1.2 操作系统的发展过程 操作系统的定义: 是一组能有效组织和管理计算机硬件和软件资源，合理对各类作业进行调度，以及方便用户使用的程序的集合。 1.2.1 未配置操作系统的计算机系统 1、人工操作方式[缺点:用户独占全机，CPU 等待人工 IO 操作]，严重降低了计算机资源的利用率。2、脱机输入/输出(Off-Line I/O)方式，程序和数据的输入实在外围机(脱离主机)的控制下完成，[优点:减少了 CPU 的空闲时间(IO 操作不占用主机时间)，提高了 IO 速度(数据直接从高速磁带上输入到内存中)] 1.2.2 单道批处理系统 先把一批作业以脱机方式输入到磁带上，在系统监督程序的控制下，使作业能够一个接着一个的连续处理。批处理系统旨在提高系统资源的利用率和系统吞吐量。[缺点:资源得不到充分的利用(当程序发出 IO 请求后，CPU 必须等待)] 1.2.3 多道批处理系统 利用作业调度算法，当一程序因 IO 操作而使 CPU 空闲时，调用执行另个程序。[优点:资源利用率高，系统吞吐量大。缺点:平均周转时间长，无交互能力] 多道批处理系统需要解决的问题有：1 处理机争用问题，2 内存分配和保护问题，3IO 设备分配问题，4 文件的组织和管理问题，5 作业管理问题，6 用户与系统的借口问题。1.2.4 分时系统 指在一台主机上连接了多个配有显示器和键盘的终端并由此所组成的系统，该系统允许多个用户通过自己的终端，以交互方式使用和计算机，共享主机中的资源。及时接受，及时处理。[特性:多路性，独立性，及时性，交互性] 1.2.5 实时系统 指系统能够及时响应外部事件的请求，在规定时间内完成对该事件的处理，并控制所有实时人物协调一致地运行。[实时系统的类型]：工业控制系统，信息查询系统，多媒体系统，嵌入式系统。1.2.6 微机操作系统的发展 1、单用户单任务操作系统:只允许一个用户上机，且只允许用户程序作为一个任务运行。2、单用户多任务操作系统:只允许一个用户上机，但允许用户把程序分为若干个任务，并发执行，有效的改善系统的性能。3、多用户多任务操作系统:允许多个用户通过各自的终端，使用同一台机器，共享主机系统中的各种资源。每个用户程序又可以进一步分为几个任务，并发执行，进一步提高资源利用率和系统吞吐量。

1.3 操作系统的基本特征 **1.3.1 并发与并行** 并行指两个或多个事件在同一时刻发生，并发性指两个或多个事件在同一时间间隔内发生。1.3.2 进程 指在系统中能够独立运行并作为资源分配的基本单位，由一组机器指令，数据和堆栈等组成，是一个活动实体。多个进程之间可以并发执行和交换信息。能够极大的提高系统资源的利用率和增加系统吞吐量。

2.1.3 程序并发执行 只有在不存在前趋关系的程序之间才有可能并发执行。2.程序并发执行的特征:间断性，失去封闭性，不可再现性。程序并发执行时由于共享系统资源和为完成同一项任务而相互合作，致使并发的程序之间形成了相互制约的关系。程序并发执行时，系统中共享的资源状态有并发程序来改变，使任一程序在运行时，其环境都会受到其他程序的影响而失去封闭性。程序失去了封闭性，也将导致失去可再现性。

2.2.1 进程的定义与特征 定义：为了使程序能够并发运行并保持独立性，对并发的程序加以描述和控制，引入进程。操作系统为保证并发程序的独立性，设置了一个专门的数据结构：进程控制块 PCB。利用 PCB 来描述进程的基本情况和活动过程，进而控制和管理进程。创建撤销进程，就是创建撤销进程实体中的 PCB。 进程是进程实体的运行过程，是系统进行资源分配和调度的独立单位。进程的特征 动态性，并发性，独立性，异步性。进程同步机制，当进程在并发运行具有异步性时，保证进程并发执行的结果是可再现的。进程的三种基本状态: 就绪状态，执行状态，阻塞状态。另外有创建状态和终止状态。引入创建状态是为了保证进程的调度必须在创建工作完成后进行，确保对 PCB 操作的完整性，也增加了管理的灵活性。挂起操作和进程状态的转换:引入挂起操作的原因：终端用户的需要，父进程请求，对换需要，负荷调节的需要，操作系统的需要。处于挂起状态的进程不能接收处理机调度。 状态的转换：1.活动就绪->静止就绪，2.活动阻塞->静止阻塞，3.静止就绪->活动就绪，4.静止阻塞->活动阻塞。

2.2.4 进程控制块 PCB 的作用 PCB 是操作系统中最重要的记录型数据结构。1.作为独立运行基本单位的标志，PCB 已成为进程存在于系统中的唯一标志。2.实现间断性运行方式。3.提供进程管理所需要的信息。4.提供进程调度所需要的信息。5.实现与其他进程的同步与通信。进程控制块中的信息：主要包括 1.进程标志符，2.处理机状态，3.进程调度信息，4.进程控制信息。处理机状态信息主要由处理机的各种寄存器中的内容组成，包括 1.通用寄存器，2.指令计数器，3.程序状态字，4 用户栈指正。当进程被切换时，处理机状态信息必须保存在相应的 PCB 中，以便在进程重新执行时能够从断点继续。进程控制块的组织方式: 1.线性方式，将系统中所有的 PCB 都组织在一张线性表中。2.链接方式，把具有相同状态的 PCB 分别通过 PCB 中的链接字链接成一个队列。如就绪队列，阻塞队列和空白队列。3.索引方式，根据进程状态的不同，建立几张索引表，如就绪索引表和阻塞索引表。

2.4.1 进程同步的基本概念 进程同步机制的主要任务是对多个相关进程在执行次序上进行协调，使并发执行的诸进程之间能够按照一定的规则共享系统资源，并能很好的相互合作，从而使程序的执行具有可再现性。1。两种形式的制约关系： 1).间接相互制约关系，进程之间因为临界资源的使用而相互制约。因此系统的资源必须有系统实施统一分配，用户先申请才能使用资源。2).直接相互制约，进程之间有严格的执行次序的制约。2.临界资源： 不同进程之间所共享的数据，变量，资源，涉及使用和修改时，都需要作为临界资源互斥访问。3.临界区： 访问临界资源的代码称为临界区。4.进程同步应遵循的规则： 1).空闲让出，临界资源空闲时需要立即让请求的进程使用该资源。2).忙则等待，临界资源被使用时，其他进程需要等待资源使用完毕再使用。3).有限等待，保证进程能在有限的时间内进入临界区，以免进入忙等。4).让权等待，进程不能进入临界区时，让出处理机，以免进入忙等状态。

2.4.2 硬件同步机制 1.关中断 2.利用 Test-and-Set 指令 3.利用 Swap 指令 硬件指令能够有效的实现进程互斥，但是容易让进城进入忙等状态，也很难解决复杂的进程同步问题。

2.4.3 信号量机制 1.整型信号量： 一个用于表示资源数目的整型量 S，仅能通过两个标准的院子操作 wait(S)和 signal(S)来访问。整型信号量未遵循让权等待原则 2.记录型信号量：是一种不存在忙等的进程同步机制。在信号量机制中，还增加了一个进程链表指针 list，用于链接所有等待进程。3.AND 型信号量： 将进程在整个运行过程中需要的资源一次性分配，结束后一次性释放。4.信号量集： 对 AND 信号量进行扩充，对进程所申请的所有资源以及每类资源不同的资源需求量，再一次 PV 原语操作中完成申请和释放。信号量集的特殊情况：1). Swait(S,d,d),信号量集中只有一个信号量 S，允许每次申请 d 个资源，当现有资源少于 d 时，不予分配。2).Swait(1,1),蜕变为记录型信号量或互斥量。3).Swait(s,1,0)，当 S》=1 时，允许多个进程进入临界区，当 S 变为 0 时，阻止任何进程进入临界区。

2.4.5 管程机制 管程是对进程共享资源的申请，释放和其他操作的封装。代表共享资源的数据结构及有对该共享数据结构操作的一组过程组成的资源管理程序构成了一个操作系统的资源管理模块，管程。管程的主要特性：1).模块化，2).抽象数据类型，3).信息掩蔽 2.条件变量： 当一个进程调用了管程，在管程中时被阻塞或挂起，直到解除为止的期间内，如果不释放管程，其他进程便无法进入管程，被迫长时间等待。因此引入条件变量，指引起进程阻塞和挂起的条件的变量。x.wait 是将因某个条件阻塞或挂起的进程送入等待队列，并释放管程，x.signal 是将因某个条件送入等待队列的进程激活。

3.1.1 处理机调度的层次 1.高级调度：调度对象是作业，根据某种算法将外村上处于后备队列中的哪几个作业调入内存，创建进程，分配必要的资源，放入就绪队列。多用于多道批处理系统中。2.低级调度：调度对象是进程，根据某种算法，保存处理机的现场信息，并决定就绪队列中的哪个进程获得处理机。进程调度是一种最基本的调度。3.中级调度: 又为内存调度，对象是进程。主要目的是提高内存利用率和系统吞吐量，把暂时不能运行的进程调至外存等待，进程状态变为挂起状态。当运行条件具备和内存空余时，再把进程调度到内存中的就绪队列，变为就绪状态。

3.1.2 处理机调度算法的目标 共同目标: 1.提高资源利用率，使处理机和其他所有资源尽可能的保持忙碌状态。2.公平性，每个进程都获得合理的 CPU 时间，不发生进程饥饿现象。3.平衡性，尽可能保持系统资源使用的平衡性。4.策略强制执行。批处理系统的目标: 1.平均周转时间短，作业被提交给系统开始到作业完成为止的时间间隔短。应使作业周转时间和平均周转时间都尽可能短。2.系统吞吐量高，指单位时间内系统所完成的作业数，与处理的作业从平均长度有关。3.处理机使用率高。分时系统的目标: 1.响应时间快，从用户提交一个请求开始到显示出处理结果为止。2.均衡性好，指系统响应时间的快慢与用户请求服务的复杂性相适应。实时系统的目标: 1.截止时间短，2.可预测性。

3.2.3 先来先服务 FCFS 和短作业优先 SJF 调度算法 FCFS:系统按照作业到达的先后次序进行调度。主要用于与其他调度算法结合，形成一种更为有效的调度算法，如可以把进程按优先级设置多个队列，每个队列采用 FCFS。SJF:作业时间短，优先级高。[缺点：]1.必须预知作业的运行时间。2.对长作业非常不利。3.无法时间人机交互。4.不能保证紧迫作业及时执行。

3.2.4 优先级调度算法 PSA 和高响应比优先调度算法 HRRN PSA：根据作业或进程的紧迫程度设置的优先级进行调度。HRRN：是既考虑了作业的等待时间，又考虑了运行时间的调度算法[优点:]既照顾了短作业，又不使长作业的等待时间过长，改善了处理机调度的性能。优先权=(等待时间+要求服务时间)/要求服务时间=响应时间/要求服务时间。1.如果作业等待时间相同，则要求服务时间越短，优先权越高，有利于短作业。2.当要求服务时间相同，优先权取决于响应时间。3.对于长作业，优先级随着等待时间的增加而提高。[缺点:]调度前需要做响应比计算，增加系统开销。

3.3.1.1 进程调度的任务 1.保存处理机的现场信息。2.按某种算法选取进程。3.把处理器 分配给进程。**3.3.1.2 进程调度机制** 1.排队器，事先将系统中的所有就绪进程按照一定的策略排成一个或多个队列。2.分派器，将处理机分配个新选出的进程。3.上下文切换器，分派处理器时，新选进程的 CPU 现场信息装入到处理器的各个寄存器中；阻塞进程时，将处理器寄存器中的现场信息保存到进程 PCB 中。**3.3.1.3 进程调度方式** 1.非抢占方式：把处理机分配给进程后，只有当进程运行结束或者阻塞时，才将处理机分配给其他进程。[优点:]实现简单，系统开销小。适用于大多数批处理系统。2.抢占方式，允许调度程序根据某种原则，去暂停某个正在执行的进程，将处理机分配个另一个进程。抢占原则 1.优先权原则，2.短进程优先原则，3.时间片原则。

3.3.2 转轮调度算法 基本原理：基于时间片的调度算法，让就绪队列上的每个进程每次仅运行一个时间片，保证就绪队列中的所有进程在确定的时间内，都能获得一个时间片的处理机时间。进程切换时机：1.时间片未用完，进程已经结束。2.时间片已用完。时间片大小的确定：一个较为可取的时间片是略大于一个典型交互所需要的时间，使大多数交互进程能在一个时间片内完成，从而获得很小的响应时间。

3.3.3 优先级调度算法 1.非抢占式优先级调度算法。2.抢占式优先级调度算法。优先级类型：1.静态优先级，在创建进程时确定，整个运行期间不会改变。2.动态优先级，进程创建初期赋予一个优先级，随着进程的推进或等待时间的增加而改变，以便获得更好的调度性能。

3.3.4 多队列调度算法 将不同类型或性质的进程固定分配在不同的就绪队列中，不同的就绪队列采用不同的调度算法。

3.3.5 多级反馈队列调度算法 事先不需要知道各种进程所需的执行时间，还可以较好的满足各类进程的需求。调度机制：1.设置多个队列，为每个队列赋予不同的优先级。优先级越高，时间片越小。2.每个队列都采用 FCFS 算法。3.按队列优先级调度。调度程序首先调度最高优先级队列中的进程运行，第一队列空闲时才调度第二队列。调度算法的性能：如果规定第一个队列的时间片略大于多数人机交互的处理时间，便能较好满足各类用户的需求。终端用户作业在第一队列时间片中完成，短批处理用户的周转时间较短，长批处理用户不用担心长期得不到处理。

3.3.6 基于公平原则的调度算法 保证调度算法 不是保证优先运行，而是明确的性能保证，每个相同类型的进城获得相同相同的运行时间。公平分享调度算法 调度的公平性针对用户，使所有用户能获得相同的处理机时间，或所要求的时间比例。调度是以进程为基本单位的，必须考虑到每个用户的进程数。

3.5.2 计算机系统中的死锁 死锁是源于多个进程对资源的争夺，对不可抢占资源争夺，对可消耗资源的争夺和进程推进顺序不当时，会引起死锁。

3.5.3 死锁的定义，必要条件和处理方法 死锁的定义：如果一组进程中的每个进程都在等待仅由该组进程中的其他进程才能引发的事件，那么该组进程时死锁的。死锁的必要条件：1.互斥条件，在一段时间内，某资源只能被一个进程占用。2.请求和保持条件，进程已经保持了一些资源，又请求新的资源，却被其他进程占用，此时进程被阻塞。3.不可占用条件，进程已获得的资源在未使用完之前不能被抢占。4.循环等待条件,发生死锁时必然存在一个进程-资源的循环链，即每个进程都在等待另一个进程的资源释放。处理死锁的方法：1.预防死锁，通过设置限定条件，破坏产生死锁四个必要条件中的一个或几个。易实现。2.避免死锁，在资源的动态分配中，用某种方法防止系统进入不安全状态，从而避免死锁的发生，例如使用银行家算法。3.检测死锁，事先不采取任何措施，通过检测机构及时的检测出死锁的发生，然后采取措施解脱死锁。4.解除死锁，当检测到系统已发生死锁，撤销一些进程，回收资源，并将资源分配各阻塞状态中的进程，解除死锁。

3.6 死锁的预防 破坏请求和保持条件： 1.一次性地申请进程在整个运行过程中所需要的全部资源，进程在整个运行期间，不会再提出资源请求。[缺点:1.资源被严重浪费，严重恶化了资源的利用率。2.使进程经常发生饥饿现象。]，2.允许一个进程获得运行初期所需资源化后开始运行。在运行过程中初步释放自己使用完毕的资源，申请新的资源。能使进程更快的完成任务，提高设备的利用率，减少进程发生饥饿的几率。破坏不可抢占条件： 当一个已经保持了某些不可被抢占资源的进程，提出新的资源请求而不能满足时，必须释放自己所有的资源，以后需要时再申请。[缺点:可能使进程无限推迟执行，延长了进程的周转时间，增加系统开销，降低吞吐量。] 破坏循环条件： 对系统所有资源进行线性编号，规定每个进程都按照资源编号顺序请求资源。进程必须释放所有具有相同或者更高序号的资源后才能申请较低序号的资源。[缺点:各类资源的序号必须相对稳定，限制了新设备的增加。作业使用的资源顺序和资源编号不一致。限制用户自主，简单的编程。]

3.7 避免死锁 系统安全状态： 指系统能按照某种进程推进顺序为每个进程分配所需资源，直至满足每个进程对资源的最大需求，使每个进程都可顺利的完成，此时为一个安全资源分配序列。有安全序列则为安全状态。避免死锁的实质在于在进行资源分配时，是系统不进入不安全状态。

3.8 死锁的检测与解除 **死锁检测**：1.资源分配图，用圆圈代表一个进程，用方框代表一类资源，请求边是从进程指向资源，分配边是从资源指向进程。2.死锁定理：当且仅当某一状态时的资源分配图是不可简化的。简化的方法： 1).在资源分配图中找到不阻塞也不独立的进程结点 P1，正常情况下该结点是可以获得资源顺利执行完并释放所有资源的。去除该结点的请求边和分配边，使其孤立。2).P1 释放资源后，P2 才可以继续运行，消除 P2 的请求边和分配边，使其孤立。3).若能够消除所有进程结点的请求边和分配边，则该资源分配图是可简化的，反之不能简化，就会产生死锁。

3.死锁检测的数据结构： 类似于银行家算法的数据结构。

死锁解除： 解除死锁的方法有两种，1 是抢占资源，从一个或多个进程中抢占足够数量的资源，分配给死锁进程，以解除死锁状态。2 是终止进程，终止一个或多个死锁的进程，知道打破死锁环路。终止的方法有：1).终止所有死锁进程,2).逐个终止死锁进程。

终止进程时要考虑的因素：1).进程的优先级大小，2).进程执行的时间，还要执行的时间，3).进程使用的资源，还需要的多少资源，4).进程是交互式还是批处理式

代价最小的死锁解除算法： 从死锁状态 S 中先终止一个死锁进程 P1，使系统状态由 S 演变成 U1，将 P1 记入被终止进程的集合 d(T)中，并把所付出的代价 C1 加入到 Rc(T)中；对死锁进程 P2，P3 等重复上述过程。得到状态 U1，U2...Un 后，在按终止进程时话费的代缴大小，把它插入到由 S 状态所演变的新状态队列 L 中。队列 L 的第一个状态 U1 时有 S 状态花最小代价终止一个进程所演变成的状态。在终止一个进程后，若系统还是处于死锁状态，则再从 U1 状态按照上述处理方式在依次的终止一个进程，得到 U1',U2'...Uk'状态，再从 U'状态中选出一个代价最小的 Uj',如此下去直到死锁解除。把系统从死锁中解除出来所花费的代价表示：R(S)min=min{Cui}+min{Cuj}+min{Cuk}+...

4.2 进程的装入和链接 用户程序要在系统中运行，必须先装入内存，然后将其转变为一个可以执行的程序。1.编译，编译程序对源程序进行编译，形成若干个目标模块。2.链接(静态链接和动态链接)，链接程序将一组目标模块和他们所需的库函数链接在一起，形成一个完成的装入模块。3.装入，将装入模块装入内存。**4.2.1 程序装入：** 1.绝对装入方式，事先知道程序将驻留在内存的位置，装入模块装入内存后，程序中的逻辑地址与内存物理地址相同。2.可重定位装入方式，编译形成的目标模块，起始地址都是从 0 开始，程序中的其他地址都是相对于其实地址计算的。装入时对目标程序中指令和数据地址的修改成为重定向。地址变换实在程序装入时一次性完成，故称为静态重定向。3.动态运行时装入方式，指把模块装入内存后，不立即把装入模块中的逻辑地址转换为物理地址，而使把地址转换推迟到程序真正要执行时进行。装入内存后的所有地址都是逻辑地址。三种装入方式都适用于多道程序环境中。**4.2.2 程序链接：** 1.静态链接方式，程序在运行之前，先将各个目标模块及他们所需的库函数链接成一个完整的装配模块，以后不可拆开。[需要解决的问题：1.修改模块间的相对地址。2.变换外部调用符号，将其变换为相对地址。] 2.装入时动态链接方式，指目标模块装入内存时，边装入，边链接。[优点：1.便于修改和更新目标模块。2.便于实现目标模块的共享。] 3.运行时动态链接方式， 将模块的链接推迟到程序执行时才进行。程序执行过程中发现被调用模块未装入内存，OS 立即寻找该模块，并装入内存，连接到调用者模块上。[优点:加快程序的装入过程，节省大量的内存空间。]

4.5 分页存储管理方式 分页存储管理的基本方法：将用户程序的地址空间分为若干个固定大小的区域，成为页或页面。将内存空间分为若干个物理块，页和块的大小相同，可以将程序的若干页面放入内存的多个多个可以不相邻的物理块中。最后一页经常装不满，形成不可利用的碎片，称为页内碎片。页面过小，可以减少内存碎片的总空间，提高内存的利用效率，但会使一个程序占用较多的页面。导致页表过长，占用大量内存，降低页面换进换出的效率。页面过大，减少了页表的长度，但会增大页内碎片。**页表：** 实现从页号到物理块号的地址映射，允许将进程的各个页面离散的存储在内存的任一物理块中。

4.5.2 地址变换机构 任务是将逻辑地址中的页号转换为内存中的物理块号，是借助页表来完成的。1.基本的地址变换机构： 页表大多驻留在内存中，在系统中设置一个页表寄存器，存放页表在内存的起始地址和页表的长度。进程未执行时，页表的起始地址和长度存放在进程的 PCB 中。当调度到该进程时才将这两个数据装入到页表寄存器中。当进程要访问某个逻辑地址中的数据时，分页地址变换机构会自动地将有效地址(相对地址)分为页号和页内地址，以页号为索引检索页表。检索前先将页号和页表长度比较，若页号大于页表长度，则报错。否则将页表起始地址和页号与页表项长度的乘积相加，得到该表项在页表中的位置，于是可以得到该页的物理块号，将之装入物理地址寄存器中。再将有效地址寄存器中的页内地址送入物理地址寄存器的块内地址字段中，完成逻辑地址到物理地址的转换。2.具有快表的地址变换机构： 提高地址变换速度，在地址变换机构中增设一个具有并行查询能力的高速缓冲寄存器，快表。在 CPU 给出有效地址后，有地址变换机构自动将页号 P 送入快表中，并将此页号与快表中所有页号进行比较，若有匹配的页号，便可直接从快表中读出该页所对应的物理块号，并送到物理寄存器中。如果在快表中没有找到，则需要访问内存中的页表，把从页表项中读出的物理块号送到地址寄存器，同时将此页表项存入快表中。若满了，就替换认为不再使用的页表项。

4.5.4.1 两级页表 当页表过大很难找到连续的内存空间来存放页表的时候，可将页表进行分页，是每个页面的大小与内存物理块的大小相同，编号，并离散的存在不同的物理块中，同时创建一张外层页表，在页表项中记录了页表页面的物理块号。利用外层页表和页表的两级页表结构时间从逻辑地址到物理地址的变换。[两级页表并没有解决长页表占用内存大的现象。]在采用两级页表结构的情况下，对于正在运行的进程，必须将外层页表调入内存，离散页表只需调入一页或几页，保证进程运行，需要更多时再调入。

4.5.4.2 多级页表 将外层页表再进行分页，将各分页离散的装入到内存物理块中。

6.7.1 缓冲的引入 1.缓和 CPU 与 IO 设备间速度不匹配的矛盾。2.减少 CPU 的中断频率，放宽对 CPU 中断响应时间的限制。3,。解决数据力度不匹配的问题。4.提高 CPU 与 IO 设备间的并行性，提高系统吞吐量和设备的利用率。

6.7.2 单缓冲区和双缓冲区 1.单缓冲区： 每当用户进程发出 IO 请求时，操作系统便在主存中为之分配一个缓冲区。在字符设备输入时，缓冲区用于暂存用户输入的一行数据，在输入期间，用户进程被挂起以等待数据输入完毕；在输出时，用户进程将一行数据输入到缓冲区后继续进行处理。当用户进程有第二行数据输出时，如果第一行数据未提取完毕，则用户进程阻塞。2.双缓冲区： 为了加快输入和输出速度，提高设备利用率，在设备输入时，先将数据送入第一缓冲区，装满后便转向第二缓冲区。此时 OS 可以从第一缓冲区中取数据，送入用户进程。对于字符设备，若采用行输入方式，双缓冲能消除用户等待的时间，即用户在输入完第一行后，CPU 执行第一行的命令，用户可继续向第二缓冲区中输入下一行数据。同样的，为了实现数据的双向传输，必须设置两个缓冲区，一个用作发送缓冲，另一个用作接受缓冲。

6.7.3 环形缓冲区 将多个缓冲区组成环形的缓冲区形式。1.环形缓冲区的组成： 多个缓冲区，缓冲区的大小相同。输入的多缓冲区分为用于装输入数据的空缓冲区 R，已装满数据的缓冲区 G 和计算进程正在使用的现行工作缓冲区 C。多个指针。输入的缓冲区可设置 3 个指针，用于指示计算进程下一个可用缓冲区 G 的指针 Nextg，指示输入进程下次可用的空缓冲区 R 的指针 Nexti，用于指示计算进程正在使用的缓冲区 C 的指针 Current。2.环形缓冲区的使用：

6.7.4 缓冲池 为了提高缓冲区的利用率，在缓冲池中设置了多个可供若干个进程共享的缓冲区。缓冲区仅仅是一组内存的链表，缓冲池测试包含了一个管理的数据结构及一组操作函数的管理机制，用户管理多个缓冲区。1.缓冲池的组成： 每个缓冲区由用于标识和管理的缓冲首部以及用于存放数据的缓冲体两部分组成。一般将缓冲池中具有相同类型的缓冲区链接成一个队列：空白缓冲队列，输入队列，输出队列。

7.2 文件逻辑结构的类型 文件逻辑结构的基本要求：1.有助于提高对文件的检索速度。2.方便对文件进行修改。3.降低文件存放在外存上的存储费用，尽量减少文件占用的存储空间。从结构上分：有结构文件(文本文件)和无结构文件(二进制文件)，从组织方式上分： 顺序文件，一系列记录按照某种顺序排列的文件；索引文件，可变长记录文件建立一张索引表，为每个记录设置一个表项，加速对记录的检索；索引顺序文件，为每个文件创建索引表时，不是为每个记录建一个索引表项，而使为一组记录中的第一个记录建立索引表项。

7.2.2 顺序文件 顺序文件中的排列结构分有 1.串结构，按存入时间的先后进行排序，与关键词无关。每次检索时都需要从头开始。2.顺序结构，制定一个字段作为关键词，每个关键词都是唯一的。顺序文件的优缺点： 顺序文件的最佳应用场合是对文件中的记录进行批量存取时。所有逻辑文件中顺序文件的存取效率最高。在查找和修改单个记录时，顺序文件需要逐个查找，表现的性能会很差。增加或删除一个记录都比较困难。

7.2.3 记录寻址 1.隐式寻址方式： 对于定长记录的顺序文件，如果已知当前记录的逻辑地址，文件指针移动一个定长文件的长度 L，便可以找到下一个文件的逻辑地址。对于变长记录的顺序文件，每次读或写完当前记录后，需将读写指针加上当前文件的记录长度 Li，便是下一个文件记录的逻辑地址。 2.显示寻址方式： 可对定长记录的文件实现直接或随机访问。1).确定文件中记录的位置，为文件中的每个记录用一个 0~N-1 的整数表示，每一个记录可以通过第一个记录的相对地址+文件长度*记录号计算得到。2).利用关键字，利用关键字匹配的方式，按顺序逐个匹配寻找。

7.2.4 索引文件 1.按关键字建立索引： 为变长记录文件建立一张索引表，为主文件中的每个记录在索引表中分别设置一个表项，记录指向记录的指针(即记录在逻辑地址空间的首址)及记录的长度 L，按关键字排序。因此索引表也是一张定长记录的顺序表。把变长记录顺序文件的顺序检索转变为对定长记录索引文件的随机检索，从而加快对记录检索的速度和实现直接存取。2 具有多个索引表的索引文件： 为满足不同用户的不同目的，使用不同的关键词来检索记录，需要为顺序文件建立多个索引表。[优点：极大的提高了对文件的查找速度，插入和删除记录也很方便。缺憾：每个记录都有一个索引项，增加了存储开销。]

7.2.5 索引顺序文件 1.索引顺序文件的特征： 一个是引入了文件索引表，通过索引表可以时间对索引顺序文件的随机访问；另一个是增加了溢出文件，用来记录新增加的，删除和修改的记录。能有效克服变长记录文件的缺点，又不需要付出过多代价。2.一级索引顺序文件： 只使用了一级索引，索引项中含有首记录的关键字和指向记录的指针。在对索引顺序文件检索时，首先利用用户提供的关键字以及某种查找算法检索索引表，找到该记录所在记录组中的第一个记录的表项，从中得到该记录组的第一个记录在为主文件中的位置，然后利用顺序查找方式查找主文件。3.两级索引顺序文件: 顺序文件建立了多级索引,即为索引文件建立一张索引表,形成两级索引表。

7.3 文件目录 文件目录也是一种数据结构，用于标识系统中的文件及其物理地址，供检索时使用。对目录管理的要求：1.实现"按名存取"，2.提高对目录的检索速度，3.文件共享，4.允许文件重名。

7.3.1 文件控制块和索引结点 文件控制块 FCB 是为了能对一个文件进行正确的存取而必须为文件设置用于描述和控制文件的数据结构。文件与文件控制块一一对应，文件控制块的有序集合称为文件目录，也是一个文件。1.文件控制卡 FCB： FCB 中含有基本信息，存取控制信息和使用信息三类信息。2.索引结点： 把文件名与文件描述信息分开，时文件描述信息单独形成一个称为索引结点的数据结构。在文件目录中的每个目录项仅由文件名和指向该文件所对应的索引结点的指针所构成。磁盘索引结点：存放在磁盘上的索引结点，每个文件有唯一的一个磁盘索引结点。内存索引结点：存放在内存中的索引结点，文件被打开时，要将磁盘索引结点号拷贝到内存的索引结点中。

7.3.2 简单的文件目录 1.单机文件目录： 整个文件系统中只建立一张目录表，每个文件占一个目录项。为表明每个目录项是否空闲，有设置了一个状态位。优点是简单。能够实现目录管理中最基本的功能：按名存取。不满足其他三方面要求。即：查找速度慢，不允许重名，不便于实现文件共享。2.两级文件目录： 为每个用户再建立一个单独的用户文件目录 UFD，由用户所有文件的文件控制块组成。建立一个主文件目录 MFD，MFD 中每个用户目录都占有一个目录项，包括用户名和指向该用户目录文件的指针。两级文件目录提高了检索目录的速度，在不同的用户目录中可以使用相同的用户名，不同用户可以使用不同的文件名访问系统中的同一共享文件。

7.3.3 树形结构目录 1.树形目录： 可以明显提高对目录的检索速度和文件系统的性能。从当前目录开始直到数据文件位置所构成的路径名称为相对路径，把从树根开始的路径名称为绝对路径名。[优点：查询速度更快，层次结构更加清晰，能够有效的进行文件的管理和保护。]

7.3.4 目录查询技术 1.线性检索法： 在单级目录中，利用用户提供的文件名，用顺序查找法直接从文件目录中找到指定名称的目录项。在树形目录中，用户提供的文件名是由多个文件分量名组成的路径名，需要对多级目录进行查找。2.Hash 方法： 系统利用用户提供的文件名，变换为文件目录的索引值，利用索引值到目录中查找。这样显著提高了检索速度。

8.2 文件存储空间管理 为文件分配磁盘时，除了需要文件分配表外，系统还应为可分配存储空间设置相应的数据结构，即磁盘分配表，用于基础可供分配的存储空间情况，提供对盘块进行分配和回收的手段。

8.2.1 空间表法和空闲链表法 1.空闲表法： 为每个文件分配一块连续的存储空间。系统为外存上的所有空间区建立一张空闲表，每个空闲区对应于一个空闲表项，包括表项序号、该空闲区的第一个盘块号、该区的空闲盘块数等。将所有空闲区按其其实盘块号递增的次序排列，形成空闲盘块表。2.空闲链表法： 是将所有空闲盘区拉成一条空闲链。根据构成链所用基本元素的不同，链表分为空闲盘块链和空闲盘区链。1).空间盘块链： 是将磁盘上的所有空闲空间以盘块为单位拉成一条链，其中的每个盘块都有指向后继盘块的指针。[优点：用于分配和回收一个盘块的过程非常简单。缺点：分配和回收的效率较低，空闲盘块链会很长。] 2).空闲盘区链： 是将磁盘上的所有空闲盘区(每个盘区可包含若干个盘块)拉成一条链，含有用于指示下一个空闲盘区的指针和指明本盘区大小的信息。即在内存中为盘区建立一张链表。[分配和税收的过程比较复杂，效率高，空闲盘区链较短。]

8.2.2 位示图法 位示图时利用二进制的一位来表示一个盘块的使用情况。盘块的分配： 1).顺序扫描位示图，2).将所找到的二进制位数转换为相应的盘块，3).修改位示图。盘块的回收： 1).将回收盘块的盘块号转换成位示图中的行号和序列号。2).修改位示图。[优点：从位示图中很容易找打一个或一组相邻的空闲盘块。占用空间小。]

8.3.3 成组链接法 1.空闲盘块的组织： 1).空闲盘块号栈，用来存放当前可用的一组空闲盘块的盘块号及栈中尚有的空闲盘块号数 N，N 兼作栈顶指针。2).文件区中的所有空闲盘块被分成若干个组(比如每组 100 个盘块)。3).每一组韩后的盘块总数 N 和改组所有的盘块号记入其前一组的第一个盘块的 S.free(0)~S.free(99)，形成一条链。4).将第一组的盘块总数和所有的盘块号记入空闲盘块号栈中，作为当前可供分配的空闲盘块。5).最末一组只有 99 个盘块可用，存入前一组的 S.free(1)~S.free(99)中，S.free(0)作为结束标志。2.空闲盘块的分配与回收： 当系统要为用户分配文件所需的盘块时，须调用盘块分配过程来完成。先检查空闲盘块栈是否上锁，未上锁，便从栈顶取出一空闲盘块号，将对应的盘块分配给用户，然后栈顶指针向下移动一格。若盘块号已经是栈底，即 S.free(0)，这是当前栈中最后一个可以分配的盘块号。由于该盘块号对应的盘块中记录的是下一组可用的盘块号，因此必须调用磁盘读过程将栈底盘块号所对应盘块的内容读入栈中，作为新的盘块号栈的内容，并把原栈底对应的盘块分配出去(其中的有用数据已读入栈中。)。然后再分配一相应的缓冲区(盘块缓冲区)。最后把栈中的空闲盘块书减一并返回。 在系统回收空闲盘块时，须调用盘块回收过程进行回收。将回收盘块的盘块号记入空闲盘块号栈的顶部，并执行空心盘块数加 1 操作。当栈中空闲盘块号数目已达到 100 时，表示栈已满，便将现有栈中的 100 个盘块号记入新回收的盘块中，再将其盘块号作为新栈底。

进程同步问题

生产者-消费者问题

生产者-消费者问题是相互合作的进程关系的一种抽象。

1.记录型信号量解决：假定在生产者和消费者之间的公用缓冲池中有 N 个缓冲区，利用互斥信号量 mutex 实现各进程对缓冲池的互斥使用；利用信号 empty 和 full 分别表示缓冲池中空缓冲区和满缓冲区的数量。

```
int in=0,out=0; item buffer[n]; semaphore mutex=1,empty=n,full=0;
void producer(){
        do{
                producer an item nextp;
                ...
                wait(empty); wait(mutex); buffer[in] = nextp; in:=(in+1)%n;
                signal(mutex); signal(full);
        }while(TRUE);
}
void consumer(){
        do{
                wait(full); wait(mutex); nextc=buffer[out]; out=(out+1)%n;
                signal(mutex); signal(empty); consumer the item in nextc;
                ...
        }while(TRUE);
}
void main(){
```

```
        cobegin
                producer(); consumer();
        coend
}
```

2.利用 AND 信号量解决：即用 Swait(empty,mutex); 代替 wait(empty);wait(mutex);，用 Ssignal(mutex,full); 代替 signal(mutex);signal(full);，用 Swait(full,mutex) 代替 wait(full);wait(mutex);，用 Ssingl(mutex,empty)代替 Signal(mutex);Signal(empty);

3.利用管程解决：首先需要建立一个管程，包括(1).put(x);把生产的产品放入缓冲池中，用整型变量 count 来表示缓冲区中产品的数目。(2).get(x);消费者利用该过程从缓冲池中取出一个产品。cwait(condition);当管程被占用时，其他进程调用该过程时阻塞，并挂在条件 condition 的队列上。csignal(condition);唤醒在 cwait 执行后阻塞的 condition 条件队列上的进程。管程描述：

```
Monitor producerconsumer{
        item buffer[n]; int in,out; condition notfull,notempty; int count;
        public:
        void put(item x){
                if(count>=N) cwait(notfull);
                buffer[in]=x; in=(in+1)%N; count++; csignal(notempty);
        }
        void get(item x){
                if(count<=0) cwait(notempty);
                x=buffer[out]; out=(out+1)%N; count--; csignal(notfull);
        }
        {in=0;out=0;count=0;}
}PC;
void producer(){
        item x;
        while(TRUE){
                ...
                produce an item in nextp;
                PC.put(x);
        }
}
void consumer(){
        item x;
        while(TRUE){
                PC.get(x);
                consume the item in nextc;
                ...
        }
}
void main(){
        cobegin
        producer(); consumer();
        coend;
}
```

哲学家进餐问题

有五个哲学家共用一个圆桌，分别坐在周围的五张椅子上，在圆桌上有五个碗和五只筷子，他们是交替的进行思考和进餐。平时，一个哲学家进行思考，饥饿时便试图取用其左右最靠近他的筷子，只有在拿到两只筷子时才能进餐。进餐完毕，放下筷子，继续思考。

1.记录型信号量解决：

为了实现对筷子的互斥使用，可以用五个信号量表示五只筷子，并构成信号量数组：

```
semaphore chopstick[5]={1,1,1,1,1};所有信号量初始值为 1。哲学家的活动描述：
do{
        .../think...
        wait(chopstick[i]); wait(chopsitck[(i+1)%5]);
        .../eat...
        singal(chopstick[(i)]; signal(chopstick[(i+1)%5]);
        .../think...
}while(TRUE);
```

上述解法可保证不会同时有两个相邻的哲学家进餐，但当五个哲学家同时拿起左边的筷子，便会进入死锁。解决方法：1).至多只有四个哲学家同时拿起左边的筷子，最终保证至少一位哲学家能偶进餐。2).仅当左右两只筷子同时可用时才拿起筷子，3).规定奇数号先拿左边筷子再拿右边筷子，偶数号相反。

2.AND 型信号量解决：

```
semaphore chopstick chopstick[5]={1,1,1,1,1};
do{
        .../think...
        Sswait(chopstick[i],chopstick[(i+1)%5]);
        .../eaat...
        Ssignal(chopstick[i],chopstick[(i+1)%5]);
}while(TRUE);
```

读者-写者问题

设置互斥信号量 Wmutex 实现 Reader 和 Writer 进程在读写时互斥，设置 Readcount 表示正在读的进程数目，仅当 Readcount=0 时，Reader 才执行 wait(Wmutex);成功后执行 Read 操作，同时 Readcount+1。当 Readcount-1 变成 0 时才执行 signal(Wmutex);以便让 Writer 写操作。Readcount 是个临界资源，需要设置一个互斥信号量 rmutex；描述如下：

```
semaphore rmutex=1, wmutex=1; int readcount=0;
void reader(){
        do{
                wait(rmutex); if(readcount==0) wait(wmutex);
                readcount++; signal(rmutex);
                ...
                perform and operation;
                ...
                wait(rmutex); readcount--; if(readcount==0) signal(wmutex);
                signal(rmutex);
        }while(TRUE);
}
void writer(){
        do{
                wait(wmutex);
                perform and operation;
                signal(wmutex);
        }while(TRUE);
}
void main(){
        cobegin
        reader(); writer();
        coend
}
```

2.利用信号量集机制解决：

此时增加一个限制，即最多允许 RN 个读者同时读。引入信号量 L，初值为 RN，通过执行 wait(L,1,1)来控制读者数量，当有一个读者进入时执行。第 RN+1 个读者会因为 wait(L,1,1)失败而阻塞。描述：

```
int RN; semaphore L=RN, mx=1;
void reader()){
        do{
                Swait(L,1,1); Swait(mx,1,0);
                ...
                perform and operation;
                ...
                Ssignal(L,1);
        }while(TRUE);
}
void writer(){
        do{
                Swait(mx,1,1;L,RN,0);
                perform and operation;
                Ssignal(mx,1);
        }while(TRUE);
}
void main(){
        cobegin
        reader(); writer();
        coend
}
```

Swait 起着开关的作用，无 writer 进程时，mx=1；reader 可以进入，mx=0 时，reader 不能进入，writer 写操作。Swait(mx,1,1;L,RN,0);表示即没有读者，也没有写者的时候，writer 才可以进行写操作。

银行家算法避免死锁

每一个进程进入系统时，必须申明需要使用的资源的最大数目。其数目不能超过系统拥有的资源总数目。当进程请求一组资源时，系统必须明确是否有足够的资源分配，若有，再考虑分配后是否会进入不安全状态，如果会不安全就不分配，让进程阻塞。

1.银行家算法的数据结构： 1).可利用资源向量 Available，含有 m 个元素的数组，每个元素代表一类资源可以利用的数目。初始值是系统中该类资源的总数目。Available[i]=K 表示系统中 Rj 类资源目前共有 K 个。2).最大需求矩阵 Max，Max[i,j]=K,表示进程 i 需要 Rj 类资源的最大数目是 K 个。 3).分配矩阵 Allocation，Allocation[i,j]=K,表示进程 i 当前已分得 Rj 类资源的数目为 K 个。 4).需求矩阵 Need，Need[i,j]=K,表示进程 i 还需要 Rj 类资源 K 个方能完成任务。

2.银行家算法： Request[i]是进程 Pi 的请求向量，进程 Pi 发出请求后，系统执行步骤：1).判断 Request[i,j]是否小于等于 Need[i,j]，若不成立，便报错，成立则接着判断 2).Request[i,j]是否小于等于 Available[j]，不成立就报错，成立则再 3).试探性的测试资源分配，修改数据结构中的数值：Available[j]=Available[j]-Request[i,j];

Allocation[i,j]=Allocation[i,j]+Request[i,j]; Need[i,j]=Need[i,j]-Request[i,j];

4).最后执行系统安全性算法，检查资源分配后系统是否处于安全状态，若安全，则 commit 资源分配，若不安全，则 rollback 资源分配，让 Pi 进程阻塞等待。

3.安全性算法： 1).设置两个向量，工作向量 work，表示系统可提供个进程继续执行所需的各类资源的数目，安全算法开始时，work:=Available。Finish 向量，表示系统是否还有足够的资源分配给进程以完成任务。开始时 Finish[i]:=false; 当判断有足够资源时才令 Finish[i]=true；2).从进程集合中找到一个资源需求小于等于系统当前可提供的资源数目的进程，即满足：Finish[i]=false; Need[i,j]<=Work[j]; 找不到则系统处于不安全状态，找到了就执行 3).测试性的赋予进程 Pi 资源，并回收 Pi 结束后能够释放的所有资源：Work[j]=Work[j]+Allocation[i,j]; Finish[i]=true; goto step 2;回到第二步，循环判断整个进程队列。如果所有进程的 Finish 都是 true，则表明系统是处于安全状态的。

页面置换算法

进程在运行过程中，若其要访问的页面不存在，而内存有没有空闲空间调入外存页面时，系统需要将内存中的页面调出到磁盘的对换区中。选择换出页面的算法为页面置换算法。进程“抖动”指页面不停的从内存换出，从磁盘换进。

1.最佳置换算法： 其所选择的被淘汰页面是以后最长时间不使用的页面。该算法能保证获得最低的缺页率。但未来的页面访问序列是无法预知的。

7,0,1,2,0,3,0,4,2,3,0,3,2,1,2,0,1,7,0,1 进程运行时，先装入 7,0,1 三个页面。当进程访问页面 2 时，产生缺页中断，最佳置换算法会选择将 7 页面淘汰，0 会在第 5 个页面使用，1 会在第 14 个页面使用，7 则在第 18 个页面使用。依次类推。

2.先进先出算法： 该算法总是淘汰最先进入内存的页面，即淘汰驻留时间最长的页面。将内存中的页面按先后次序链接成一个队列，设置一个替换指针，指向最老的页面。

3.最近最久未使用 LRU 算法： 根据页面调入内存后的使用情况作出决策。算法赋予每个页面一个访问字段，用来记录一个页面自上次被访问以来所经历的时间 t。

4.最近最少使用 LFU 算法： 该算法选择在最近时期使用最少的页面作为淘汰页。采用移位寄存器的方式，每访问某个页面，便将该移位寄存器的最高位置 1，再隔一定时间右移一次。总和最小的便是最近最少使用的页面。

5.简单 CLOCK 置换算法： 为每页设置一位访问位，再将内存中的所有页面都通过链接指针链成一个循环队列。当某页面被访问，其访问位置 1。置换算法在选择一页淘汰时，只需检查访问位，若为 0，就选择换出该页，若是 1，就置 0，给该页第二次驻留的机会，再按照 FIFO 算法检查下一个页面。0 就换出，1 就置 0。当最后一个页面还是 1 时，置 1 后回到队首检查第一个页面，再往下走。该算法是循环检查各个页面的使用情况，故称为 Clock。

6.改进型的 Clock 算法： 在将一个页面换出时，若页面被修改过，就重新写回到磁盘上，若没被修改，则写回。修改过的页面比未修改的页面置换代价大。改进的 clock 算法还增加了一个置换代价位，既是未使用过的页面，又是未修改过的页面才是首要淘汰的页面。

访问位 A 与修改为 M 的四种组合：

a 类 A=0，M=0，表示该页最近未被访问也未被修改，是最佳置换页

b 类 A=0，M=1，c 类 A=1，M=0，d 类 A=1，M=1

1).从指针所指示的当前位置开始扫描循环队列，寻找第一类页面，将遇到的第一类页面作为淘汰页。第一次扫描不修改访问位 A。

2).第一步失败，开始第二轮扫描，寻找第二类页面，找到并淘汰。将所有扫描过的访问位都置 0.

3).第二步失败，指针回到开始的位置，将所有访问位置 0，重复第一步，寻找第一类页面。改进后的算法可以减少磁盘的 IO 操作，但需要多次扫描，算法本身开销大。

7.页面缓冲算法 PBA：

影响页面换进换出的效率的因素有：1).页面置换算法，是影响效率的最主要因素。好的页面置换算法，可使进程运行时具有较低的缺页率，从而减少换进换出的开销。2).写回磁盘的频率，在系统中建立一个已修改换出页面的链表，每一个已修改的换出页面都先挂在链表上，挂到一定数量后再一次性写回到磁盘中。减少了对磁盘 IO 操作的次数，减少了已修改页面换出的开销。3).读入内存的频率，如果挂在链表上的页面需要再次被访问时，可以不用从磁盘获取，减少了将页面从磁盘读入内存的频率，减少了页面换进的开销。

PBA 的主要特点是：1.显著降低了页面换进，换出的频率，使磁盘 IO 的操作次数大为减少，减少了页面换进换出的开销。2.能够采用较简单的置换策略。

在页面中设置空闲页面链表和修改页面链表来显著降低页面换进换出的频率。

空闲页面链表：是一个空闲物理块链表，为系统掌握，用于分配给频繁发生缺页的进程，以降低该进程的缺页率。当进程需要读入一个页面时，便可利用空闲页面链表中的第一个物理块来装入该页。当有一个未被修改的页面要换出时，将其带数据一同挂在空闲页面链表的尾部，进程以后需要用到该页面时，就可直接调入，免除磁盘操作。

修改页面链表：目的是为了减少页面换出到磁盘的次数。

3. 在操作系统中为什么要引入进程概念？它会产生什么样的影响?

为了使程序在多道程序环境下能并发执行，并能对并发执行的程序加以控制和描述，从而在操作系统中引入了进程概念。 影响：使程序的并发执行得以实行。

4.试说明 PCB 的作用?为什么说 PCB 是进程存在的唯一标志?

a. PCB 是进程实体的一部分，是操作系统中最重要的记录型数据结构。PCB 中记录了操作系统所需的用于描述进程情况及控制进程运行所需的全部信息。因而它的作用是使一个在多道程序环境下不能独立运行的程序(含数据)，成为一个能独立运行的基本单位，一个能和其它进程并发执行的进程。 b. 在进程的整个生命周期中，系统总是通过其 PCB 对进程进行控制，系统是根据进程的 PCB 而不是任何别的什么而感知到该进程的存在的，所以说，PCB 是进程存在的唯一标志。

5．试说明进程在三个基本状态之间转换的典型原因。

答：（1）就绪状态→执行状态：进程分配到 CPU 资源 （2）执行状态→就绪状态：时间片用完 （3）执行状态→阻塞状态：I/O 请求 （4）阻塞状态→就绪状态：I/O 完成

7．在进行进程切换时，所要保存的处理机状态信息有哪些?

答：进行进程切换时，所要保存的处理机状态信息有：

（1）进程当前暂存信息

（2）下一指令地址信息

（3）进程状态信息

（4）过程和系统调用参数及调用地址信息。

8．试说明引起进程创建的主要事件。 答：引起进程创建的主要事件有：用户登录、作业调度、提供服务、应用请求。

9．试说明引起进程被撤销的主要事件。 答：引起进程被撤销的主要事件有：正常结束、异常结束（越界错误、保护错、非法指令、特权指令错、运行超时、算术运算错、I/O 故障）、外界干预（操作员或操作系统干预、父进程请求、父进程终止）。

10．在创建一个进程时所要完成的主要工作是什么?

答： （1）OS 发现请求创建新进程事件后，调用进程创建原语 Creat()； （2）申请空白 PCB； （3）为新进程分配资源； （4）初始化进程控制块； （5）将新进程插入就绪队列.

11．在撤销一个进程时所要完成的主要工作是什么?

答： （1）根据被终止进程标识符，从 PCB 集中检索出进程 PCB，读出该进程状态。（2）若被终止进程处于执行状态，立即终止该进程的执行，路调度标志真，指示该进程被终止后重新调度。 （3）若该进程还有子进程，应将所有子孙进程终止，以防它们成为不可控进程。 （4）将被终止进程拥有的全部资源，归还给父进程，或归还给系统。 （5）将被终止进程 PCB 从所在队列或列表中移出，等待其它程序搜集信息。

12．试说明引起进程阻塞或唤醒的主要事件是什么?

答：a. 请求系统服务；b. 启动某种操作；c. 新数据尚未到达；d. 无新工作可做.

13．为什么说多级反馈队列调度算法能较好地满足各方面用户的需要?

【解】（1）对于终端型用户来说，他们提交的大多属于较小的交互型作业，系统只要能使这些作业（进程）在第一队列所规定的时间片内完成，便可使终端型作业用户都感到满意。（2）对短批处理作业用户来说，在第一队列中执行一个时间片或至多只需在第二队列和第三队列中各执行一个时间片即可完成。 （3）对长批处理作业用户来说，只要将作业依次在第 1，2，,,,,，n 个队列中运行，然后再按轮转方式运行，用户不必担心其作业长期得不到处理。

17.为实现分页存储管理，需要哪些硬件支持?

需要有页表机制、地址变换机构的硬件支持。

19 实现 LRU 算法所需的硬件支持是什么?

a. 寄存器，用于记录某进程在内存中各页的使用情况；

b. 栈，用于保存当前使用的各个页面的页面号。

21． 在请求分页系统中，应从何处将所需页面调入内存?

外存分为文件区和对换区（对换区可认为就是以页面形式组织的虚拟内存区），若系统有足够的对换区空间，可在进程运行前，将与该进程有关的文件拷贝到对换区，需要时从对换区调入；

2）若系统缺少足够的对换区空间，则凡是不会被修改的文件，可直接从文件区调入，需换出时，便须调到对换区，以后需要时再从对换区调入.

3）UBIX 方式。。。

23．在请求分页系统中，通常采用哪种页面分配方式物理块分配策略?

三种分配方式：固定分配局部置换、可变分配全局置换、可变分配局部置换。

24. 在单缓冲情况下，为什么系统对一块数据的处理时间为 max(C, T)+M ?

在块设备输入时，假定从磁盘把一块数据输入到缓冲区的时间为 T；操作系统将缓冲区数据传送给用户区的时间为 M；而 CPU 对这一块数据进行计算得时间为 C。在单缓冲情况下，由于设备的输入操作和 CPU 的处理操作可以并行，所以系统对每一整块数据的处理时间为 max(C, T)+ M。

25. 为什么在双缓冲情况下，系统对一块数据的处理时间为 max(C, T)?

该方式又称缓冲对换方式，在设备输入时，先将数据送入第一缓冲区，装满后便转向第二缓冲区。此时操作系统可以从第一缓冲区移出数据，并送入用户进程。接着由 CPU 对数据进行计算。在双缓冲区中，不仅设备的输入操作和 CPU 的处理操作可以并行，设备的输入操作和数据的传送操作也可以并行，因此耗时大约为 max(C+M,T)。考虑到 M 是内存中数据块的“搬家”耗时，非常短暂可以省略，因此近似地认为是：max(C,T)。

26 何谓逻辑文件？何谓物理文件？(何谓文件逻辑结构？何谓文件的物理结构)

文件的逻辑结构是指从用户的观点出发所观察到的文件组织形式，也就是用户可以直接处理的数据及其结构，它独立于物理特性，；而文件的物理结构则是指文件在外存上的存储组织形式，与存储介质的存储性能有关。

28．采用单级目录能否满足对目录管理的主要要求？为什么？

采用单级目录不能完全满足对目录管理的主要要求，只能实现目录管理最基本的功能即按名存取。由于单级目录结构采用的是在系统只配置一张目录表用来记录系统中所有文件的相关信息，因此此目录文件可能会非常大，在查找时速度慢，另外不允许用户文件有重名的现象，再者由于单级目录中要求所有用户须使用相同的名字来共享同一个文件，这样又会产生重名问题，因此不便于实现文件共享。

29． 目前广泛采用的目录结构是哪种？它有什么优点？

目前广泛采用的目录结构是树型目录结构。它具有以下优点：

a.能有效提高对目录的检索速度：假定文件系统中有 N 个文件，在单级目录中，最多要检索 N 个目录项，但对于有 i 级的树型目录，在目录中每检索一个指定文件，最多可能要检索 i*N 的根号 i 次方个目录项。

b.允许文件重名：由于在树型结构的文件系统中，是利用文件路径名来检索文件的，故允许每个用户在自己的分目录中使用与其他用户文件相同的名字。

c.便于实现文件共享：在树型目录中，用户可通过路径名来共享其他用户的文件，也可将一个共享文件链接到自己的目录下，从而使文件的共享变得更为方便，其实现方式也非常简单，系统只需在用户的目录文件中增设一个目录项，填上用户赋予该共享文件的新文件名，以及该共享文件的唯一标识符即可。

31 何谓作业、作业步和作业流?

【解】作业包含通常的程序和数据，还配有作业说明书。系统根据该说明书对程序的运行进行控制。批处理系统中是以作业为基本单位从外存调入内存。

作业步是指每个作业运行期间都必须经过若干个相对独立相互关联的顺序加工的步骤。

作业流是指若干个作业进入系统后依次存放在外存上形成的输入作业流；在操作系统的控制下，逐个作业进程处理，于是形成了处理作业流。

32 在什么情况下需要使用作业控制块 JCB？其中包含了哪些内容？

【解】每当作业进入系统时，系统便为每个作业建立一个作业控制块 JCB，根据作业类型将它插入到相应的后备队列中。

JCB 包含的内容通常有：1） 作业标识 2)用户名称 3)用户账户 4)作业类型（CPU 繁忙型、I/O 芳名型、批量型、终端型）5)作业状态 6)调度信息（优先级、作业已运行）7)资源要求 8)进入系统时间 9) 开始处理时间 10) 作业完成时间 11) 作业退出时间 12) 资源使用情况等

33．在作业调度中应如何确定接纳多少个作业和接纳哪些作业?

【解】作业调度每次接纳进入内存的作业数，取决于多道程序度。应将哪些作业从外存调入内存，取决于采用的调度算法。最简单的是先来服务调度算法，较常用的是短作业优先调度算法和基于作业优先级的调度算法。

35 在抢占调度方式中，抢占的原则是什么？

【解】剩夺原则有： （1）时间片原则 各进程按时间片运行，当一个时间片用完后，便停止该进程的执行而重新进行调度。这种原则适用于分时系统、大多数实时系统，以及要求较高的批处理系统。 （2）优先权原则 通常是对一些重要的和紧急的作业赋予较高的优先权。当这种作业到达时，如果其优先权比正在执行进程的优先权高，便停止正在执行的进程，将处理机分配给优先权高的进程，使之执行。 （3）短作业（进程）优先原则 当新到达的作业（进程）比正在执行的作业（进程）明显地短时，将剩夺长作业（进程）的执行，将处理机分配给短作业（进程），使之优先执行。

36、选择调度方式和调度算法时，应遵循的准则是什么？

【解】应遵循的准则有 （1）面向用户的准则：周转时间短，响应时间快，截止时间的保证，优先权准则。 （2）面向系统的准则：系统吞吐量高，处理机利用率好，各类资源的平衡利用。

37 在批处理系统、分时系统和实时系统中，各采用哪几种进程（作业）调度算法?

【解】 批处理系统：FCFS 算法、最小优先数优先算法、抢占式最小优先数优先算法 2 分时系统：可剥夺调度、轮转调度 实时系统：时间片轮转调度算法、非抢占优先权调度算法、基于时钟中断抢 占的优先权调度算法、立即抢占的优先权调度。

38 何谓静态和动态优先权？确定静态优先权的依据是什么？

【解】静态优先权是在创建进程时确定的，且在进程的整个运行期间保持不变。动态优先权是指，在创建进程时所赋予的优先权，是可以随进程的推进或随其等待时间的增加而改变的，以便获得更好的调度性能。 确定静态优先权的依据是： （1）进程类型，通常系统进程的优先权高于一般用户进程的优先权。 （2）进程对资源的需要。 （3）用户要求，用户进程的紧迫程度及用户所付费用的多少来确定优先权的。

39、试比较 FCFS 和 SPF 两种进程调度算法。

【解】FCFS 算法按照作业提交或进程变为就绪状态的先后次序，分派 CPU。当前作业或进程占有 CPU，直到执行完或阻塞，才让出 CPU。在作业或进程唤醒后，并不立即恢复执行，通常等到当前作业或进程让出 CPU。FCFS 比较有利于长作业，而不利于短作业；有利于 CPU 繁忙的作业，而不利于 I/O 繁忙的作业。 SPF 有利于短进程调度，是从就绪队列中选出一估计运行时间最短的进程，将处理机分配给它，使它立即执行并一直执行到完成，或发生某事件而被阻塞放弃处理机时，再重新调度。比 FCFS 改善了平均周转时间和平均带权周转时间，缩短了作业的等待时间，提高了系统的吞吐量。但 SPF 有其不容忽视的缺点：该算法对长作业不利；完全未考虑作业的紧迫程度，因而不能保证紧迫性作业（进程）会被及时处理；用户可能会有意无意地干扰作业的运行时间，致使该算法不一定能真正做到短作业优先调度。

12． 在生产者消费者问题中，如果缺少了 signal(full)或 signal(empty),对执行结果有何影响?

答：如果缺少 signal(full)，那么表明从第一个生产者进程开始就没有改变信号量 full 值，即使缓冲池产品已满，但 full 值还是 0，这样消费者进程执行 wait(full)时认为缓冲池是空而取不到产品，消费者进程一直处于等待状态。

如果缺少 signal(empty)，在生产者进程向 n 个缓冲区投满产品后消费者进程才开始从中取产品，这时 empty=0，full=n，那么每当消费者进程取走一个产品 empty 值并不改变，直到缓冲池取空了，empty 值也是 0，即使目前缓冲池有 n 个空缓冲区，生产者进程要想 再往缓冲池中投放产品也会因为申请不到空缓冲区被阻塞。

13.在生产消费者问题中，如果将两个 wait 操作即 wait(full)和 wait(mutex)互换位置，或者将 signal(mutex)不 signal（full）互换位置，结果如何?

答：将 wait(full)和 wait(mutex)互换位置后，可能引起死锁。考虑系统中缓冲区全满时，若一生产者进程先执行了 wait(mutex)操作并获得成功，则当再执行 wait(empty)操作时，它将因失败而进入阻塞状态，它期待消费者进程执行 signal(empty)来唤醒自己，在此之前，它不可能执行 signal(mutex)操作，从而使试图通过执行 wait(mutex)操作而进入自己的临界区的其他生产者和所有消费者进程全部进入阻塞状态，这样容易引起系统死锁。若 signal(mutex)和 signal(full)互换位置后只是影响进程对临界资源的释放次序，而不会引起系统死锁，因此可以互换位置。