1.17 列出下列操作系统的基本特点：

a.批处理 **b.**交互式 **c.**分时 **d.**实时 **e.**网络 **f.**并行式 **g.**分布式 **h.**集群式 **i.**手持式

Answer: **a.**批处理：具有相似需求的作业被成批的集合起来，并把它们作为一个整体通过一个操作员或自动作业程序装置运行通过计算机。通过缓冲区，线下操作，后台和多道程序，运用尝试保持 CPU 和 I/O 一直繁忙，从而使得性能被提高。批处理系统对于运行那些需要较少互动的大型作业十分适用。它们可以被更迟地提交或获得。

b.交互式：这种系统由许多短期交易构成，并且下一个交易的结果是无法预知的。从用户提交到等待结果的响应时间应该是比较短的，通常为 1 秒左右。

c.分时：这种系统使用 CPU 调度和多道程序来经济的提供一个系统的人机通信功能。CPU 从一个用户快速切换到另一个用户。以每个程序从终端机中读取它的下一个控制卡，并且把输出的信息正确快速的输出到显示器上来替代用 soopled card images 定义的作业。

d.实时：经常用于专门的用途。这个系统从感应器上读取数据，而且必须在严格的时间内做出响应以保证正确的性能。

e.网络：提供给操作系统一个特征，使得其进入网络，比如;文件共享。

f.并行式：每一个处理器都运行同一个操作系统的拷贝。这些拷贝通过系统总线进行通信。

g.分布式：这种系统在几个物理处理器中分布式计算，处理器不共享内存或时钟。每个处理器都有它各自的本地存储器。它们通过各种通信线路在进行通信，比如：一条高速的总线或一个本地的网络。

h.集群式：集群系统是由多个计算机耦合成单一系统并分布于整个集群来完成计算任务。

i.手持式：一种可以完成像记事本，email 和网页浏览等简单任务的小型计算机系统。手持系统与传统的台式机的区别是更小的内存和屏幕以及更慢的处理能力。

2.12 采用微内核方法来设计系统的主要优点是什么？在微内核中如何使客户程序和系统服务相互作用？微内核方法的缺点是什么？

Answer:优点主要包括以下几点：

a）增加一个新的服务不需要修改内核

b）在用户模式中比在内核模式中更安全、更易操作

c）一个简单的内核设计和功能一般导致一个更可靠的操作系统

用户程序和系统服务通过使用进程件的通信机制在微内核中相互作用，例如发送消息。这些消息由操作系统运送。微内核最主要的缺点是与进程间通信的过度联系和为了保证用户程序和系统服务相互作用而频繁使用操作系统的消息传递功能。

3.1 论述短期,中期和长期调度之间的区别.

Answer:a.短期调度:在内存作业中选择就绪执行的作业,并为他们分配 CPU。

b.中期调度:作为一种中等程度的调度程序，尤其被用于分时系统，一个交换方案的实施，将部分运行程序移出内存，之后，从中断处继续执行。

c.长期调度（作业调度程序）:确定哪些作业调入内存以执行.

它们主要的不同之处是它们的执行的频率。短期调度必须经常调用一个新进程，由于在系统中，长期调度处理移动的作业时，并不频繁被调用，可能

在进程离开系统时才被唤起。

3.2 问:描述一下内核在两个进程间进行上下文功换的动作.

Answer:总的来说，操作系统必须保存正在运行的进程的状态，恢复进程的状态。保存进程的状态主要包括 CPU 寄存器的值以及内存分配，上下文切换还必须执行一些确切体系结构的操作，包括刷新数据和指令缓存。

（书中答案）进程关联是由进程的 PCB 来表示的，它包括 CPU 寄存器的值和内存管理信息等。当发生上下文切换时，内核会将旧进程的关联状态保存在其 PCB 中，然后装入经调度要执行的新进程的已保存的关联状态。

4.2 描述一下线程库采取行动进行用户级线程上下文切换的过程

答：用户线程之间的上下文切换和内核线程之间的相互转换是非常相似的。但它依赖于线程库和怎样把用户线程指给内核程序。一般来说，用户线程之间的上下文切换涉及到用一个用户程序的轻量级进程（LWP）和用另外一个线程来代替。这种行为通常涉及到寄存器的节约和释放。

4.3 在哪些情况下使用多内核线程的多线程方案比单处理器系统的单个线程方案提供更好的性能。

答：当一个内核线程的页面发生错误时，另外的内核线程会用一种有效的方法被转换成使用交错时间。另一方面，当页面发生错误时，一个单一线程进程将不能够发挥有效性能。因此，在一个程序可能有频繁的页面错误或不得不等待其他系统的事件的情况下，多线程方案会有比单处理器系统更好的性能。

5.1 为什么对调度来说，区分 **I/0** 限制的程序和 **CPU** 限制的程序是重要的？

答：I/0 限制的程序有在运行 I/O 操作前只运行很少数量的计算机操作的性质。这种程序一般来说不会使用很多的 CPU。另一方面，CPU 限制的程序利用整个的时间片，且不做任何阻碍 I/O 操作的工作。因此，通过给 I/O 限制的程序优先权和允许在 CPU 限制的程序之前运行，可以很好的利用计算机资源。

5.2 讨论以下各对调度标准在某种背景下会有的冲突

a.CPU 利用率和响应时间

b.平均周转时间和最大等待时间

c.I/O 设备利用率和 CPU 利用率

答：a.CPU 利用率和响应时间：当经常性的上下文切换减少到最低时，CPU 利用率增加。通过减少使用上下文切换程序来降低经常性的上下文切换。但这样可能会导致进程响应时间的增加。

b.平均周转时间和最大等待时间：通过最先执行最短任务可以使平均周转时间最短。然而，这种调度策略可能会使长时间运行的任务永远得不到调度且会增加他们的等待时间。

c.I/O 设备利用率和 CPU 利用率：CPU 利用率的最大化可以通过长时间运行 CPU 限制的任务和同时不实行上下文切换。I/O 设备利用率的最大化可以通过尽可能调度已经准备好的 I/O 限制的任务。因此，导致上下文切换 。

5.4 考虑下列进程集，进程占用的 **CPU** 区间长度以毫秒来计算：

进程	区间时间	优先级
P_1	10	3
P_2	1	1
P_3	2	3
P_4	1	4
P_5	5	2

假设在时刻 0 以进程 P_1，P_2，P_3，P_4，P_5 的顺序到达。

a.画出 4 个 Gantt 图分别演示用 FCFS、SJF、非抢占优先级（数字小代表优先级高）和 RR（时间片＝1）算法调度时进程的执行过程。

b.在 a 里每个进程在每种调度算法下的周转时间是多少？

c.在 a 里每个进程在每种调度算法下的等待时间是多少？

d.在 a 里哪一种调度算法的平均等待时间对所有进程而言最小？

答：a.略

b.周转时间

	FCFS	RR	SJF	非抢占优先级
P1	10	19	19	16
P2	11	2	1	1
P3	13	7	4	18
P4	14	4	2	19
P5	19	14	9	6

c.等待时间

	FCFS	RR	SJF	非抢占优先级
P1	0	9	9	6
P2	10	1	0	0
P3	11	5	2	16
P4	13	3	1	18
P5	14	9	4	2

d.SJF

5.10 解释下面调度算法对短进程编程度上的区别：

a. FCFS

b. RR

c．多级反馈队列

答:a. FCFS----区别短任务是因为任何在长任务后到达的短任务都将会有很长的等待时间。

b. RR-----对所有的任务都是能够相同的（给它们相同的 CPU 时间区间），所以，短任务可以很快的离开系统，只要它们可以先完成。

c．多级反馈队列和 RR 调度算法相似——它们不会先选择短任务。

6.3 忙等待的含义是什么？在操作系统中还有哪些其他形式的等待？忙等待能完全避免吗？给出你的答案。

答：忙等待意味着一个进程正在等待满足一个没有闲置处理器的严格循环的条件。或者，一个进程通过放弃处理器来等待，在这种情况下的块等待在将来某个适当的时间被唤醒。忙等待能够避免，但是承担这种开销与让一个进程处于沉睡状态，当相应程序的状态达到的时候进程又被唤醒有关。

6.5 如果一个同步元是在一个用户级程序中使用的，解释在一个单处理器系统中为什么通过停止中断去实现这个同步元是不适合的？

答：如果一个用户级程序具有停止中断的能力，那么它能够停止计时器中断，防止上下文切换的发生，从而允许它使用处理器而不让其他进程执行。

6.15 讨论在读者-作者问题中的公平和吞吐量的权衡问题。提出一种解决读者-作者问题而不引起饥饿的方法

答：在读者-作者问题中吞吐量是由利益多的读者增加的，而不是让一个作家独占式地获得共同的价值观。另一个方面，有利于读者可能会导致饥饿的作者。在读者-作者问题中的借能够通过保持和等待进程有关的时间戳来避免。当作者完成他的任务，那么唤醒那些已经等了最长期限的进程。当读者到达和注意到另一个读者正在访问数据库，那么它只有在没有等待的作者时才能进入临界区域。这些限制保证公平。

6.16

管程的 signal 操作和信号量的 signal 操作有什么不同？

管程的signal操作在以下情况下是不能继续进行的:当执行signal操作并且无等待线程时，那么系统会忽略 signal 操作，认为 signal 操作没有发生过。如果随后执行 wait 操作，那么相关的线程就会被阻塞。然后在信号量中，即使没有等待线程，每个 signal 操作都会是相应的信号量值增加。接下来的等待操作因为之前的信号量值的增加而马上成功进行。

6.21

假设将管程中的 wait 和 signal 操作替换成一个单一的构件 await（B），这里 B 是一个普通的布尔表达式，进程执行直到 B 变成真

a. 用这种方法写一个管程实现读者—作者问题。

b. 解释为什么一般来说这种结构实现的效率不高。

c. 要使这种实现达到高效率需要对 await 语句加上哪些限制？（提示，限制 B 的一般性，参见 Kessels[1977].）

a. 读者—作者问题可以进行以下修改，修改中产生了 await 声明：读者可以执行“await(active writers == 0 && waiting writers == 0)”来确保在进入临界区域时没有就绪的作者和等待的作者。作者可以执行“await(active writers == 0 && active readers == 0)”来确保互斥。

b. 在 signal 操作后，系统检查满足等待条件满足的等待线程，检查其中被唤醒的等待线程。这个要求相当复杂，并且可能需要用到交互的编译器来评估在不同时间点下的条件。可以通过限制布尔条件，使布尔变量和其他部分分开作为独立的程序变量（仅仅用来检查是否相等的一个静态值）。在这种情况下，布尔条件可以传达给运行时系统，该系统可以执行检查每一个它所需要的时间，以确定哪些线程被唤醒。

7.1

假设有如图 7.1 所示的交通死锁。

a. 证明这个例子中实际上包括了死锁的四个必要条件。

b. 给出一个简单的规则用来在这个系统中避免死锁。

a. 死锁的四个必要条件：(1)互斥；(2) 占有并等待；(3) 非抢占；(4) 循环等待。

互斥的条件是只有一辆车占据道路上的一个空间位置。占有并等待表示一辆车占据道路上的位置并且等待前进。一辆车不能从道路上当前的位置移动开（就是非抢占）。最后就是循环等待，因为每个车正等待着随后的汽车向前发展。循环等待的条件也很容易从图形中观察到。

b. 一个简单的避免这种的交通死锁的规则是，汽车不得进入一个十字路口如果明确地规定，这样就不会产生相交。

7.2

考虑如下的死锁可能发生在哲学家进餐中，哲学家在同个时间获得筷子。讨论此种情况下死锁的四个必要条件的设置。讨论如何在消除其中任一条件来避免死锁的发生。

死锁是可能的，因为哲学家进餐问题是以以下的方式满足四个必要条件：1）相斥所需的筷子， 2 ）哲学家守住的筷子在手，而他们等待其他筷子， 3 ）没有非抢占的筷子，一个筷子分配给一个哲学家不能被强行拿走，4 ）有可能循环等待。死锁可避免克服的条件方式如下： 1 ）允许同时分享筷子， 2 ）有哲学家放弃第一双筷子如果他们无法获得其他筷子，3 ）允许筷子被强行拿走如果筷子已经被一位哲学家了占有了很长一段时间 4 ）实施编号筷子，总是获得较低编号的筷子，之后才能获得较高的编号的筷子。

7.4

对下列问题对比循环等待方法和死锁避免方法(例如银行家算法)：

a. 运行费用

b. 系统的吞吐量

死锁避免方法往往会因为追踪当前资源分配的成本从来增加了运行费用。然而死锁避免方法比静态地防止死锁的形成方法允许更多地并发使用资源。从这个意义上说，死锁避免方案可以增加系统的吞吐量。

7.11 考虑下面的一个系统在某一时刻的状态：

	Allocation	**Max**	**Available**
	A B C D	**A B C D**	**A B C D**
P0	**0 0 1 2**	**0 0 1 2**	**1 5 2 0**
P1	**1 0 0 0**	**1 7 5 0**	
P2	**1 3 5 4**	**2 3 5 6**	
P3	**0 6 3 2**	**0 6 5 2**	
P4	**0 0 1 4**	**0 6 5 6**	

使用银行家算法回答下面问题：

a.Need 矩阵的内容是怎样的？

b.系统是否处于安全状态？

c.如果从进程 **P1** 发出一个请求（**0 4 2 0**），这个请求能否被满足？

Answer：a.Need矩阵的内容是P_0（0 0 0 0） P_1（0 7 5 0） P_2（1 0 0 2） P_3（0 0 2 0） P_4（0 6 4 0）。

b. .系统处于安全状态，因为Available矩阵等于（1 5 2 0），进程P_0和P_3都可以运行，当进程P_3运行完时，它释放它的资源，而允许其它进程运行。

c.可以被满足，满足以后，Available矩阵等于（1 1 0 0），当以次序P_0, P_2, P_3, P_1 ,P_4运行时候，可以完成运行。

8.7 比较页式存储与段式存储为了从虚地址转变为物理地址，在被要求的地址转化结构的内存数量方面的有关内容。

c 页式存储需要更多的内存来保持转化结构，段式存储的每个段只需要两个寄存器，一个保存段的基地址，另一个保存段的长度。另一方面，页式存储每一页都需要一个入口，这个入口提供了那页所在的物理地址。

8.15 问：比较段页式表和哈希页表在处理大量的地址空间上，在什么环境下，哪一个方案更好？

答：当一个程序占用大的虚拟地址空间的一小部分时，哈希页表更适合小一点的。哈希页表的缺点是在同样的哈希页表上，映射多个页面而引起的冲突。如果多个页表映射在同个入口处，则横穿名单相应的哈希页表可能导致负担过重。这种间接最低的分割分页方案，即每一页表条目保持有关只有一页。

9.3 问：什么是写时拷贝功能，在什么情况下，有利于此功能？支持此功能的硬件是什么？

答：当两个进程正在访问同一套程序值（例如，代码段的二进制代码）在写保护的方式下，映射相应的页面到虚拟地址空间是有用的，当写操作进行时，拷贝必须允许两个程序分别进行不同的拷贝而不干扰对方。硬件要求：在每个内存访问的页表需要协商，以检查是否该页表是写保护。如果确实是写保护，陷阱会出现，操作系统可以解决这个问题。

9.7 问：讨论在哪一种情况下，**LFU**（最不经常使用）页置换比 **LRU**（最近最少使用）页置换法产生较少的页面错误，什么情况下则相反？

答：考虑下面顺序存取在内存的系统的串，可容纳 4 页内存：1 1 2 3 4 5 1，当访问 5 时，LFU 算法将会替换除了 1 以外的其他页面，则在接下来读取 1 时，就不用更次替换了。反来过说，如果串为：1 2 3 4 5 2，LRU 算法性能更好。

9.15 颠簸的原因是什么？系统怎样检测颠簸？一旦系统检测到颠簸，系统怎样做来消除这个问题？

Answer:

分配的页数少于进程所需的最小页数时发生颠簸，并迫使它不断地页错误。该系统可通过对比多道程序的程度来估计 CPU 利用率的程度，以此来检测颠簸。降低多道程序的程度可以消除颠簸。

12.1 除了 **FCFS,**没有其他的磁盘调度算法是真正公平的（可能会出现饥饿）。

a:说明为什么这个断言是真。

b:描述一个方法，修改像 **SCAN** 这样的算法以确保公平性。

C:说明为什么在分时系统中公平性是一个重要的目标。

D:给出三个以上的例子，在这些情况下操作系统在服务 **I/O** 请求时做到“不公平”很重要。

【答】a. 人们提出了关于磁头目前具备理论上可以尽快达到这些要求的磁道新要求

b. 所有那些预定的年龄更老的要求可能是“被迫”处于队列的顶端，一个有关为每个位可定表明，没有任何新的要求可提前这些请求。对于 SSTF，其余的队列将不得不根据最后的这些“旧”的要求重新组织。

c. 为了防止超长的响应时间。

d. 寻呼和交换应优先于用户的要求。

为了其他内核启动的 I / O，如文件系统元数据的写入，优先于用户 I / O 可能是可取的。如果内核支持实时进程的优先次序，这些进程的 I / O 请求该是有利的。