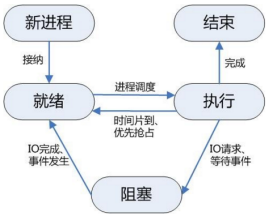
.操作系统：操作系统是管理计算机硬件的程序，为应用程序提供基础，充当硬件和用户的中介。用户视角：为了用户使用方便，提高资源利用率 系统视角：资源分配器 2.OS
三种基本类型：批处理系统、分时系统、实时系统 3.操作系统是控制程序，控制程序管理用户，程序的执行和防止错误和计算机的使用不当。4.批处理系统：脱机输入系统，
批量送入执行，自动运行作业表 优点：节省作业装入时间 缺点：CPU 经常偷闲，人机交互性差 5. 多道程序设计系统：同时在内存中驻留多个程序，当一个进程等待时，系
统会自己切换到另一个进程执行。优点：通过组织作业使 CPU 中总有一个作业可以执行，充分利用 CPU。缺点：引起作业调度，CPU 调度和内存磁盘管理等问题 6.分时系统
是多道程序设计系统的延伸，作业切换频率很高，用户可以在程序运行期间与之交互。7.（1）分时操作系统允许用户共享计算机（2）采用 CPU 调度和多道程序设计以提供用
户分时计算机的一小部分（3）在存储期中同时保存几个作业（4）操作系统保证合理的响应时间（5）提供文件系统 8.多道程序设计系统和分时是现代计算机操作系统的主题 9.
双重模式操作：分为用户模式和内核模式。系统引导时，硬件开始处于内核模式。接着装入操作系统，开始在用户模式下运行。一旦出现陷阱或中断，硬件会从用户模式切换
到内核模式。因此只要操作系统获得了对计算机的控制，它就处于内核模式。系统在讲控制交还给用户模式时切换到用户模式。10.特权指令：将能引起机器损害的指令成为
特权指令，硬件仅允许在监督程序模式下使用特权指令 11.系统调用：用户与操作系统进行交互，从而请求系统执行一些只有操作系统才能做到的指令，每个这样的请求都是
由用户调用来执行特权指令的，这种请求成为系统调用。12 硬件保护：IO：为防止用户执行非法 IO，可定义所有 IO 指令为特权指令 内存保护：通过基址寄存器和界限寄存
器来确定程序所能访问的合法地址空间并保护其他内存空间 CPU 空间：使用定时器来防止用户程序执行时间过长（作用：防止用户程序无限占用 CPU）

第二章 操作系统结构

操作系统的服务：用户：用户界面、程序执行、IO 操作、文件系统操作、通信、错误检测 系统：资源分配、统计、保护和安全 2.用户界面：命令行和图形界面 3.API（应
用程序接口）：一些列适用于程序员的函数 系统调用提供了操作系统提供的有效服务界面 4.区别：程序员调用的是 API（API 函数），然后通过与系统调用共同完成函数的功
能，跟内核无直接关系。系统调用则不与程序员进行交互的，它根据 API 函数，通过一个软中断机制向内核提交请求，以获取内核服务的接口。联系：一个 API 可能会需要一
个或多个系统调用来完成特定功能。并不是所有的 API 函数都一一对应一个系统调用，有时，一个 API 函数会需要几个系统调用来共同完成函数的功能，甚至还有一些 API 函
数不需要调用相应的系统调用（因此它所完成的不是内核提供的服务）。5.系统调用类型：进程控制、文件管理、设备管理、信息维护、通信 6.系统程序：最底层是硬件，上
面是操作系统，接着是系统程序，最后是应用程序。系统程序提供了一个方便的环境，以便开发程序和执行程序，其中一小部分只是系统调用的简单接口.(文件管理，状态信
息，文件修改，程序语言支持，程序装入和执行，通信) 6.操作系统结构：（1）简单结构：较小，简单且功能有限（2）微内核：将所有非基本部分从内核移走，并将它们实
现为系统程序和应用程序，剩余部分为微内核。优点：便于扩充操作系统，具有更好的安全性和可靠性，容易从一硬件平台设计移植到另外一个 缺点：要忍受系统功能总开
销的增加而导致系统性能下降 （3）分层方法：将操作系统模块化，分成若干层，每层建立在较低层次上，最底层为硬件最高层为用户接口。特点：采用模块化简化了操作系
统的设计和实现，每层都是利用较低层次的功能实现。但是对层的仔细认证的定义比较困难，效率较差。7.虚拟机：单个计算机的硬件抽象为几个不同的执行部件。有的系统
程序可以很容易的被应用程序调用，虽然系统程序比其他字程序的层次要高，但是应用程序还是可以将他们的一切下层当成硬件的一部分看做一个整体，这种分成方法自然而
然的逻辑延伸为虚拟机的概念。功能：提供与基本硬件相同的接口 8.策略与机制分离：机制如何做，策略做什么。策略可能会随时间或位置有所改变，在最坏情况下，每次策
略改变都会造成机制的改变。系统更需要调用机制，这样策略的改变只需重定义一些系统参数。

第三章进程

.1 进程概念 1. 进程：进程是能和其他程序并发执行的程序段在某数据集合上的一次运行过程，它是系统资源分配和调度的一个独立单位 2. 进程包含：文本段（代码段）、
堆栈段（临时数据，如函数参数，返回地址和局部变量）、数据段（包含全局变量）、堆（进程运行期间动态分配内存）。3.程序和进程的区别：a.程序是一组指令的集合，
它只规定了运行活动时所要完成的功能，本身没有运行的含义，因此程序是静态的概念，而进程是一段程序的一次运行活动，它的着眼点是活动，运行，过程，因此进程是动
态概念；b.进程是一个独立调度并能和其他进程并行运行的单位，而程序通常不能作为独立调度进行的单位；C.一个程序运行在两个不同数据集合上，就是两个不同的进程，
因此进程和程序不存在一一对应关系，一个程序可以对应多个进程，反之，一个进程至少要对应一个程序，或对应多个程序，多个进程也可以对应相同的程序； 4.多个进程
运行时，虽然文本段相同，但是数据段、堆、堆栈段不同。5.进程状态：新的、运行、等待、就绪、终止。 6. PCB：进程控制块，进程存在的唯一标示。进程状态、程序寄
存器、CPU 寄存器、CPU 调度信息、内存管理信息、记账信息、IO 状态信息。3.2 进程调度 1.调度队列：作业队列、就绪队列、设备队列（IO 队列）2.进程在其生命周期中
会在各种队列之间迁移。进程选择是由相应的调度程序来完成的。3.长期调度程序：从大容量存储设备的缓冲池中选择进程并将它们装入内存以执行。长期调度程序控制控制
着到程序设计的程度，即内存的进程数量。进程创建的平均速度等于进程离开系统的平均速度。因此长期调度程序需要在离开系统时才被唤起。由于每次执行之间的较长的时
间间隔，长期调度程序能使用更多的时间来选择执行进程。4.中期调度程序：也成为交换，将进程移除内存，因此降低了多道程序设计的程度，之后进程能被重新调用并从中
所处继续执行。分时系统引入中期调度程序。5.短期调度程序：从准备可执行的进程中选择进程并为之分配 CPU。6.区别：选择进程的位置不一样：从执行频率上看短期频率
最高中期次之长期最低。7 上下文切换：将 CPU 切换到另一个进程需要保护当前进程的状态并恢复另一个进程的状态。8 进程上下文用进程的 PCB 表示，它包括寄存器的值、
进程状态和内存管理信息等。通常通过执行一个状态保存来保存 CPU 的当前状态（不管他是内核模式还是用户模式），之后执行一个状态恢复重新开始运行。3.3 进程操作 1.
进程在执行过程中，通过创建进程系统调用创建多个新进程。创建进程成为父进程，而新进程称为子进程。2.进程需要一定的资源（如 CPU 时间、内存、文件、I/O 设备）来



完成任务，在一个进程创建子进程时，子进程可能从操作系统那里直接获得资源，也可能只从其父进程那里获得资源。父进程可能必须在子
进程中分配或共享资源，限制子进程只能使用父进程的资源能防止创建过多的进程带来的系统超载。3.通过 fork 系统调用，创建一个新进
程：对于新进程，系统调用 fork 返回值为 0，而对于父进程，返回值为子进程的进程标识符。4.当进程执行最后的语句并使用系统调用 exit
（）请求操作系统删除自身时，系统终止。这是进程可以返回状态值（通常整数）到父进程（通过系统调用 wait（））所有进程资源被操
作系统释放。5.当进程创建新进程时，有两种执行可能：（1）父进程与子进程并发执行（2）父进程等待，直到某个或全部子进程执行完
6.新进程的地址空间也有两种可能：（1）子进程是父进程的复制品（具有与父进程相同的程序和数据）（2）子进程装入另一个新程序 7.
父进程终止进程的原因有很多：子进程使用了超多它所分配到的一些资源（为判定是否发生这种情况，要求父进程有一个检查其子进程状态
的机制）；分配给子进程的任务已经不再需要；父进程退出，如果父进程终止，那么操作系统不允许子进程继续。3.4 进程间通信（IPC）
1.两种基本模式：共享内存（建立起一块供协作进程共享的内存区域，进程通过此此共享区读或写入数据来交换信息）和消息传递（通过在协
作进程间交换信息来实现通信，直接通信，间接通信，通过邮箱接受发送消息，缓冲队列，阻塞非阻塞）2.允许进程协作的理由：信息共享；
提高运算速度；模块化；方便。3.消息传递对于交换较少数量的数据很有用，它通常用系统调用来实现，且更易实现；共享内存允许以较快
的速度进行通信，比消息传递更快。

第四章 线程

.线程是 CPU 使用的基本单元，它由线程 ID，程序计数器，寄存器集合和栈组成。它与属于同一进程的其他线程共享代码段、数据段和其他操作资源，如打开文件和信号。
.多线程编程的优点：响应度高，资源共享，经济，多处理器体系结构的利用。3.两种提供线程支持的方法：用户线程（受内核支持，无需内核管理）和内核线程（由操作系
统直接支持和管理）4.多对一模型：许多用户级线程映射到一个内核线程。线程管理是由线程库在用户空间进行的，因而效率较高。但是如果一个线程执行了阻塞系统调用，
那么整个进程会阻塞。而且，因为任一时刻只有一个线程能访问内核，多个线程不能并行运行在多处理器上。5 一对一模型：将每个用户线程直接映射到一个内核线程。该模
型在一个线程执行阻塞系统调用时，能允许另一个线程继续执行，所以它提供了更好的并发功能；它也允许多个线程并行在多处理器系统上。缺点是每创建一个用户线程需要
创建一个内核线程，大量的开销影响性能，限制了线程数量。6.多对多模型：开发人员可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，而且当一
个线程执行阻塞系统调用时，内核能调度另一个线程来执行。7.三种线程库：（1）POSIX Pthread（2）Win32（3）JAVA 8.线程与进程的比较：（1）调度：在传统 OS 中，调
度和分配的基本单位是进程，用于资源的基本单位也是进程。引入线程的 OS 中，调度和分配的基本单位是线程，而拥有资源的基本单位是进程。（2）并发性：在引入线程的
系统中，进程之间可以并大执行，同一进程的多个线程也可以并发执行，使系统具有更好的并发性，进一步提高了资源利用率及系统吞吐量（3）拥有资源：进程是拥有资源
的基本单位，线程本身不具有资源，只拥有一定的必不可少的资源，可以访问其隶属进程的资源，可供同一进程的资源共享（4）系统开销：系统创建及撤销进程时的开销远
大于线程，切换也是。9.为什么要引入进程和线程：（1）进程：使多个程序并发执行，改善资源利用率，提高系统的吞吐量（2）线程：减少程序并发执行时所付出的系统开
销，具有更好的并发性

第五章 CPU 调度

1.CPU 调度依赖进程的如下属性：进程执行由 CPU 执行和 IO 等待周期组成，进程在这两个状态之间切换。2.CPU 调度决策可在如下 4 种环境下发生：（1）当一个进程从运行
状态切换到等待状态（例如，IO 请求，或调用 wait 等待一个子进程的终止）（2）当一个进程从运行状态切换到就绪状态（例如，当出现中断时）（3）当一个进程从等待状
态切换到就绪状态（例如 IO 完成）（4）当一个进程终止时 3.对于 1、4 两种情况没有选择只有调度，1 个新进程（如果就绪队列已有一个进程存在）必须被执行，不过对于
2、3 两种情况，可以进行选择。当调度只发生在 1 和 4 责成调度方案是非抢占的，否则是抢占的。
.分派程序是一个模块，用来将 CPU 的控制交给由短期调度程序选择的进程。其功能包括：切换到上下文、切换到用户模式、跳转到用户程序的合适位置，以重启启动程序
.调度准则：CPU 使用率（使 CPU 尽可能忙）、吞吐量（指一个时间单位内完成进程的数量）、周转时间（从进程提交到进程完成的时间间隔）、等待时间（就绪队列中等待
所花时间之和）、响应时间（从提交请求到产生第一相应的时间）5.调度算法：FCFS（优：绝对公平，简单自然，非占先式调度。缺：平均等待时间最长，护航效果，CPU 和
设备利用率低）、SJF（优：系统吞吐量大，平均等待时间短，有利于短作业。缺：不利于长作业，饿死，采用预测，不适合 CPU 短期调度）、优先权（优：干预 CPU 调度 缺：
饥饿 解决：增加等待时间很长的进程）、轮转发（优:有利于分时系统的实现，提高并发性，缩短等待时间 缺：对时间片长度需要仔细确定：过长类似 FCFS，多段需要大量
上下文切换导致 CPU 产生大量空闲时间）、多级队列调度（将就绪队列分成多个独立队列，根据进程的某些属性，如内存大小，进程优先级或进程类型，进程会被永久分配到
一个队列，每个队列都有自己的算法。优：低调度开销 缺：不够灵活）、多级反馈队列调度、响应比最高者优先（响应比=等待时间/运行时间+1）。

第六章 进程同步

.进入区、临界区、退出区、剩余区。没有两个进程可以在临界区内执行。1.1 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发生的特定顺序有关 2.1 不能
同时被并发执行的进程操作的资源被称为临界资源 2.临界区满足的要求：互斥（如果进程 Pi 在临界区内执行，那么其他进程不准进入）、前进（如果没有进程在其临界区内
执行且有进程需要进入临界区，那么只有那些不在剩余区内执行的进程可参加选择，以确定谁能下一个进入临界区，且这种选择不能无线推迟）、有限等待（从一个进程做出
进入临界区的请求到该请求允许为止，其他进程允许进入临界区的次数有上限）。让权等待。3.抢占内核（更适合编程，响应更快，处于内核模式的进程不会运行太久）与非
抢占内核。是否允许内核模式的进程被抢占。4.解决临界区问题的硬件方法：（1）对于单处理器环境：在修改共享变量时关中断，通常为非抢占内核所用。（2）对多处理器
环境使用特殊硬件指令（原子的，不可中断的指令）
.为什么关中断不可行？这是特权指令不能随意在用户态执行（1）处理器间的消息传递导致进入每个临界区都会延迟，进而降低系统效率（2）影响了系统时钟 6.boolean

```
estAndSet(boolean *target){                    do{
  Boolean rv =*target;                             while(TestAndSet(&Lock)) ;//do nothing
    *target =TRUE;                                 //临界区
    Return rv;}                                     lock=FALSE;
                                                    // 剩余区 } while （TRUE）;
. void swap (boolean *a, boolean *b){              do {
   Boolean temp= *a;                                 key=TRUE;
   *a=*b;                                           while(key=TRUE)
   *b=temp; }                                       swap(&lock,&key)    ;
                                                    //临界区
                                                    Lock=FALSE;
                                                   //剩余区 }  while （TRUE）;
```

. 所有硬件方法都不能实现让权等待。9.原语：若干指令构成的不可分割的程序（用关中断的方法实现）10.事务：执行单个逻辑功能的一组指令或操作 原子性：要么不做
要么都做 原语：不允许中途做别的事情 10.信号量：wait（S）{while（S<=0）； s- -;} signal（S）{S++;} wait（semaphore *S）{S->value- -; if （S->value<0）
add this process to S->list; block();}} signal（semaphore *S）{S->value++; if （S->value<=0）{remove this process to S->list; wakeuo(P);}}

第七章 死锁

程在指令重内无限等待。2.在正常模式下，进程按如下顺序使用资源：申请、使用、释放 3.死锁产生的必要条件：互斥、占有并等待、非抢占、循环等待 4.死锁处理方法：（1）使用协议以预防或避免死锁（2）允许系统进入死锁状态，然后检测它并加以恢复（3）可忽视这个问题，认为死锁没有发生 5.死锁预防：破坏四个特征 6.死锁避免：根据每个进程可能申请的各种资源类型实力的最大需求的实现信息可以构造一个算法以确保系统不会进入死锁状态。7.银行家算法：当新进程进入系统时，他必须说明其可能需要的每种资源类型实例的最大数量，这一数量不能超过系统资源的总量，当用户申请一组资源时，系统必须确保这些资源的分配是否仍使系统出于安全状态，如果会，分配资源，否则必须等待其他进程释放资源。

第八、九章 存储管理

1.存储管理（1）存储管理的功能：a.存储空间的分配和回收 b.地址映射和重定位 c.存储共享和保护 d.主存扩充（2）存储分配的三种方式 a.直接存储分配方式：在程序设计过程中，或汇编程序对源程序进行编译时，所用的是实际物理地址，以确保各程序所用的地址之间互不重叠；b.静态存储分配方式：编写程序或由编译系统产生的目标程序中采用的地址空间为逻辑地址，当连接装入程序时对它们进行装入，连接时，才确定它们在主存中的相应位置，从而产生可执行程序，这种分配方式要求用户在进行装入，连接时，系统必须分配其要求的全部存储空间，若存储空间不够，则不能装入该用户程序，同时，用户程序一旦装入到主存空间后，它将一直占据着分配给它的存储空间，直到程序结束时才释放该空间，再者，在整个运行过程中，用户程序所占据的存储空间是固定不变的，也不能动态地申请存储空间；c.动态存储分配方式：用户程序在存储空间中的位置也是在装入时确定，但它不必一次性将整个程序装入到主存，可根据执行的需要，一部分一部分地动态装入，同时，装入主存的程序不在执行时，系统可以回收该程序所占据的主存空间，再者，用户程序装入主存后的位置，在运行期间可根据系统需要而发生改变，此外，用户程序在运行期间也可动态地申请存储空间以满足程序需求；

2.重定位的定义、两种重定位的特点与区别、覆盖与交换（1）重定义定义：由于用户程序的装入而引起地址空间中的相对地址转换为存储空间中的绝对地址的地址变换过程，称为地址重定位，也称地址映射；（2）实现地址重定位的方法：静态地址重定位，动态地址重定位 a.静态地址重定位：用户程序在装入时由装配程序一次完成，即地址变换只是在装入时一次完成，以后不再改变；优点：实现简单；缺点：用户程序必须分配一个连续的存储空间；难以实现程序和数据的共享；b.动态地址重定位：在程序执行的过程中，当 CPU 要对存储器进行访问时，通过硬件地址变换机构（重定位寄存器 BR 和相对地址寄存器 VR），将要访问的程序和数据地址转换成主存地址；优点：有利于提高主存的利用率和存储空间使用的灵活性；有利于程序段的共享实现；为实现虚拟存储器管理提供了基础；缺点：实现存储器管理的软件比较复杂；需要附加的硬件支持；

（3）覆盖与交换（从逻辑上扩充主存,解决在较小主存空间中如何执行大程序的问题）

.覆盖：把程序划分为若干个功能相互独立的程序段，并且让那些不会同时被 CPU 执行的程序段共享同一主存区，通常这些程序段被保存在外存中，当 CPU 要求某一程序段执行时，才将该程序段装入主存来覆盖以前的某一程序段；b.交换：将系统暂时不用的程序或数据部分部分或全部地从主存中调出，以腾出更大的存储空间，同时将系统要求使用的程序和数据调入主存中，并将控制权转交给它，让其在系统上运行；c.交换技术主要是在进程或作业间进行，覆盖技术则主要是在同一个进程或作业之间进行，交换技术的运用，可以在较小的存储空间中运行较多的作业或进程，覆盖技术的运用，可以在较小的存储空间中运行比其容量大的作业或进程；

3.分区存储管理、页式存储管理(各种方法采用的分配回收算法，数据结构，地址变换过程，共享与保护，优缺点比较)

（1）分区存储管理：将主存的用户可用区域划分成若干大小不等的区域，每一个进程占据一个区域或多个区域，从而实现多道程序设计环境下各并发进程共享主存空间；

固定分区法：系统在初始化时，将主存空间划分为若干个固定大小的区域，用户程序在执行过程中，不允许改变划分区域的大小，只能够根据各自的要求，由系统分配一个存储区域；数据结构：分区说明表

动态分区法：采用将主存的空闲区单独构成一个可用分区表或可用分区自由链表的形式来描述系统主存管理；(P94 习题 3.6)

①分配方法：a.最先适应法：将作业分配到主存的第一个足够装入它的可用空闲区中；b.最佳适应法：将作业分配到主存中与它所需大小最接近的一个可用空闲区中；（要求分区表或自由链接表按照空闲区从小到大的次序排列）c.最坏适应法：将作业分配到主存中最大的空闲区中；（要求分区表或自由链接表按照空闲区从大到小的次序排列）

②回收方法：a.释放区与上下两个空闲区相邻，在这种情况下，将三个空闲区合并为一个空闲区；b.释放区与上空闲区相邻，在这种情况下，将释放区与上空闲区合并为一个空闲区；c.释放区与下空闲区相邻，在这种情况下，将释放区与下空闲区合并为一个空闲区；d.释放区与上下两个空闲区都不相邻，在这种情况下，释放区作为一个新的空闲区用区插入到可用分区表或自由链表中；

③数据结构：可用分区表或可用分区自由链表；

④地址变换过程：采用动态重定位装入作业，当作业**执行时由硬件地址转换机构完成地址转换（基址寄存器，限长寄存器）；**

⑤分区共享：各道作业的共享存储区域部分有相同的基址/限长值，就可实现分区共享；

⑥分区保护：对共享区的信息规定只能执行或读出，不能写入；

⑦分区存储管理的优缺点：a.优点：实现了多道程序的设计，从而提高了系统资源的利用率；系统要求的硬件支持少，管理简单，实现容易；b.缺点：由于作业在装入时的连续性，导致主存利用率不高；主存的扩充只能采用覆盖和交换技术，无法真正实现存储器；

（2）页式存储管理：页式存储器管理取消了存储分配的连续性，它能够将用户进程分配到不连续的存储单元中连续执行；（根据作业装入主存的时机不同，一般分为：1，静态分页管理 2，虚拟分页管理）

①分页存储器的逻辑地址格式：　页号　　单元号

②分配的考虑：将进程的页分配到主存的块中。

③静态分页管理：用户作业在开始执行以前，将该作业的程序和数据全部装入到主存中，然后，操作系统通过页表和硬件地址变换机构实现逻辑地址到物理地址的转换，从而执行用户程序；

④分配回收算法：依据存储页框表，请求表和页表实现；

⑤地址变换：首先用户作业提出存储分配的要求，此时操作系统根据主存页框的大小将进程要求的存储空间分成相应的页面；根据主存的实际情况，将进程的每个页面分配到主存页框中，　系统分配并设置页表的内容，此时，系统完成用户进程的存储器分配；当用户进程开始执行时，系统首先设置控制寄存器的内容，控制寄存器包括页表长度和页表起始地址两项；为了对逻辑地址进行变换，由硬件组成的地址变换机构必须将其分成两部分—页号和页内偏移；根据逻辑地址中页号在页表中找到相应的页框号；将页表中的页框号和逻辑地址中的页内偏移分别写入绝对地址中的相应位置上；然后根绝绝对地址提供的页框号和页内偏移计算出存储空间的物理地址，用户进程可以访问主存中的绝对地址，取出数据或取出指令执行；③快表：存放在高速缓冲存储器中的页表（引入快表时为了减少访问主存的次数提高地址变换的速度）；

⑥加入快表后的地址转换：CPU 在给出逻辑地址后，地址变换机构首先根据页号在快表中进行检索，若存在相应的页号，则直接从快表中读出该页号对应的页框号，形成物理地址，否则需要访问主存中的页表，从页表中读出相应的页框号，形成相应的页框号，形成物理地址，同时将找到的页表登记到快表中，当块表填满后，又要在快表中登记一新的页表项时，则需要一定的淘汰策略；⑤数据结构：存储页框表，请求表和页表等⑥共享：能方便地实现多个作业共享程序和数据，页的共享可大大提高主存空间的利用率；在页式存储器中实现程序共享时，必须对共享程序给出相同的页号；　⑦保护：a.保护权限域　　b.保护键　⑧优点与缺点：优点：解决了分区管理时的碎片问题；　缺点：仍受主存中可用页框数的限制；

4.虚拟存储器基本思想，虚拟页式存储工作原理

进程执行的前提是进程的全部页都已经在内存中。虚拟内存技术就解决了这个问题，不需要一个进程把全部的页都放在内存才能执行。动态载入虽然也解决了这个问题，但是需要程序员完成，非常麻烦。其实我们发现，一个程序包含了很多条件语句还有异常处理等，这些代码肯定要选择执行的。所以全部调入会显得冗余，增加了磁盘到内存的传输时间。并且减少一个进程的空间可以使得内存容纳更多的进程，增加多道程序的度。

虚拟地址空间就是一个逻辑视图，通过 MMU 映射到物理地址空间。 稀地址空间：进程的堆和栈之间的空白的虚拟地址空间。进程虚拟地址空间内容：堆向上伸展，栈向下伸展，代码和数据。中间放共享库。 虚拟内存可以通过共享页将文件或内存共享给多个进程。每个进程都认为库位于自己的虚拟地址空间。

按需调页：一个进程分成多个页，在需要时调入相应的页，也称为懒惰交换。

按需调页对于性能的影响很大，因为会发生页错误，所以尽量减少页错误。

交换是将整个进程看成一个整体，通过交换程序进行交换，而懒惰交换是将一个进程看成多个页，通过调页程序进行交换。

可我们通过有一张页表维护有效-无效位来区分哪些页在内存，哪些页在磁盘。如果在内存且页号有效，则记为 v，如果在磁盘上，则为 i。

如果访问无效位的页，则通过页错误方式来解决即如果发现要访问的页不在内存，则：

1.终止进程，保存现场，并通知操作系统。2.寻找空闲帧，如果没找到，用到页面置换技术。3.将磁盘上的页调度到相应帧中。4.重新开始指令。

也粹按需调页：不断页错误，不断将页调入内存。

页错误发生也是有不同代价的，如果在取指时就发生，则只需要重新取指，如果在获得操作数时发生，则需要重新取指，译码，取操作数，代价就大了很多。

是如果 ADD A,B,A　A+B->A，当写入 A 时出错，重新执行这条指令，但是 A 已经被改变了，所以结果会不对。解决方案是用临时的寄存器保存原来的值。

有效访问时间：（1-p）*内存访问时间+p*页错误时间。

页错误的具体过程：1.告知操作系统。2.保存现场。3.确定页表的 i 代表页错误，因为 i 可能代表页不存在。4.磁盘读入空闲帧。5.修改页表和帧表。6.恢复。

页错误处理时间主要是：1.处理页错误。2.读入页。3.恢复。

磁盘专门有个交换空间，用于交换数据，处理速度要快。可以在交换空间执行按页调度。

写时复制：创建进程的开始可能不需要按需调页，子进程共享父进程的资源和地址空间。当任何一个进程写入时，就为另一个进程创建一个页面副本。fork

分配空闲页当作页面副本也有技巧，可以通过空闲缓冲池，分配时采用按需填零即只在分配之时清零，清除以前内容。

增加多道程序的度数会有“过度分配”的情况，当页错误发生，需要调入页面，但是内存中没有空闲帧，那么必须页置换。页置换是将页错误的时间加倍了，因为要换入和换出。

基本页置换：

按需调页中需要在内存中有一块空闲帧来存放调入的页，如果没有空闲帧，就要发生页置换。

页置换过程：挑一个牺牲帧，并更新帧表和页表。

可以用修改位或脏位降低开销，当修改位被设置，则意味着这个页从磁盘读入后已被修改，那么就需要写回，如果修改位没被修改，则不需写回。

引用串用来评估一个页置换算法的性能：内存引用的页号序列。如 1，2，3 表示引用页号 1，2，3。

当不考虑内存帧的数量的情况下，只有页的第一次引用发生页错误，但是如果帧的数量限制，则会复杂的多。

.FIFO 页置换：

将内存的页看成是一个 FIFO 队列，替换的页是最旧的页即最先调入内存的页，加入的页放到队列的尾部。这不是一个好办法，因为最先进入的页可能是一直要用到的页，如果是替换的是活动页，则此页会马上又一次页

错误，换回内存。

Belady 异常：页错误数随着内存帧数的增加而增加。

.最优置换 OPT

不会产生 Belady 异常，并且页错误率是最低的。因为他替换的是将来最久才被用到的页。

.LRU 置换

选择的页是最近最少使用的页。

对于引用串，如果倒转，LRU 置换和 OPT 置换的页错误率是一样的。

LRU 实现需要硬件支持：

.计数器。每个页帧会附带一个最近使用时间，置换出的页就是时间最小的。

.栈。栈顶加入页，栈尾写出页。

.近似 LRU 置换

因为 LRU 实现需要许多硬件支持，因此可能只能近似的实现 LRU。

通过添加引用位，每当一个页被引用，该为就置 1.通过检查该位，可以知道哪些页未被使用。

（1）附加引用位算法。

在一个页的引用位保留 8 位，初始为 0000 0000，第一次被使用，则 1000 0000，第二次未被使用，则 0100 0000.以此类推。找到这 8 位化为 10 进制后最小的页置换。

是在页中保留一个引用位，如果为1，则给予第二次机会，如果为0，则直接置换。
（3）增强型二次机会算法。
将引用位和修改位作为有序对，（0，0）表示未被引用和修改，则说明是最佳的 LRU 置换页。
.基于计数的页置换
（1）Least frequently used LFU：保留一个计数器，记录引用次数。缺点是以前一直用，但是现在不用，则仍会留在内存。
解决方法：定期将计数右移一位。
（2）MFU：最常使用页置换，即把计数最大的页换出。
.页缓冲算法
.应用程序：不需要提供虚拟内存。
.磁盘：没有文件系统的磁盘。
接下来讲帧分配,帧分配的最小数量是根据计算机体系结构，最大数量是根据物理内存数量。
分配方法：
.平均分配。2.比例分配。根据优先级和进程大小。
页分配和页置换是相关联的。
全局置换：一个进程从任意帧中选择一个置换，不管该帧是否已分配。 缺点是不能控制页错误率。
局部置换：仅从自己的分配帧中置换。
页颠簸：频繁调页。
页颠簸发生情景：
当 CPU 使用率降低时，CPU 导入更多进程，如果用全局置换，如果一个进程需要很多帧，因此发生页错误会从其他进程中拿一点，而如果其他进程也页错误，则会从其他进程中拿帧，因此进程都到进程的等待调页设备
队列，CPU 使用率降低，恶性循环。
采用局部置换也不能防止系统颠簸。因此我们必须要预先就分配足够多的帧。因此引入了局部模型。
局部模型是那些经常使用的帧，因此进程分配的帧一定要大于局部模型的帧大小。
.工作集合模型：working-set model 。基于局部性。
工作集合窗口：预设△是最近用到的页。对于△的取值很关键。
工作集合：在工作窗口中使用的页的集合。
如果总的可用帧数量为 W，进程 i 的工作集合为 WSi，则 W-WSi 为其余的可用帧，直到<0 就不分配。
如果在内存中的进程的工作集合之和大于可用帧数量，则系统颠簸。
.页错误频率 PFF。
预先设定页错误率的上下界。当页错误率大于上界，则为进程分配更多的帧。这样非常直接。
当对于一个新的局部进行按需调页时，页错误率达到波峰。
如果调用 open()，需要先找目录，再找文件，因此需要内存访问+磁盘访问。如果将文件 I/O 作为内存的普通访问，则称为文件的内存映射。
在内存中的文件可以修改就不立即写回。
前面讲到内存映射文件，下面是内存映射 I/O，一组内存专门映射到设备寄存器，读取该块地址如同读取设备寄存器。
内核内存的分配与用户进程的分配内存不同，用户进程都是按页分配，但是内核分配需要从空闲内存池中获取，而不是通过空闲帧链表中获取。

分页存储管理优缺点：优点 1）解决主存的零头问题，能有效地利用主存。2）方便多道程序设计，并且程序运行的道数增加了。3）可提供大容量的虚拟存储器，作业的地址空间不再受实际主存大小的限制。4）更加方便了用户，特别是大作业的用户。当某作业地址空间超过主存空间时，用户也无需考虑覆盖结构。
缺点：1）要有相应的硬件支持，如需要动态地址变换机构、缺页中断处理机构等。2）必须提供相应的数据结构来管理存储器，它们不仅占用了部分主存空间，同时还要花费 CPU 时间。3）在分页系统中页内的零头问题仍然存在。4）在请求分页管理中，需要进行缺页中断处理，还有可能出现抖动现象，增加了系统开销，降低系统效率。
分页式缺点（除上述的缺点外还有）：
）程序的逻辑地址空间是连续的 ，装配好的程序段和数据块的存储空间是确定的，在执行中是无法动态增长和收缩。
）无法做到页与逻辑意义完整的子程序或数据段的唯一对应，增大了其信息共享实现的难度。
）从连接的角度上看，分区管理和分页管理只能采用静态连接 ，不仅花费了大量的 CPU 时间，而且也浪费了许多主存空间 。
☆5.常用的页面置换算法 （1）FIFO 页置换 置换掉最旧的页 优点：容易理解和实现 缺点：其性能不总是很好 Bleady 异常：页错误率可能会随分配的帧数增加而增加（2）最优置换算法：置换掉使用次数最少的 优点：是所有算法中产生也错误率最低的且不存在 BLEADY 异常 缺点：难以实现 （3）LRU（置换最长时间没有使用的页）优点：相对于最优页置换算法置换容易实现且效果也不错，优于 FIFO 缺点：需要大量硬件支持
.段式存储管理的思想，段式虚拟存储管理流程
（1）段式存储管理的思想：把程序按逻辑含义或过程分成段，每段都有自己的段名，用户程序可用段名和入指出调用一个段的功能，程序在编译或汇编时，再将段名定义一个段号，每段逻辑地址均是以 0 开始进行顺序编址，这样用户或进程的地址空间就形成了一个二维线性地址空间，任意一个地址必须首先指出段号，其次指出段内偏移地址，段式存储管理程序以段为单位分配主存，然后，执行时通过地址转换机构把段式逻辑地址转换成主存物理地址；
在段式存储器中实现程序共享时，共享段的段号不一定要相同；
逻辑地址格式： 段号 单元号
段式管理信息共享和保护 ：共享：只要用户使用相同的共享段名，系统在建立段表时，只须在相应的段表栏目上填入已在主存的段的始址和长度，即可实现程序和数据段的共享，从而提高系统主存的利用率。保护：（1）在段表中增设一个存取权限域。存取权限可分为：只执行（共享程序段）、只读（共享数据段）和可读/写（私人段）。访问段时，核对存取权限。（2）在地址转换时，将段表中的长度与段内地址比较，实现地址越界保护。
分段与分页比较
分段是信息的逻辑单位，是用户可见的，段的大小是用户程序决定。而分页是信息的物理单位，分页对用户来说是不可见的，页的大小是事先固定的。