基本信息和运行状态，所谓的创建进程和撤销进程，都是指对 PCB 的操作。

下图显示了 4 个程序创建了 4 个进程，这 4 个进程可以并发地执行。

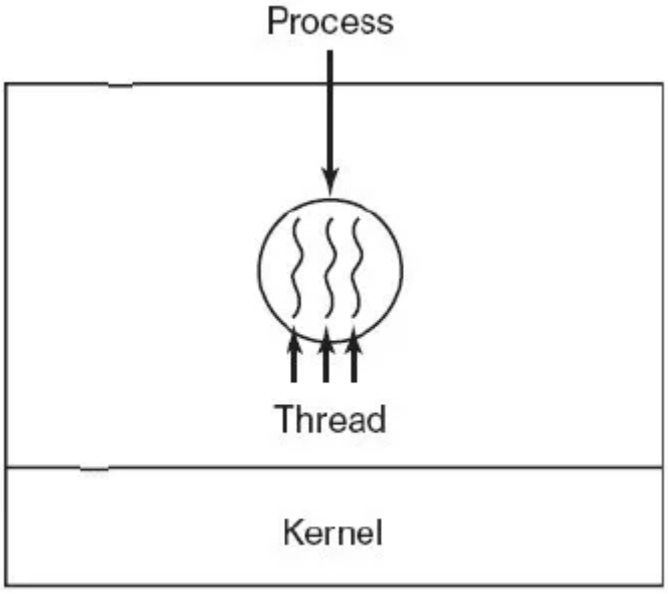


线程

线程是独立调度的基本单位。

一个进程中可以有多个线程，它们共享进程资源。

QQ 和浏览器是两个进程，浏览器进程里面有很多线程，例如 HTTP 请求线程、事件响应线程、渲染线程等等，线程的并发执行使得在浏览器中点击一个新链接从而发起 HTTP 请求时，浏览器还可以响应用户的其它事件。



区别

Ⅰ 拥有资源

进程是资源分配的基本单位，但是线程不拥有资源，线程可以访问隶属进程的资源。

Ⅱ 调度

线程是独立调度的基本单位，在同一进程中，线程的切换不会引起进程切换，从一个进程中的线程切换到另一个进程中的线程时，会引起进程切换。

Ⅲ 系统开销

由于创建或撤销进程时，系统都要为之分配或回收资源，如内存空间、I/O 设备等，所付出的开销远大于创建或撤销线程时的开销。类似地，在进行进程切换时，涉及当前执行进程 CPU 环境的保存及新调度进程 CPU 环境的设置，而线程切换时只需保存和设置少量寄存器内容，开销很小。

Ⅳ 通信方面

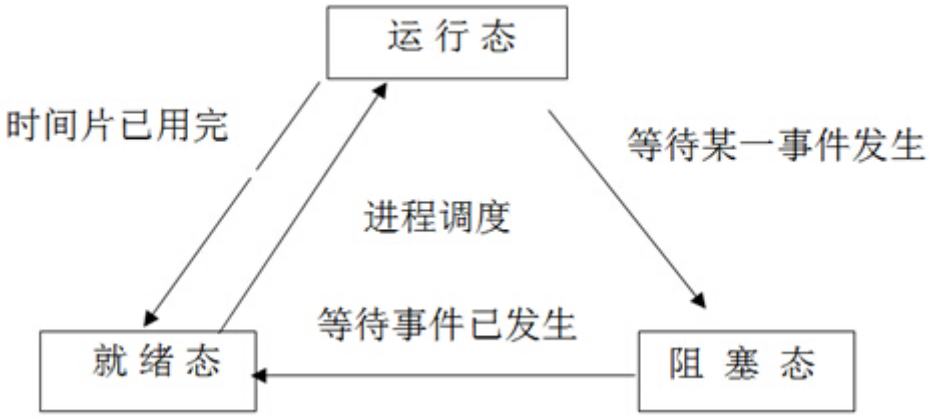
线程间可以通过直接读写同一进程中的数据进行通信，但是进程通信需要借助 IPC。

1、简单说说进程、线程以及它们的区别

进程： 是系统进行资源分配和调度的一个独立单位，是最小的资源管理单位。

线程： 是进程的一个实体，是 CPU 调度和分派的基本单位，是最小的 CPU 执行单元。线程自己不拥有任何系统资源，但是它可以访问其隶属进程的全部资源。所以线程创建、撤销、切换的开销远小于进程，一个进程可以拥有多个线程。

2、进程的状态和转换



三态模型： 一个进程从创建而产生至撤销而消亡的整个生命周期，可以用一组状态加以划分，根据三态模型，进程的生命周期可分为如下三种进程状态：

1. **运行态**(running)：占有处理器正在运行；
2. **就绪态**(ready)：具备运行条件，等待系统分配处理器以便运行；
3. **阻塞态**(blocked)：不具备运行条件，正在等待某个事件的完成。

处于运行态的进程由于出现等待事件而进入阻塞态，直到事件结束则由阻塞态进入就绪态，而处理器的调度策略有会引起运行态和就绪态之间的切换【比如进程被选中进入运行态，时间片用完进入就绪态】

3、进程间的通信方式有哪些

- 管道、命名管道、消息队列、信号量、信号、共享内存、套接字

进程间通信

每个进程都各自拥有不同的用户地址空间，任何一个进程的全局变量在另一个进程中都看不到，所以**进程之间要交换数据必须通过内核**，在内核中开辟一块缓冲区，进程A把数据从用户空间拷到内核缓冲区，进程B再从内核缓冲区把数据读走，内核提供的这种机制称为**进程间通信。**

3.1、管道(pipe)

管道是一种半双工的通信方式，数据只能单向流动，而且只能在具有亲缘关系的进程间使用。进程的亲缘关系通常是指父子进程关系。【是由内核管理的一个缓冲区，速度慢，容量有限】

3.2、命名管道 (named pipe)

命名管道也是半双工的通信方式，但是它允许无亲缘关系进程间的通信。

3.3、消息队列(message queue)

消息队列是由消息组成的链表，存放在内核中并由消息队列标识符标识。消息队列克服了信号传递信息少、管道只能承载无格式字节流以及缓冲区大小受限等缺点。

是用于两个进程之间的通讯，首先在一个进程中创建一个消息队列，然后再往消息队列中写数据，而另一个进程则从那个消息队列中取数据。需要注意的是，消息队列是用创建文件的方式建立的，如果一个进程向某个消息队列中写入了数据之后，另一个进程并没有取出数据，即使向消息队列中写数据的进程已经结束，保存在消息队列中的数据并没有消失，也就是说下次再从这个消息队列读数据的时候，就是上次的数据。

3.4、信号量(semophore)

信号量是一个计数器，可以用来控制多个进程对共享资源的访问。

它常作为一种锁机制，防止某进程正在访问共享资源时，其他进程也访问该资源。因此，主要作为进程间以及同一进程内不同线程之间的同步手段。

3.5、信号 (sinal)

用于通知接收进程某个事件已经发生。

3.6、共享内存(shared memory)

共享内存由一个进程创建，但多个进程都可以访问。【两个不同进程 A、B 共享内存的意思是：同一块物理内存被映射到进程 A、B 各自的进程地址空间。进程 A 可以即时看到进程 B 对共享内存中数据的更新，反之亦然】

共享内存是最快的 IPC(进程间通信) 方式，它是针对其他进程间通信方式运行效率低而专门设计的。它往往与其他通信机制，如信号量，配合使用，来实现进程间的同步和通信。

3.7、套接字(socket)

套接字也是一种进程间通信机制，与其他通信机制不同的是，它可用于不同机器间的进程通信。

4、进程（或作业）的调度算法有哪些

- 先来先服务、短进程优先、优先权调度算法、高响应比优先调度算法、时间片轮转调度算法、多级队列调度算法

4.1、先来先服务（FCFS，First-Come-First-Served)

按照进程进入就绪队列的先后次序来选择进程。

4.2、短进程优先（SPF,Shortest Process Next)

从就绪队列中选出一个估计运行时间最短的进程，将处理机分配给它。

4.3、优先权调度算法(Priority)

按照进程的优先权大小来调度。

4.4、高响应比优先调度算法（HRRN，Highest Response Ratio Next)

按照高响应比（（已等待时间 + 要求运行时间）/ 要求运行时间）优先的原则【等待时间长和运行时间短都会增加其优先值】，每次先计算就绪队列中每个进程的响应比，然后选择其值最大的进程投入运行。

4.5、时间片轮转调度算法（RR，Round-Robin)

当某个进程执行的时间片用完时，调度程序便停止该进程的执行，并将它送就绪队列的末尾，等待分配下一时间片再执行。然后把处理机分配给就绪队列中新的队首进程，同

时也让它执行一个时间片。这样就可以保证就绪队列中的所有进程，在一给定的时间内，均能获得一时间片处理机执行时间。

4.6、多级队列调度算法

多队列调度是根据进程的性质和类型的不同，将就绪队列再分为若干个子队列，所有的进程按其性质排入相应的队列中，而不同的就绪队列采用不同的调度算法。

5、同步和互斥的区别，同步、异步、阻塞、非阻塞的区别

5.1、同步和互斥【同步体现的是一种协作性，互斥体现的是一种排他性】

同步：就是并发的线程在一些关键点上可能需要互相等待与互通信息，这种相互制约的等待与互通信息称为进程（线程）同步。

互斥：是指某一资源同时只允许一个访问者对其进行访问，具有唯一性和排它性。但互斥无法限制访问者对资源的访问顺序，即访问是无序的。

5.2、同步、异步【关注的是消息通信机制】

同步： 就是指调用者会主动等待调用的返回结果。

异步： 就是指调用者不会主动等待调用结果，而是在调用发生后，被调用者通过状态、通知来通知调用者。

5.3、阻塞、非阻塞 【关注的是程序在等待调用结果（消息，返回值）时的状态】

阻塞： 是指调用结果返回前，当前线程会被挂起，即阻塞。

非阻塞： 是指即使调用结果没返回，也不会阻塞当前线程。

6、线程【进程】同步的方式有哪些

6.1、线程同步的基本概念

所谓同步，就是并发的线程在一些关键点上可能需要互相等待与互通信息，**这种相互制约的等待与互通信息称为进程（线程）同步。**

> “同”其实是协同，而不是同时，同步体现的是一种协作性。
>
> 互斥（指某一资源同时只允许一个访问者对其进行访问）体现的是一种排他性。一般情况下，同步已经实现了互斥，特别是写入资源的情况必定是互斥的。

临界资源是指一次仅允许一个线程使用的资源。

临界区指的是一个访问共用资源（被多个线程共享的临界资源）的程序片段，而这些共用资源又无法同时被多个线程访问的特性。当有线程进入临界区段时，其他线程或是进程必须等待（例如：bounded waiting 等待法），有一些同步的机制必须在临界区段的进入点与离开点实现，以确保这些共用资源是被互斥获得使用，例如：semaphore。只能被单一线程访问的设备，例如：打印机。

6.2、线程同步的方式

临界区用于单个进程中线程间的同步；互斥量、信号量、事件用于多个进程间的各个线程间实现同步。

1、临界区：

使用临界区对象。拥有临界区对象的线程可以访问被保护起来的资源或代码段，其他线程若想访问，则被挂起，直到拥有临界区对象的线程放弃临界区对象为止【只用于同一进程】

2、互斥量：

采用互斥对象机制，只有拥有互斥对象的线程才有访问公共资源的权限，因为互斥对象只有一个，所以可以保证公共资源不会同时被多个线程访问。【互斥对象和临界区对象非常相似，只是其允许在进程间使用，而临界区只限制于同一进程的各个线程之间使用】

3、信号量：

它允许多个线程同一时刻访问同一资源，但是需要限制同一时刻访问此资源的最大线程数目。

> 信号量（Semaphore）是一个整型变量，可以对其执行 down 和 up 操作，也就是常见的 P 和 V 操作。
>
> down ：如果信号量大于 0 ，执行 -1 操作；如果信号量等于 0，进程睡眠，等待信号量大于 0;
>
> up ： 对信号量执行 +1 操作，唤醒睡眠的进程让其完成 down 操作。
>
> down 和 up 操作需要被设计成原语，不可分割，通常的做法是在执行这些操作的时候屏蔽中断。如果信号量的取值只能为 0 或者 1，那么就成为了 互斥量（Mutex） ，0 表示临界区已经加锁，1 表示临界区解锁。

信号量（semaphore）的数据结构为一个值和一个指针，指针指向等待该信号量的下一个进程。信号量的值 S 与相应资源的使用情况有关。当 S 大于 0 时，表示当前可用资源的数量；当 S 小于 0 时，其绝对值表示等待使用该资源的进程个数。注意，信号量的值仅能由 PV 操作来改变。

执行一次 P 操作意味着请求分配一个单位资源，因此 S 的值减 1；当 S < 0 时，表示已经没有可用资源，请求者必须等待别的进程释放该类资源，它才能运行下去。

而执行一个 V 操作意味着释放一个单位资源，因此 S 的值加 1；若 S < 0，表示有某些进程正在等待该资源，因此要唤醒一个等待状态的进程，使之运行下去。

4、事件（信号）：

事件机制，则允许一个线程在处理完一个任务后，主动唤醒另外一个线程执行任务。【进程间通信中唯一的一个异步机制】

7、什么是缓冲区溢出，有什么危害，原因是什么?

1、缓冲区溢出:

是指当计算机向缓冲区内填充数据时超过了缓冲区本身的容量，溢出的数据覆盖在合法数据上。

2、危害:

在当前网络与分布式系统安全中，被广泛利用的 50% 以上都是缓冲区溢出。缓冲区溢出中，最为危险的是堆栈溢出，因为入侵者可以利用堆栈溢出，在函数返回时改变返回程序的地址，让其跳转到任意地址，带来的危害一种是程序崩溃导致拒绝服务，另外一种就是跳转并且执行一段恶意代码，比如得到 shell，然后为所欲为。

通过往程序的缓冲区写超出其长度的内容，造成缓冲区的溢出，从而破坏程序的堆栈，使程序转而执行其它指令，以达到攻击的目的。

3、造成缓冲区溢出的主原因:

是程序中没有仔细检查用户输入的参数。

二、死锁

2.1 什么是死锁?

由于系统中存在一些不可剥夺资源，而当两个或两个以上进程占有自身资源，并请求对方资源时，会导致每个进程都无法向前推进，这就是**死锁。**

2.2 死锁产生的原因?

1、竞争资源;

2、进程推进顺序不当（比如 a，b 互相等待对方发信息）。

2.3 死锁产生的必要条件?

1、互斥: 一个资源一次只能被一个进程所使用，即是排它性使用。

2、不剥夺： 一个资源仅能被占有它的进程所释放，而不能被别的进程强制剥夺。

3、请求与保持： 指进程占有自身本来拥有的资源并要求其他资源。

4、循环等待： 存在进程资源的循环等待链，链中每一个进程已获得的资源同时被下一个进程所请求。

2.4 怎么处理死锁?

死锁的处理策略：

1、鸵鸟策略；

2、预防死锁；

3、避免死锁；

4、死锁的检测及解除。

2.5 鸵鸟策略

把头埋在沙子里，假装根本没发生问题。因为解决死锁问题的代价很高，因此鸵鸟策略这种不采取任务措施的方案会获得更高的性能。**当发生死锁时不会对用户造成多大影响，或发生死锁的概率很低，可以采用鸵鸟策略。**大多数操作系统，包括 Unix，Linux 和 Windows，处理死锁问题的办法仅仅是忽略它。

2.6 预防死锁：破坏四个必要条件之一

1. 破坏互斥条件

即允许进程同时访问某些资源。但是，有的资源是不允许被同时访问的，像打印机等等。所以，这种办法并无实用价值。

2. 破坏不可剥夺条件

当一个进程已占有了某些资源，它又申请新的资源，但不能立即被满足时，它必须释放所占有的全部资源，以后再重新申请。这就相当于该进程占有的资源被隐蔽地强占了。这种预防死锁的方法实现起来困难，会降低系统性能。

3. 破坏请求与保持条件

可以实行资源预先分配策略。即进程在运行前，一次性地向系统申请它所需要的全部资源。

如果某个进程所需的全部资源得不到满足，则不分配任何资源，此进程暂不运行。只有当系统能够满足当前进程的全部资源需求时，才一次性地将所申请的资源全部分配给该进程。由于运行的进程已占有了它所需的全部资源，所以不会发生占有资源又申请资源的现象，因此不会发生死锁。

4.破坏循环等待条件：实行顺序资源分配法

首先给系统中的资源编号，规定每个进程，必须按编号递增的顺序请求资源，同类资源一次申请完。也就是说，只要进程提出申请分配资源 Ri，则该进程在以后的资源申请中，只能申请编号大于 Ri 的资源。

2.7 避免死锁

- 银行家算法【在动态分配资源的过程中，银行家算法防止系统进入不安全状态，从而避免死锁】

银行家算法：

当进程首次申请资源时，要测试该进程对资源的最大需求量，如果系统现存的资源可以满足它的最大需求量则按当前的申请量分配资源，否则就推迟分配。

当进程在执行中继续申请资源时，先测试该进程已占用的资源数与本次申请资源数之和是否超过了该进程对资源的最大需求量。若超过则拒绝分配资源。若没超过则再测试系统现存的资源能否满足该进程尚需的最大资源量，若满足则按当前的申请量分配资源，否则也要推迟分配。

安全序列：

是指系统能按某种进程推进顺序（P1, P2, P3, ..., Pn），为每个进程 Pi 分配其所需要的资源，直至满足每个进程对资源的最大需求，使每个进程都可以顺序地完成。这种推进顺序就叫安全序列【银行家算法的核心就是找到一个安全序列】。

系统安全状态：

如果系统能找到一个安全序列，就称系统处于安全状态，否则，就称系统处于不安全状态

2.8 死锁的检测和解除

即在死锁产生前不采取任何措施，只检测当前系统有没有发生死锁，若有，则采取一些措施解除死锁。

1、死锁的检测

不试图阻止死锁，而是当检测到死锁发生时，采取措施进行恢复。

- **每种类型一个资源的死锁检测**

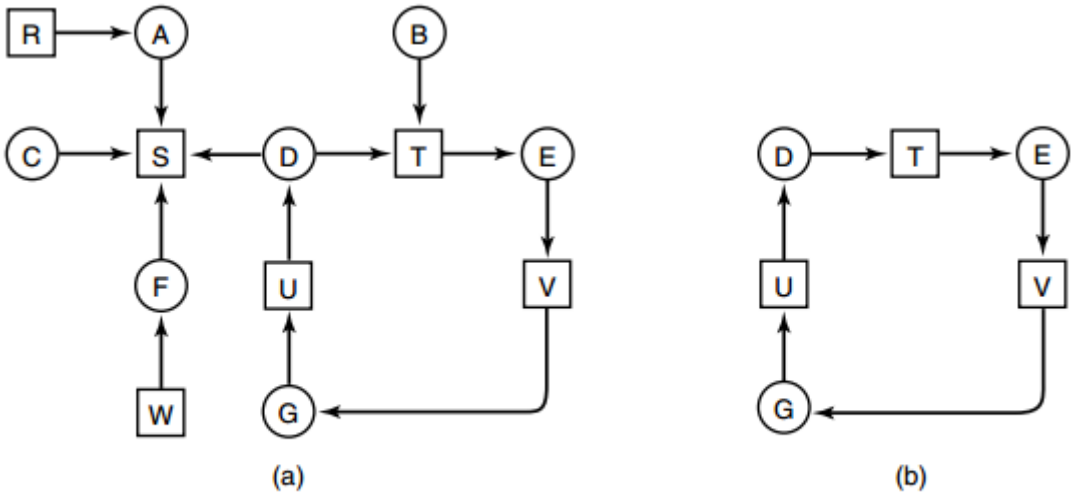
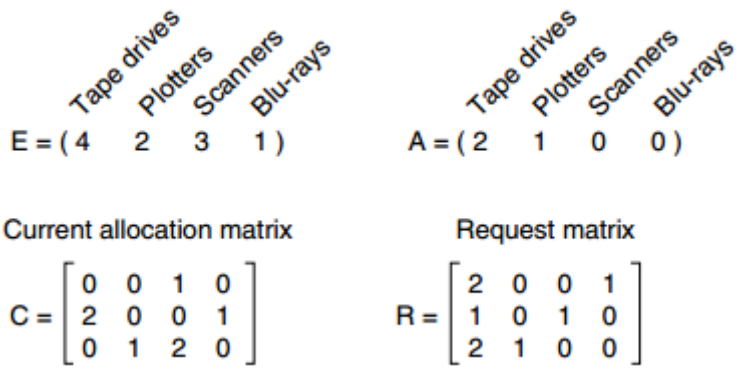


Figure 6-5. (a) A resource graph. (b) A cycle extracted from (a).

上图为资源分配图，其中方框表示资源，圆圈表示进程。资源指向进程表示该资源已经分配给该进程，进程指向资源表示进程请求获取该资源。

图 a 可以抽取出环，如图 b，它满足了**环路等待条件**，因此会发生死锁。**每种类型一个资源的死锁检测算法是通过检测有向图是否存在环来实现，从一个节点出发进行深度优先搜索，对访问过的节点进行标记，如果访问了已经标记的节点，就表示有向图存在环，也就是检测到死锁的发生。**

- **每种类型多个资源的死锁检测**



上图中，有三个进程四个资源，每个数据代表的含义如下：

E 向量：资源总量

A 向量：资源剩余量

C 矩阵：每个进程所拥有的资源数量，每一行都代表一个进程拥有资源的数量

R 矩阵：每个进程请求的资源数量

进程P1 和 P2 所请求的资源都得不到满足，只有进程 P3 可以，让 P3 执行，之后释放 P3 拥有的资源，此时 A = (2 2 2 0)。P2 可以执行，执行后释放 P2 拥有的资源，A = (4 2 2 1) 。P1 也可以执行。所有进程都可以顺利执行，没有死锁。

- **算法总结如下：**

每个进程最开始时都不被标记，执行过程有可能被标记。当算法结束时，任何没有被标记的进程都是死锁进程。

1、寻找一个没有标记的进程 Pi，它所请求的资源小于等于 A；

2、如果找到了这样一个进程，那么将 C 矩阵的第 i 行向量加到 A 中，标记该进程，并转回 1；

3、如果没有这样一个进程，算法终止。

2、死锁的解除

1、**资源剥夺**：挂起某些死锁进程，并抢占它的资源，将这些资源分配给其他死锁进程。（但应该防止被挂起的进程长时间得不到资源）；

2、**撤销进程**：强制撤销部分、甚至全部死锁进程并剥夺这些进程的资源。（撤销的原则可以按进程优先级和撤销进程代价的高低进行）；

3、**进程回退**：让一个或多个进程回退到足以避免死锁的地步。【进程回退时自愿释放资源而不是被剥夺。要求系统保持进程的历史信息，设置还原点。】

三、内存

3.1、固定分区、动态分区、分段式存储管理和分页式存储管理的区别

内存分配分为连续分配和非连续分配管理两种。

3.1.1、内存连续分配

1、单一连续分配：分为系统区和用户区，系统区供给操作系统使用，用户区供给用户使用，内存中永远只有一道程序。

2、固定分区分配：最简单的一种多道程序管理方式，它将用户内存空间划分为若干个固定大小的区域，每个分区只装入一道作业。【方法一：分区大小相等；方法二：分区大小不等，划分为含有多个较小的分区，适量的中等分区及少量的大分区】

3、动态分区分配：又称为可变分区分配，是一种动态划分内存的方法。这种分区方法不预先将内存划分，而是在进程装入内存时，根据进程的大小动态的建立分区，并使分区的大小正好适合进程的需要。因此系统中分区的大小和数目是可变的。

3.1.2、非连续分配

1、**分页式存储管理**：分页存储管理是将一个进程的地址（逻辑地址空间）空间划分成若干个大小相等的区域，称为页，相应地，将内存空间划分成与页相同大小（为了保证页内偏移一致）的若干个物理块，称为块或页框（页架）。在为进程分配内存时，将进程中的若干页分别装入多个不相邻接的块中。【只需给出一个地址，所以是一维】

分页系统中，允许将进程的每一页离散地存储在内存的任一物理块中，为了能在内存中找到每个页面对应的物理块，系统为每个进程建立一张页面映射表，简称**页表。页表的作用是实现从页号到物理块号的地址映射。**【若给定一个逻辑地址为 A，页面大小为 L，则页号 P = INT[A / L]，页内地址 W=A MOD L】

分页系统中，CPU每次要存取一个数据，都要两次访问内存（访问页表、访问实际物理地址）。为提高地址变换速度，增设一个具有并行查询能力的特殊高速缓冲存储器，称为“联想存储器”或“快表”，存放当前访问的页表项。

2、**分段式存储管理**：在分段存储管理方式中，作业的地址空间被划分为若干个段，每个段是一组完整的逻辑信息，如有主程序段、子程序段、数据段及堆栈段等，每个段都有自己的名字，都是从零开始编址的一段连续的地址空间，各段长度是不等的。【因为每段的长度是不确定的，所以不能只给一个逻辑地址通过整数除法得到段号，求余得出段内偏移，所以一定要显式给出（段号，段内偏移），因此分段管理的地址空间是二维的】

3.1.3、分页与分段的区别

对程序员的透明性：分页透明，但是分段需要程序员显式划分每个段；

地址空间的维度：分页是一维地址空间，分段是二维的；

大小是否可以改变：页的大小不可变，段的大小可以动态改变；

出现的原因：分页主要用于实现虚拟内存，从而获得更大的地址空间；分段主要是为了使程序和数据可以被划分为逻辑上独立的地址空间并且有助于共享和保护。

3.2、页面置换算法有哪些?

最佳置换算法、先进先出置换算法、最近最久未使用置换算法 LRU、时钟置换算法、最少使用置换算法：

1、**最佳置换算法（Optimal）**：即选择那些永不使用的，或者是在最长时间内不再被访问的页面置换出去。（它是一种理想化的算法，性能最好，但在实际上难于实现）；

2、**先进先出置换算法 FIFO**：总是淘汰最先进入内存的页面；

3、**最近最久未使用置换算法 LRU**（Least Recently Used）：即选择最近最久未使用的页面予以淘汰。【系统在每个页面设置一个访问字段，用以记录这个页面自上次被访问以来所经历的时间 T，当要淘汰一个页面时，选择 T 最大的页面。】

4、**时钟（Clock）置换算法**：也叫最近未用算法 NRU（Not RecentlyUsed）。该算法为每个页面设置一位访问位，将内存中的所有页面都通过链接指针链成一个循环队列。当某页被访问时，其访问位置“1”。在选择一页淘汰时，就检查其访问位，如果是“0”，就

选择该页换出；若为“1”，则重新置为“0”，暂不换出该页，在循环队列中检查下一个页面，直到访问位为“0”的页面为止。由于该算法只有一位访问位，只能用它表示该页是否已经使用过，而置换时是将未使用过的页面换出去，所以把该算法称为最近未用算法。

5、**最少使用置换算法 LFU**：该算法选择最近时期使用最少的页面作为淘汰页。

3.3、逻辑地址、物理地址、虚拟内存、操作系统的内容

1、**物理地址**：它是地址转换的最终地址，进程在运行时执行指令和访问数据最后都要通过物理地址从主存中存取，是内存单元真正的地址。

2、**逻辑地址**：是指从应用程序角度看到的内存地址，又叫相对地址。编译后，每个目标模块都是从 0 号单元开始编址，称为该目标模块的相对地址或逻辑地址。不同进程可以有相同的逻辑地址，因为这些相同的逻辑地址可以映射到主存的不同位置。用户和程序员只需要知道逻辑地址。

3、**虚拟内存**：虚拟内存是一些系统页文件，存放在磁盘上，每个系统页文件大小为 4K，物理内存也被分页，每个页大小也为 4K，这样虚拟页文件和物理内存页就可以对应，实际上虚拟内存就是用于物理内存的临时存放的磁盘空间。页文件就是内存页，物理内存中每页叫物理页，磁盘上的页文件叫虚拟页，物理页+虚拟页就是系统所有使用的页文件的总和。

虚拟内存技术：允许将一个作业分多次调入内存。可以用分页式、分段式、段页式存储管理来实现。

基于局部性原理，在程序装入时，可以将程序的一部分装入内存，而其余部分留在外存，就可以启动程序执行。在程序执行过程中，当所访问的信息不在内存时，由操作系统将所需要的部分调入内存，然后继续执行程序。另一方面，操作系统将内存中暂时不使用的内容换出到外存上，从而腾出空间放入将要调入内存的信息。这样，系统就好像为用户提供了一个比实际内存大得多的存储器，称为虚拟存储器。

4、操作系统主要包括：进程和线程的管理、存储管理、设备管理、文件管理

3.4、动态链接库和静态链接库的区别

静态连接库： 一种是 LIB 包含函数代码本身，在编译时直接将代码加入程序当中，称为静态链接库。

静态链接： 静态链接使用静态链接库，链接器从静态链接库 LIB 获取所有被引用函数，并将库同代码一起放到可执行文件中。

动态链接库： 一种是 LIB 包含了函数所在的DLL文件和文件中函数位置的信息（入口），代码由运行时加载在进程空间中的 DLL 提供，称为动态链接库。

动态链接： 使用动态链接库，允许可执行模块（.dll 文件或 .exe 文件）仅包含在运行时定位 DLL 中函数的可执行代码所需的信息。

二者的区别：

静态库本身就包含了代码，地址符号表等，而对于导入库而言，其实际的执行代码位于动态库中，导入库只包含了地址符号表等，确保程序找到对应函数的一些基本地址信息；

静态链接库是一个或多个 obj 文件的打包，所以有人干脆把从 obj 文件生成lib的过程称为 Archive 即合并到一起。当我们应用工程在使用静态链接库的时候，静态链接库要参与编译，在生成执行文件之前的链接过程中，将静态链接库的全部指令直接链接如可执行文件中，故而，在可执行文件生成以后，静态链接库 .lib 可以弃之不用；

动态链接库是作为共享函数库的可执行文件，动态链接库提供了一种方法，是进程可以调用不属于其可执行代码的函数。dll 还有助于共享数据和资源。多个应用程序可同时访问内存中单个dll副本的内容。

使用动态链接库代替静态链接库优点：dll 节省内存，减少交换操作，节省磁盘空间，更易于升级（不需要重链接和重编译），提供售后支持，提供 MFC 库类的机制，支持多语言支持。

静态链接库与动态链接库都是共享代码的方式，如果采用静态链接库，lib中的指令都全部被直接包含在最终生成的exe文件中了。但是若使用dll动态链接库，该dll不必被包含在最终的exe文件中，执行文件执行时可以动态地引用和卸载这个与exe独立的dll文件。

另一个区别是，静态链接库不能再包含其他的动态链接库或者静态库，而在动态链接库中还可以再包含其他的动态或静态链接库。

静态链接库动态链接库使用的区别在于它允许可执行模块（dll或exe文件）仅包含在运行时定位dll函数的可执行代码的所需信息。静态链接库的使用中，连接器从静态链接库获取所有被引用的函数，并将库同代码一起放到可执行文件中。

3.5、局部性原理

时间局部性： 如果程序中某条指令一旦执行，不久后该指令可能再次执行；如果某数据被访问过，不久后该数据可能再次被访问。【原因：因为在程序中存在着大量的循环操作】

空间局部性： 一旦程序访问了某个存储单元，在不久之后，其附近的存储单元也将被访问。【原因：因为指令通常是顺序存放、顺序执行的；数据也一般是以向量、数组、表等形式簇聚存储的】

3.6、中断、系统调用、库函数

1、中断

就是指在计算机执行程序的过程中，由于出现了某些特殊事情，使得 CPU 暂停对程序的执行，转而去执行处理这一事件的程序。等这些特殊事情处理完之后再回去执行之前的程序。中断一般分为三类：

1、内部异常中断：由计算机硬件异常或故障引起的中断；

2、软中断：由程序中执行了引起中断的指令而造成的中断；

3、外部中断：由外部设备请求引起的中断；

2、系统调用

系统调用是通向操作系统本身的接口，是面向底层硬件的。通过系统调用，可以使得用户态运行的进程与硬件设备(如：CPU、磁盘、打印机等)进行交互。

用户进程需要发生系统调用时，内核将调用内核相关函数来实现（如：sys_read()、sys_write()、sys_fork()）。用户程序不能直接调用这些函数，这些函数运行在内核态，CPU 通过软中断切换到内核态开始执行内核系统调用函数。

系统调用和中断的关系就在于，当进程发出系统调用申请的时候，会产生一个软中断。产生这个软中断以后，系统会去对这个软中断进行处理，这个时候进程就处于核心态了。

3、库函数

库函数是把函数放到库里，供别人使用的一种方式。【系统调用是为了方便使用操作系统的接口，而库函数则是为了人们编程的方便。】

3.7、不同进程打开了同一个文件，那么这两个进程得到的文件描述符（fd）相同吗?

两个进程中分别生成两个独立的 fd。

文件描述符与打开文件的关系?

内核中，对应于每个进程都有一个文件描述符表，表示这个进程打开的所有文件。文件描述表中每一项都是一个指针，指向一个用于描述打开的文件的数据块————file 对象，file 对象中描述了文件的打开模式，读写位置等重要信息，当进程打开一个文件时，内核就会创建一个新的 file 对象。需要注意的是，file 对象不是专属于某个进程的，不同进程的文件描述符表中的指针可以指向相同的 file 对象，从而共享这个打开的文件。file 对象有引用计数，记录了引用这个对象的文件描述符个数，只有当引用计数为 0 时，内核才销毁 file 对象，因此某个进程关闭文件，不影响与之共享同一个 file 对象的进程。

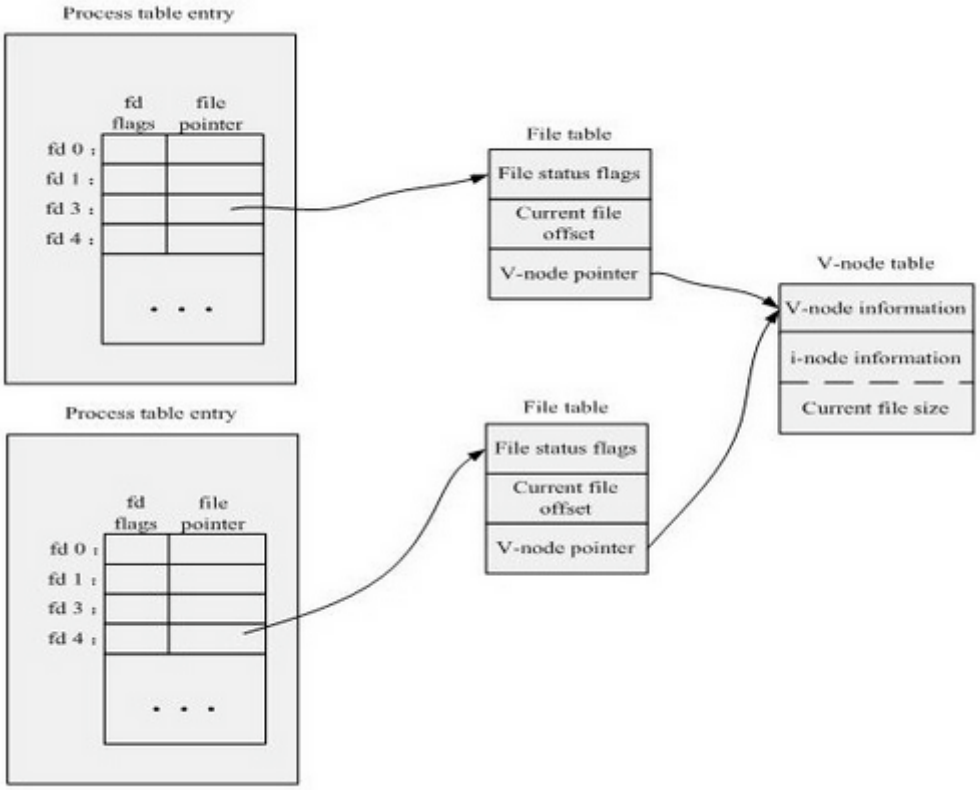


图2 两个独立进程各自打开同一个文件

4、设备管理

4.1、磁盘结构

盘面（Platter）：一个磁盘有多个盘面；

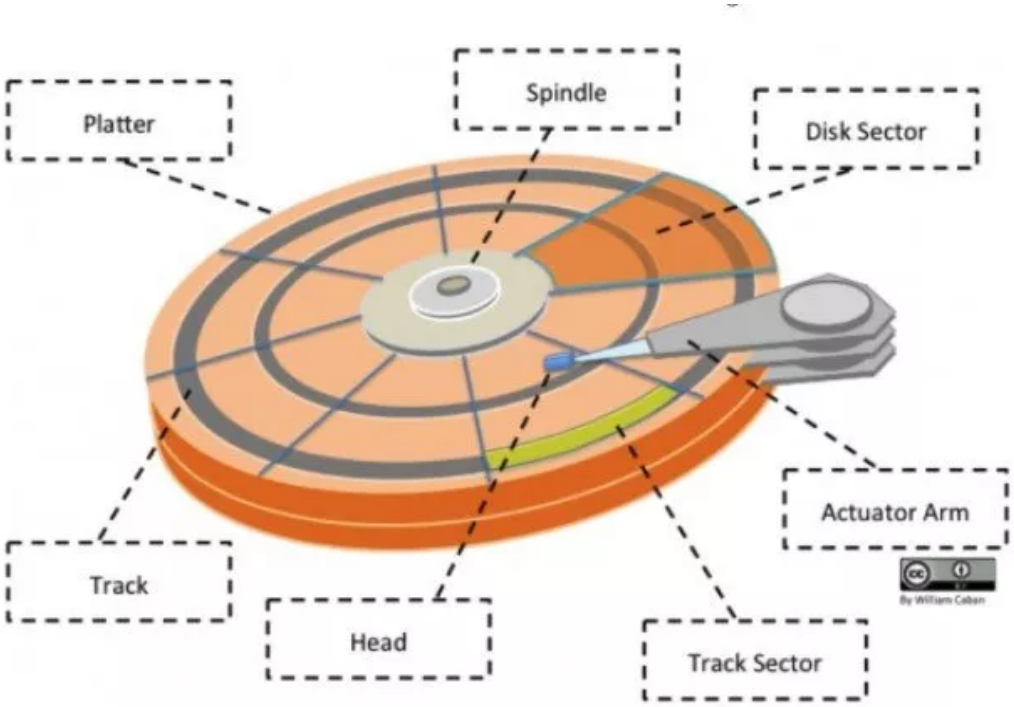
磁道（Track）：盘面上的圆形带状区域，一个盘面可以有多个磁道；

扇区（Track Sector）：磁道上的一个弧段，一个磁道可以有多个扇区，它是最小的物理储存单位，目前主要有 512 bytes 与 4K 两种大小；

磁头（Head）：与盘面非常接近，能够将盘面上的磁场转换为电信号（读），或者将电信号转换为盘面的磁场（写）；

制动手臂（Actuator arm）：用于在磁道之间移动磁头；

主轴（Spindle）：使整个盘面转动。



4.2、磁盘调度算法

读写一个磁盘块的时间的影响因素有：

> 1、旋转时间（主轴转动盘面，使得磁头移动到适当的扇区上）
>
> 2、寻道时间（制动手臂移动，使得磁头移动到适当的磁道上）
>
> 3、实际的数据传输时间

其中，寻道时间最长，因此磁盘调度的主要目标是使磁盘的平均寻道时间最短。

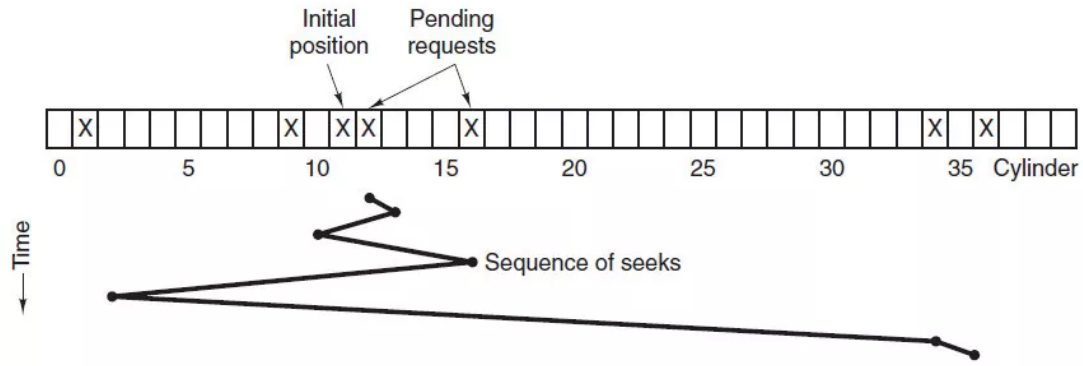
4.2.1、先来先服务 FCFS, First Come First Served

按照磁盘请求的顺序进行调度。优点是公平和简单。缺点也很明显，因为未对寻道做任何优化，使平均寻道时间可能较长。

4.2.2. 最短寻道时间优先 SSTF, Shortest Seek Time First

优先调度与当前磁头所在磁道距离最近的磁道。

虽然平均寻道时间比较低，但是不够公平。如果新到达的磁道请求总是比一个在等待的磁道请求近，那么在等待的磁道请求会一直等待下去，也就是出现饥饿现象。具体来说，两端的磁道请求更容易出现饥饿现象。



4.2.3、电梯算法 SCAN

电梯总是保持一个方向运行，直到该方向没有请求为止，然后改变运行方向。

电梯算法（扫描算法）和电梯的运行过程类似，总是按一个方向来进行磁盘调度，直到该方向上没有未完成的磁盘请求，然后改变方向。

因为考虑了移动方向，因此所有的磁盘请求都会被满足，解决了 SSTF 的饥饿问题。