1.1 在多道程序和分时环境中，多个用户同时共享一个系统，这种情况导致多种安全问题。a. 列出此类的问题 b.在一个分时机器中，能否确保像在专用机器上一样的安全度？并解释之。

Answer:a.窃取或者复制某用户的程序或数据；没有合理的预算来使用资源（CPU，内存，磁盘空间，外围设备）ｂ．应该不行，因为人类设计的任何保护机制都会不可避免的被另外的人所破译，而且很自信的认为程序本身的实现是正确的是一件困难的事。

1.2 资源的利用问题在各种各样的操作系统中出现。试例举在下列的环境中哪种资源必须被严格的管理。（ａ）大型电脑或迷你电脑系统（ｂ）与服务器相联的工作站（ｃ）手持电脑

Answer: （ａ）大型电脑或迷你电脑系统：内存和 CPU 资源，外存，网络带宽（ｂ）与服务器相联的工作站：内存和 CPU 资源（ｃ）手持电脑：功率消耗，内存资源

1.3 在什么情况下一个用户使用一个分时系统比使用一台个人计算机或单用户工作站更好？

Answer:当另外使用分时系统的用户较少时，任务十分巨大，硬件速度很快，分时系统有意义。充分利用该系统可以对用户的问题产生影响。比起个人电脑，问题可以被更快的解决。还有一种可能发生的情况是在同一时间有许多另外的用户在同一时间使用资源。当作业足够小，且能在个人计算机上合理的运行时，以及当个人计算机的性能能够充分的运行程序来达到用户的满意时，个人计算机是最好的，。

1.4 在下面举出的三个功能中，哪个功能在下列两种环境下，(a)手持装置(b)实时系统需要操作系统的支持？(a)批处理程序(b)虚拟存储器(c)分时

Answer:对于实时系统来说，操作系统需要以一种公平的方式支持虚拟存储器和分时系统。对于手持系统，操作系统需要提供虚拟存储器，但是不需要提供分时系统。批处理程序在两种环境中都是非必需的。

1.5 描述对称多处理（ＳＭＰ）和非对称多处理之间的区别。多处理系统的三个优点和一个缺点？

Answer: ＳＭＰ意味着所以处理器都对等，而且 I/O 可以在任何处理器上运行。非对称多处理有一个主处理器控制系统，与剩下的处理器是随从关系。主处理器为从处理器安排工作，而且 I/O 也只在主处理器上运行。多处理器系统能比单处理器系统节省资金，这是因为他们能共享外设，大容量存储和电源供给。它们可以更快速的运行程序和增加可靠性。多处理器系统能比单处理器系统在软、硬件上也更复杂（增加计算量、规模经济、增加可靠性）

1.6 集群系统与多道程序系统的区别是什么？两台机器属于一个集群来协作提供一个高可靠性的服务器的要求是什么？

Answer: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。另一方面，多道程序系统可以被看做是一个有多个 CPU 组成的单一的物理实体。集群系统的耦合度比多道程序系统的要低。集群系统通过消息进行通信，而多道程序系统是通过共享的存储空间。为了两台处理器提供较高的可靠性服务，两台机器上的状态必须被复制，并且要持续的更新。当一台处理器出现故障时，另一台处理器能够接管故障处理的功能。

1.7 试区分分布式系统（distribute system）的客户机-服务器（client-server）模型与对等系统（peer-to-peer）模型

Answer: 客户机-服务器（client-server）模型可以由客户机和服务器的角色被区分。在这种模型下，客户机向服务器发出请求，然后服务器满足这种请求。对等系统（peer-to-peer）模型没有这种严格的区分角色，。实际上，在系统中的所有结点被看做是对等的，而且这些结点既可以是客户机也可以是服务器，或者两这都是。也许一个结点从另一个对等结点上请求一个服务，或者，这个结点满足在系统中的另一个结点的请求。比如，一个系统中的结点共享烹饪方法。在客户机-服务器（client-server）模型下，所有方法都被存储在服务器上。如果一个客户机想要获得烹饪方法，它必须向那台服务器发出请求。在对等系统（peer-to-peer）模型下，一个结点可以向另外的结点请求指定的烹饪方法。存储了这种烹饪方法的那个结点（或几个结点）可以把烹饪的方法提供给发出请求的结点。注意每个对等结点既可以扮演客户机（发出请求），也可以扮演服务器（提供请求）。

1.8 如果一个由两个结点组成的集群系统正在运行一个数据库，试描述集群软件可以用哪两种方法管理存取磁盘的数据，并说明每种方法的优点和缺点。

Answer:两种方法:非对称集群系统(asymmetric clustering)和并行集群系统(parallel clustering).对于非对称集群系统，一个主机运行这个数据库，而其它主机只是监测这个数据库。如果服务器出现故障，进行监测的主机就会转变成运行这个数据库的主机。这是提供适当的冗余。然而，它没有利用具有潜在处理能力的主机。对于并行集群系统，数据库可以在两个并行的主机上运行。在并行集群系统上实现的困难是提供一些分布式锁机制给共享磁盘上的文件。

1.9 网络计算机是怎样不同与传统的个人计算机的？试取出一些使用网络计算机的好处的方案。

Answer:网络计算机是基于一台核心的计算机作为其服务器。同时，它也具有一个最小化的操作系统来管理这些资源。另一方面，个人计算机必须在不依赖于核心计算机的基础上，能够独立提供所有被请求的功能。在行政花费太高以及共享导致更高效的使用资源的情景下是精确的，在这些环境中网络计算机是理想的。

1.10 中断（interupt）的目的是什么？陷阱（trap）与中断的区别是什么？陷阱可以被用户程序（user program）有意地的产生吗？如果可以，那目的是什么？

Answer: 中断是一种在系统内硬件产生的流量变化。中断操作装置是用来处理中断请求；然后返回控制中断的上下文和指令。陷阱是软件产生的中断。中断可以被用来标志 I/O 的完成，从而排除设备投票站（device polling）的需要。陷阱可以被用来调用操作系统的程序或者捕捉到算术错误。

1.11 内存存储是被用于高速的 I/O 设备，其目的是为了避免增加 CPU 的过度运行。

(a)设备的 CPU 接口是怎样与转换器（transfer）协作的？

(b)当内存操作完全时，CPU 是怎么知道的？

(c)当 DMA 控制器正在转换数据时，CPU 是被允许运行其它程序的。这种进程与用户程序的运行冲突吗？如果冲突的话，试描述可能引起哪种冲突？

Answer: CPU 可以通过写数据到可以被设备独立存储的寄存器中来启动 DMA 操作。当设备接收到来自 CPU 的命令时，启动响应的操作。当设备完成此操作时，就中断 CPU 来说明操作已经完成。设备和 CPU 都可以被内存同时访问。内存控制器对这两个实体以公平的方式给内存总线提供存取。CPU 可能不能同时以很快的速度配给给内存操作，因为它必须去竞争设备而使得自己存取到内存总线中去。

1.12 一些计算机系统没有在硬件中提供个人模式（privileged mode）。对于这种

计算机系统来说，可能构成安全的操作系统吗？对可能和不可能两种情况分别给出理由。

Answer:一种类型处理器的操作系统需要在任何时候都被控制（或监测模式）。有两种方法可以完成这个操作：a.所有用户程序的软件翻译（像一些 BASIC，Java，LISP systems）。在软件中，软件解释程序能够提供硬件所不能提供的。b.要求所有程序都用高级语言编写，以便于所以目标代码都被编译出来。编译器将会产生硬件忽略的防护性检查（in-line 或功能调用）。

1.13 给出缓存（caches）十分有用的两个理由。他们解决了什么问题？他们引起了什么问题？

如果缓存可以被做成装备想要缓存的容量（例如，缓存像磁盘那么大），为什么不把它做的那么大，其限制的原因是什么？

Answer:当两个或者更多的部件需要交换数据，以及组成部件以不同的速度完成转换时，缓存是十分有用的。缓存通过在个组成部件之间提供一个中间速度的缓冲区来解决转换问题。如果速度较快的设备在缓存中发现它所要的数据，它就不需要再等待速度较慢的设备了。缓存中的数据必须与组成部件中的要一致。如果一个组成部件中的数据值改变了，缓存中的这个数据也必须更新。在多进程系统中，当有不止一个进程可能进入同一个数据时，这就成了一个显著的问题。一个组成部件将会被一个同等大小的组成部件所消除，但是只有当;(a)缓存和组成部件有相同状态存储能力（也就是，当断电的时候，组成部件还能保存它的数据，缓存也一样能保存它的数据），(b)缓存是可以负担的起的，因为速度更快的存储器意味着更高的价格。

1.14 试举例说明在下列的进程环境中，快速缓冲贮存区的数据保持连贯性的问题是怎样表明的？(a)单道程序系统（Single-processor systems）(b)多道程序系统（Mulitiprocessor systems）(c)分布式系统（Distribute systems）

Answer: 在单道程序系统（Single-processor systems）中，当一个进程发布更新给快速缓冲贮存区的数据时，内存需要被更新。这些更新一种快速的或缓慢的方式执行。在多道程序系统（Mulitiprocessor systems）中，不同的进程或许在它的本地存储上存储相同的内存位置。当更新发生时，其它存储的位置需要使其无效或更新。在分布式系统（Distribute systems）中，快速存储区数据的协调不是问题，然而，当客户机存储文件数据时，协调问题就会被提及。

1.15 试描述一个机器装置为了阻止一个程序避免修改与其它程序有联系的内存而执行内存保护。

Answer:处理器可以追踪哪个位置是与每个进程相联系的以及限制进入一个程序的范围的外面位置。信息与一个程序的内存范围有关，它可以通过使用库，限制寄存器和对每个进入内存的信息执行检查来维持其本身。

1.16 哪种网络结构最适合下列环境：（a）一个寝室楼层（b）一个大学校园（c）一个州（d）一个国家。

Answer:

（a）一个寝室楼层：A　LAN

（b）一个大学校园: A　LAN,possibly a WAN for a very large campuses.

（c）一个州:A WAN

（d）一个国家: A WAN

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。

1.18 手持计算机中固有的折中属性有哪些？

Answer:手提电脑比传统的台式 PC 机要小的多。这是由于手提电脑比台式 PC 机具有更小的内存，更小的屏幕，更慢的处理能力的结果。因为这些限制，大多数现在的手提只能完成基本的任务，比如：记事本，email 和简单的文字处理。然而，由于它们较小的外形，而十分便于携带，而且当它们具备无线上网时，就可以提供远程的 email 通信和上网功能。

2.1 操作系统提供的服务和功能可以分为两个类别。简单的描述一下这两个类别并讨论他们的不同点。

Answer:第一种操作系统提供的服务是用来保护在系统中同时运行的不同进程。进程只被允许获得与它们地址空间有联系的内存位置。同样，进程不允许破坏和其他用户有关的文件。一个进程同样不允许在没有操作系统的干预下直接进入设备。第二种服务由操作系统提供的服务是提供一种新的功能，而这种功能并不直接被底层的硬件支持。虚拟存储器和文件系统就是由操作系统提供的这种新服务的实例。

2.2 列出操作系统提供的五项服务。说明每项服务如何给用户提供便利。说明在哪些情况下用户级程序不能够提够这些服务。

Answer： a.文件执行.操作系统一个文件的目录（或章节）装入到内存并运行。一个用户程序不能被信任，妥善分配 CPU 时间。

b.I/O 操作. 磁盘，磁带，串行线，和其他装置必须在一个非常低的水平下进行通信。用户只需要指定装置和操作执行要求，然后该系统的要求转换成装置或控制器的具体命令.用户级程序不能被信任只在他们应该获得时获得装置

和只使用那些未被使用的装置。

c.文件系统操作.在文件创建、删除、分配和命名时有许多细节是用户不能执行的。磁盘空间块被文件所使用并被跟踪。删除一个文件需要清除这个文件的信息和释放被分派给这个文件的空间。用户程序不仅不能够保证保护方法的有效实施，也不能够被信任只会分配空闲的空间和在删除文件是清空空间。

d.通信.信息在系统间交换要求信息转换成信息包，送到网络控制器中，通过通信媒介进行传播，并由目的地系统重新组装。信息包调整和数据修改是一定会发生的。此外，用户程序也许不能够协调网络装置的取得，或者接收完全不同的其他进程的信息包。

e.错误检测.错误检测在硬件和软件水平下都会发生。在硬件水平下，所有数据转移都必须仔细检查以确保数据在运送中不会被破坏。在媒介中的所有数据都必须被检查以确保他们在写入媒介时没有被改变。在软件水平下，为了数据，媒介不需不间断的被检查。例如，确保信息存储中被分配和还未被分配的空间块的数量和装置中所有块的数量的一致。进程独立经常有错误（例如，磁盘中数据的破坏），所以必须有一个统筹的程序（操作系统）来处理各种错误。同样，错误经过操作系统的处理，在一个系统中程序不再需要包含匹配和改正所遇可能错误的代码。

2.3 讨论向操作系统传递参数的三个主要的方法。

Answer:

1.通过寄存器来传递参数

2.寄存器传递参数块的首地址

3.参数通过程序存放或压进堆栈中，并通过操作系统弹出堆栈。

2.4 描述你怎样能够统计到一个程序运行其不同部分代码时，它的时间花费数量的数据图表，并说明它的重要性。

Answer:一个能够发布定期计时器打断和监控正在运行的命令或代码段当中断被进行时。一个满意的配置文件，其中的代码块都应积极覆着被程序在代码的不同的部分花费时间。一旦这个配置文件被获得，程序员可以尽可能的优化那些消耗大量 CPU 资源的代码段。

2.5 操作系统关于文件管理的五个主要活动是什么？

Answer:

1.创建和删除文件

2.创建和删除目录

3.提供操作文件和目录的原语的支持

4.将文件映射到二级存储器上

5.在稳定（非易失的）的存储媒介上备份文件。

2.6 在设备和文件操作上用相同的系统调用接口的好处与不足是什么？

Answer:每一个设备都可以被得到只要它是一个在文件系统的文件。因此大多数内核通过文件接口处理设备，这样相对容易，加一个新的设备通过执行硬件确定代码来支持这种抽象的文件接口。因此，这种方式不仅有利于用户程序代码的发展，用户程序代码可以被写入设备和文件用相同的方式，还有利于设备驱动程序代码，设备驱动程序代码可以书面支持规范定义的 API.使用相同接口的缺点是很难获得某些设备档案存取的 API 范围内的功能，因此，结果或者是丢失功能或者是丢失性能。但有些能够被克服通过使用 ioctl 操作，这个操作为了进程在设备上援引操作提供一个通用接口。

2.7命令解释器的用途是什么？为什么它经常与内核是分开的？用户有可能通过使用由操作系统提供的系统调用接口发展一个新的命令解释器？

Answer:命令解释器从用户或文件中读取命令并执行，一般而言把他们转化成系统调用。它通常是不属于内核，因为命令解释会有所变动。用户能够利用由操作系统提供的系统调用接口开发新的命令解释器。这命令解释器允许用户创建、管理进程和确定它们通信的方法（例如通过管道和文件）。所有的功能都被用户程序通过系统调用来使用，这个也可能有用户开发一个新的命令行解释。

2.8 通信的两种模式是什么？这两种模式的优点和缺点是什么？

Answer:通信的两种模式是 1）共享内存，2）消息传递。这两种模式的最基本的不同是在它们的性能上。一个内存共享块是通过系统调用创建的。然而，一旦内存共享块在两个或更多的进程间建立，这些进程可以借助内存共享块来通信，不再需要内核的协助。另一方面，当 send()和 receive()操作被调用时,信息传递通常包含系统调用。因此，因为内核是直接的包含在进程间通信的，一般而言，它的影响比内存共享小。然而，消息传递可以用作同步机制来处理通信进程间的行动。也就是说，send()和 receive()段可以用来协调两个通信进程的动作。另一方面，内存共享没有提供这种同步机制的进程。

2.9 为什么要把机制和策略区分开来？

Answer:机制和策略必须区分开来，来保证系统能够被很容易的修改。没有两个系统的装置是完全相同的，所以每一个装置都想要把操作系统改为适合自己的。当机制和政策分开时，政策可以随意的改变但机制还是不能改变。这种安排提供了一个更灵活的制度

2.10 为什么 Java 提供了从 Java 程序调用由 C 或 C++编写的本地方法的能力？举出一个本地方法有用的例子。

Answer:Java 程序的开发是用来作为 I/O 独立的平台。因此，这种语言没有提供途径给许多特殊的系统资源，例如从 I/O 设备读取。为了运行一个系统特定的 I/O 操作，你必须用一种支持这些特性的语言（例如 C 或 C++）写。记住一个 Java 程序调用由另外一种语言编写的本地方法写将不再结构中立。

2.11 有时获得一个分层方法是有困难的如果操作系统的两个部件相互依存。识别一个方案，在这个方案中并不非常清楚如何为两个作用紧密相连的系统部件分层。

Answer:虚拟内存子系统和存储子系统 通常是紧密耦合，并由于以下的相互作用需要精心设计的层次 系统。许多系统允许文件被映射到一个执行进程的虚拟内存空间。另一方面，虚拟内存子系统通常使用存储 系统来提供当前不在内存中的页。此外，在刷新磁盘之前，更新的文件有时会缓冲到物理内存，从而需要认真 协调使用的内存之间的虚拟内存 子系统和文件系统。

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。

2.13 模块化内核方法的什么方式与分层方法相似？什么方式与分层方法不同？

Answer:模块化内核方法要求子系统通过创建的一般而言狭隘（从功能方面来说是揭露外部模块）的接口来相互作用。分层内核方法在细节上与分层方法相似。但是，分层内核必须要是有严格排序的子系统，这样的子系统在较低层次中不允许援引业务相应的上层子系统 。在模块化内核方法中没有太多的限制，模式在哪方面是随意援引彼此的是没有任何约束的。

2.14 操作系统设计员采用虚拟机结构的主要优点是什么？对用户来说主要有什么好处？

Answer:系统是容易被调试的，此外，安全问题也是容易解决的。虚拟机同样为运作体系提供了一个很好的平台，因为许多不同的操作系统只可以在一个物理系统中运行。

2.15 为什么说一个 JIT 编译器对执行一个 Java 程序是有用的？

Answer:Java 是一种解释语言。这就意味着 Java 虚拟机一次解释一个字节代码。一般来说，绝大多数解释环境是比运行本地二进制慢，因为解释进程要求把每一个命令转化为本地机器代码。一个 JIT 编译器把字节代码转换成本地机器代码，第一次这种方法是偶然碰到的。这就意味着 Java 程序作为一个本地用途（当然，JIT 的这种转换过程是要花费时间的，但并没有像字节代码花费的这么多）是非常重要的一种运行方式。此外，JIT 存储器编译代码以便能够在下一次需要时使用。一个是被 JIT 运行的而不是传统的一般的解释运行的 Java 程序是非常快的。

2.16 在一个系统（例如 VWware）中，来宾作业系统和主机操作系统的关系是什么？在选择主机操作系统时哪些因素需要考虑？

Answer:一个来宾作业系统提供它的服务通过映射到有主机操作系统提供的功能上。一个主要的事情需要被考虑，为了能够支持与来宾作业系统相联系的功能，选择的主机操作系统，从系统调用接口而言，是否足够一般。

2.17 实验性的综合操作系统在内核里有一个汇编器。为了优化系统调用的性能，内核通过在内核空间内汇编程序来缩短系统调用在内核必须经过的途径。这是一种与分层设计相对立的方法，经过内核的途径在这种设计中被延伸了，使操作系统的构造更加容易。分别从支持和反对的角度来综合设计方式对讨论这种内核设计和系统性能优化的影响。

Answer:综合是令人钦佩的由于这种性能通过即时复杂化取得了成功。不幸的是，由于代码的流动很难在内核中调试问题。这种复杂化是系统的详细的表现，让综合很难 port（一个新的编译器必须写入每一种架构）。

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。

3.3 考虑 RPC 机制。考虑的 RPC 机制。描述不可取的情况下可能出现或者不执行的”最多一次”或”到底一旦“语义。说明在没有这些保障的情况下,可能使用的一种机制。

Answer:如果一个 RPC 机制无法支持无论是“最多一次” 或“至少一次”的语义，那么 RPC 服务器不能保证远端程序不会引起多个事件的发生。试想，如果一个远端程序在一个不支持这些语义的系统上从银行账户中撤回投资的资金。很可能一个单一调用的远程过程会导致多种服务器的撤回。

如果一个系统不能支持这两种语义，那么这样一个系统只能安全提供远程程序，这些远程程序没有改变数据，没有提供时间敏感的结果，用我们的银行账户做例，我们当然需要“最多一次” 或“至少一次”的语义执行撤销（或存款）。然而，账户余额成其它账户信息的查询，如姓名，地址等，不需要这些语义。

3.4 图表 3.24 里显示的程序，说明 A 行将会输出什么？

Answer:当控制回到父进程时，它的值会保持在 5，而子进程将更新并拷贝这个值。

3.5 问:下面设计的好处和坏处分别是什么?系统层次和用户层次都要考虑到.

A,对称和非对称通信

B,自动和显式缓冲

C,复制发送和引用发送

D,固定大小和可变大小消息

Answer:A.对称和非对称通信：对称通信的影响是它允许发送者和接收者之间有一个集合点。缺点是阻塞发送时，不需要集合点，而消息不能异步传递。因此，消息传递系统，往往提供两种形式的同步。

B.自动和显式缓冲：自动缓冲提供了一个无限长度的队列，从而保证了发送者在复制消息时不会遇到阻塞，如何提供自动缓存的规范，一个方案也许能保存足够大的内存，但许多内存被浪费缓存明确指定缓冲区的大小。在这种状况下，发送者不能在等待可用空间队列中被阻塞。然而，缓冲明确的内存不太可能被浪费。

C.复制发送和引用发送：复制发送不允许接收者改变参数的状态，引用发送是允许的。引用发送允许的优点之一是它允许程序员写一个分布式版本的一个集中的应用程序。Java’s RMI 公司提供两种发送，但引用传递一个参数需要声明这个参数是一个远程对象。

D.固定大小和可变大小消息：涉及的太多是有关缓冲问题，带有定长信息，一个拥有具体规模的缓冲课容纳已知数量的信息缓冲能容纳的可变信息数量是未知的。考虑 Windows 2000 如何处理这种情况。带有定长信息（<256bytes）,信息从发送者的地址空间被复制至接受进程的地址空间。更大的信息（如变长信息）使用共享内存传递信息。

第四章 线程

4.1 举两个多线程程序设计的例子来说明多线程不比单线程方案提高性能

答：1）任何形式的顺序程序对线程来说都不是一个好的形式。例如一个计算个人报酬的程序。

2）另外一个例子是一个“空壳”程序，如 C-shell 和 korn shell。这种程序必须密切检测其本身的工作空间。如打开的文件、环境变量和当前工作目录。

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。

4.4 以下程序中的哪些组成部分在多线程程序中是被线程共享的？

a.寄存值

b.堆内存

c.全局变量

d.栈内存

答：一个线程程序的线程共享堆内存和全局变量，但每个线程都有属于自己的一组寄存值和栈内存。

4. 5 一个采用多用户线程的多线程方案在多进程系统中能够取得比在单处理器系统中更好的性能吗？

答：一个包括多用户线程的多线程系统无法在多处理系统上同时使用不同的处理器。操作系统只能看到一个单一的进程且不会调度在不同处理器上的不同进程的线程。因此，多处理器系统执行多个用户线程是没有性能优势的。

4.6 就如 4.5.2 章节描述的那样，Linux 没有区分进程和线程的能力。且 Linux 线程都是用相同的方法：允许一个任务与一组传递给 clone()系统调用的标志的进程或线程。但许多操作系统，例如 windows XP 和 Solaris，对进程和线程都是一视同仁。基本上，这种使用 notation 的系统，一个进程的数据结构包括一个指向属于进程的不同线程的指针。区别建模过程和在内核中线程的两种方法。

答：一方面，进程和线程被视为相似实体的系统中，有些系统代码可以简化。例如，一个调度器可以在平等的基础上考虑不同的进程和线程，且不需要特殊的代码，在调度中审查有关线程的进程。另一方面，这种统一会使进程资源限制更加困难。相反，一些额外的复杂性被需要，用来确定哪个线程与哪个进程一致和执行重复的计数任务。

4.7 由 4.11 给出的程序使用了 Pthread 的应用程序编程接口（API），在程序的第 c 行和第 p 行分别会输出什么？

答：c 行会输出 5，p 行会输出 0.

4.8 考虑一个多处理器系统和用多线程对多线程模式编写的多线程程序。让程序中的用户线

程数量多于系统中的处理器的数量，讨论下列情况下的性能意义：

a.由程序分配的内核线程的数量比处理器少

b．由程序分配的内核线程的数量与处理器相同

c．由程序分配的内核线程的数量大于处理器数量但少于用户线程的数量

答：当内核线程的数量少于处理器时，一些处理器将仍然处于空闲状态。因为，调度图中只有内核线程的处理器，而不是用户线程的处理器。当程序分配的内核线程的数量与处理器相同时，那么有可能所有处理器将同时使用。然而，当一个内核块内的内核（因页面错误或同时援引系统调用）相应的处理器将闲置。当由程序分配的内核线程的数量大于处理器数量时，封锁一个内核线程并调出，换入另一个准备执行的内核线程。因此，增加多处理器系统的利用率。

第五章 CPU 调度

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

答：I/O 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 。

5.3 考虑指数平均公式来预测下一次 CPU 区间的长度，使用以下参数值会有什么影响？

a.a=0 和 t=100 毫秒

b.a=0.99 和 t=10 毫秒

答：当 a=0 和 t=100 毫秒时，公式总是会预测下一次的 CPU 区间为 100 毫秒。当 a=0.99 和 t=10 毫秒时，进程最近的行为是给予更高的重量和过去的就能成相比。因此，调度算法几乎是无记忆的，且简单预测未来区间的长度为下一次的 CPU 执行的时间片。

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.甘特图略

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

5.5 下面哪些算法会引起饥饿

a.先来先服务

b.最短工作优先调度

c.轮换法调度

d.优先级调度

答：最短工作优先调度和优先级调度算法会引起饥饿

5.6 考虑 RR 调度算法的一个变种，在这个算法里，就绪队列里的项是指向 PCB 的指针。

a.如果把两个指针指向就绪队列中的同一个进程，会有什么效果？

b.这个方案的主要优点和缺点是什么？

c.如何修改基本的 RR 调度算法，从而不用两个指针达到同样的效果？

答. a. 实际上，这个过程将会增加它的优先权，因为通过经常得到时间它能够优先得以运行。

b. 优点是越重要的工作可以得到更多的时间。也就是说，优先级越高越先运行。然而，结果将由短任务来承担。

c.分配一个更长的时间给优先级越高的程序。换句话说，可能有两个或多个时间片在RR调度中。

5.7 考虑一个运行十个I/O限制任务和一个CPU限制任务的系统。假设，I/O限制任务一次分配给一个I/O操作1毫秒的CPU计算，但每个I/O操作的完成需要 10毫秒。同时，假设间接的上下文切换要0.1毫秒，所有的进程都是长进程。对一个RR调度来说，以下情况时CPU的利用率是多少：

a.时间片是1毫秒

b.时间片是10毫秒

答：a.时间片是1毫秒：不论是哪个进程被调度，这个调度都会为每一次的上下文切换花费一个0.1毫秒的上下文切换。CPU的利用率是1/1.1*100=92%。

b.时间片是10毫秒：这I/O限制任务会在使用完1毫秒时间片后进行一次上下文切换。这个时间片要求在所有的进程间都走一遍，因此，10*1.1+10.1(因为每个I/O限定任务执行为1毫秒，然后承担上下文切换的任务，而CPU限制任务的执行10毫秒在承担一个上下文切换之前）。因此，CPU的利用率是20、21.1*100=94%。

5.8 考虑一个实施多层次的队列调度系统。什么策略能够使一个计算机用户使用由用户进程分配的最大的CPU时间片。

答：这个程序可以使分配给它的没有被完全利用的CPU时间最大化。它可以使用分配给它的时间片中的绝大部分，但在时间片结束前放弃CPU，因此提高了与进程有关的优先级。

1.5.9 考虑下面的基于动态改变优先级的可抢占式优先权调度算法。大的优先权数代表高优先权。当一个进程在等待CPU时（在就绪队列中，但未执行），优先权以α速率改变；当它运行时，优先权以速率β改变。所有的进程在进入就绪队列时被给定优先权为0。参数α和β可以设定给许多不同的调度算法。

a.β＞α＞0时所得的是什么算法？

b.α＜β＜0时所得的是什么算法？

答：a.FCFS

b.LIFO

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。

5.11 用 Window XP 的调度算法，下列什么是数字优先的线程。

a.相对优先级的值为 REALTIME_PRIORITY_CLASS 的属于实体优先类型的线程

b.相对优先级的值为 NORMAL_PRIORITY_CLASS 的属于 NORMAL 类型的线程

c.相对优先级的值为 HIGH_PRIORITY_CLASS 的属于 ABOVE_NORMAL 类型的线程

答：a.26

b.8

c.14

5. 12 考虑在 Solaris 操作系统中的为分时线程的调度算法：

a:一个优先权是 10 的线程的时间片是多少？优先权是 55 的呢？

b:假设优先权是 35 的一个线程用它所有的时间片在没有任何阻止的情况下，这调度算法将会分配给这个线程什么样新的优先权？

c:假设一个优先权是 35 的线程在时间片结束前阻止 I/O 操作。这调度算法将会分配给这个线程什么样新的优先权？

答：a:160 和 40

b:35

C：54

5.13 传统 UNIX 调度在优先数和优先级间成反比关系：数字越高，优先权越低。该调度进程利用下面的方程重新计算进程的优先权一次一秒:

优先权= （最近 CPU 使用率/ 2 ） +基本数

这里的基本数= 60,最近的 CPU 使用率是指一个表明优先权从上一次重新计算后开始进程被 CPU 使用的情况。

假设最近进程 p1 的 CPU 使用率是 40 个，p2 是 18 ，p3 是 10。当优先权重新计算后这三个进程的新的优先权是什么？在此信息的基础上，传统 UNIX 的调度会不会提高或降低 CPU 限制的进程的相对优先权？

答 ： 分配给这些进程的优先权分别是 80，69 和 65.这调度降低了 CPU 限制的进程的相对优先权。

第六章 管程

6.1 第一个著名的正确解决了两个进程的临界区问题的软件方法是 Dekker 设计的。两个进程 P0 和 P1 共享以下变量：

```
boolean flag[2];   /*initially false*/

int turn;
```

进程 Pi(i==0 或 1)和另一个进程 Pj(j==0 或 1)的结构见图 7.27。

证明这个算法满足临界区问题的所有三个要求。

```
do{
    flag[i]=ture;
    while(flag[j]){
        if(turn==j){
            flag[i]=false;
            while(turn==j);
            flag[i]=true;
        }
    }
        临界区
    turn=j;
    flag[i]=false;
        剩余区
}while(1);
```

图 7.27　Dekker 算法中的进程 P_i 结构

答：该算法满足三个相互排斥条件。（1）相互排斥是为了确保使用的变量和标志是可变的。如果所有进程都把他们的变量设置为真，只有一个会成功，那就是哪个进程轮到的问题了。等待中的进程只能够进入它的重要部分当其他进程在更新变量值时。

6.1 这两个进程的临界区域问题的最初的正确的软件解决方案是由 Dekker 提出的。P0、P1 两个进程，具有以下共同的变量：

boolean flag[2]; /* initially false */

int turn;

进程 Pi(i==0 or1)的结构在 6.25 中已经出现过；其他进程为 Pj(j == 1 or 0)。证明这个算法满足关键问题的三个要求。

答：这个算法满足临界区域的三个条件：

（1）在标记和返回变量的使用中，互斥条件是保证的。如果两个进程将它们的标识设为

真，那么只有一个进程会成功进行，即轮到的那个进程。当另一个进程更新它的返回变量时，等待的那个进程只能进入它的临界区域。

（2）就绪的进程，通过标志，返回变量。这个算法没有提供严格的交替。如果一个进程想要进入它们的临界区域，它可以将它的标识设为真，然后进入它们的临界区域。当退出它的临界区域，它可以设置转向其他进程的值。如果这个进程想要在其他进程之前再次进入它的临界区域，它会重复这样的进程：进入它的临界区域，在退出时转向另一个进程。

（3）在双 T 返回变量的使用过程中，界等待受阻。假设两个进程想要进入它们的责任所在的临界区域。它们都将它们的标志的值设为真；而只有轮到的那个线程可以执行；其他的线程处于等待状态。如果界等待没有受阻，当第一个进程重复“进入-退出”它的临界区域这一过程。Dekker 算法在一个进程中设置一个转向另一个进程的值，从而保证另一个进程接下来进入它的临界区域。

6.2 针对有 n 个进程在带有较低时间限制的等待 n-1 个的轮次这样一个临界区域最早的解决该问题的正确方法是由艾森伯格和麦圭尔提出的。这些进程有以下的共同的变量：

枚举 pstate{idle, want in, in cs};

pstate flag[n];

int turn;

所有的枚举标志被初始为空，轮次的最初值是无关紧要的（在 0 和 n-1 之间）。进程 p 的结构在 6.26 中有说明。证明这个算法满足临界区域问题的三项要求。

答：a.互斥：注意到一个进程只有在下列条件满足时才能进入临界区域：没有其他进程在 CS 中有设置的标识变量。这是由于进程在 CS 中设置自身的标识变量之前要先检查其他进程的状态。我们保证没有两个进程将同时进入临界区域。

b.有空让进:考虑以下情况，当多进程同时在 CS 中设置它们的标识变量，然后检查是否有其他进程在 cs 中设置标识变量。当这种情况发生时，所有的进程意识到这里存在进程竞争，在外层 while(1)的循环下进入下一次迭代，重置它们的标识变量到 want 中。现在只有唯一的进程将设置它的轮次变量到 cs 中，这个唯一的进程就是其序号是最接近轮次的。从这个角度来说，对于哪些序号次接近轮次的新的进程就能决定进入临界区域，而且能同时在 CS 中设置它们的标识。随后这些进程意识到这里存在竞争的进程，于是重新启动进入临界区域的进程。在每次迭代中，进程在 cs 中设置的序号值将变得更加接近轮次，最后我们得出以下结论：只有进程 k 在 cs 中设置它的标识，而其他哪些序号在轮次和 k 之间不能在 cs 中设置它们的标识。这个进程仅能进入临界区域。

c.有限等待：有限等待需要满足以下事实：当进程 k 在打算进入临界区域时，它的标识不再置为空闲。任何序号不在轮次和 k 之间的进程不能进入临界区域。与此同时，所有序号落在轮次和 k 之间且又想要进入临界区域的进程能够进入临界区域(这是基于系统一直在进步的事实)，这个轮次值变得越来越接近 k。最后，要么轮次变为 k，要么没有哪些序号在轮次和 k 之间的进程，这样进程 k 就进入临界区域了。

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。

6.4 解释为什么自旋锁不适合在单处理器系统，而经常在多处理器系统下使用？
答：自旋锁不适合在单处理器系统是因为从自旋锁中打破一个进程的条件只有在执行一个不同的进程时才能获得。如果这个进程没有闲置处理器，其他进程不能够得到这个机会去设定一个第一个进程进展需要的程序条件。在一个多处理器系统中，其他进程在其他处理器上执行，从而为了让第一个进程从自旋锁中释放修改程序状态。

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。

6.6 解释为什么在一个多处理器系统中中断不适合同步元？
答：由于只有在防止其他进程在一个中断不能实现的处理器上执行来停止中断，中断在多处理器系统中是不够的。在对于进程能在其他处理器上执行是没有心智的，所以进程停止中断不能保证互斥进入程序状态。

6.7
6.8 服务器能够设计成限制打开连接的数量。比如，一台服务器可以在任何时候有 n 个插座连接。这 n 个连接一形成，服务器就不能接收再有进来的连接直到一个现有的连线释放。解释为什么信号量能够通过服务器限制当前连线的数量而被使用。
答：信号量初始化为允许开放式的插座连接的数量。当一个连接被接受，收购方法调用。当连接释放时，释放方法调用。如果系统道道了允许开放式的插座连接的数量，相继调用收购方法将受阻直到一个现有的连线终止，释放方法调用。

6.9 证明如果获得和释放的信号量操作没有动态地执行，那么互斥会受干扰。
答：收购操作自动递减和信号量有关的值。如果两个收购操作在信号量的值为 1 的信号量上执行，而且这两种操作不是自动执行的，那么这两个操作在进展中会递减信号量的值，从而干扰互斥。

6.10(程序，不用翻)（6.13）

6.12 证明管程和信号量是相当于它们能在执行相同类型的同步问题时使用
答：在用下列方法使用信号量时，管程可以实施。每个条件变量是由一个队列中的线程等待条件组成的。每个线程有一个和它的队列进入有关的信号量。当线程表现出等待操作时，它创造一个心得信号量（初始化为 0），附加信号量到和条件变量有关的队列中，在新创造的信号量上执行阻塞信号递减操作。
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。

6.17

假设 signal 语句只能作为一个管程中的最后一条语句出现，可以怎样简化 6.7 节所描述的实现？

如果 signal 语句作为最后一条语句出现，那么锁会使发出信号的进程转化成接受信号的进程。否则，发出信号的进程将解锁，并且接受信号的进程则需要和其他进程共同操作获得锁从而使操作继续下去。

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。

6.23

为什么 Solaris、Linux 和 Windows2000 都使用自旋锁作为多处理器系统的同步机制而不作为单处理器系统的同步机制？

Solaris,Linux 和 Windows 2000 中只有在多处理器系统才能使用自旋锁作为一个同步机制。自旋锁不适合单处理器的系统,因为打破了这一进程的自旋锁只有通过执行不同的进程才可以得到。如果这一进程不会放弃此处理器，其他进程就无法设置第一个进程所要求的程序条件，从而不能继续操作。在一个多处理器系统，其他进程执行其他处理器，从而修改程序状态从自旋锁中释放第一个进程。

6.24

在基于日志的系统中可以给事务提供支持，在相应日志记录写到稳定存储之前不能允许真正地更新数据项。为什么这个限制是必需的？

如果事务需要放弃，那么更新的数据项的值应该要恢复到原来的值。这就需要原来值的数据在进行操作之前完成更新。

6.25

证明两段锁协议能确保冲突的串行执行。

调度是指一个或多个事务的执行顺序。一个串行调度是指每个事务执行的原子调度。如果一个调度由两个不同的事务组成，通过连续的操作从这两个事务中获得相同的数据，并至少有一个 write 操作，然后有所谓的冲突。如果一个调度可以通过一系列非冲突操作的交换而转化成串行调度，那么这个调度为是冲突可串行化。 这两阶段加锁协议确保冲突串行化，因为独占锁（这是用于写操作）必须连续收购，不释放任何锁在获取（增长）的阶段。其他事务希望获得同样的锁必须等待第一个事务开始释放锁。通过要求任何锁必须首先释放所有锁，从来避免潜在的冲突。

6.26

分配一个新时间戳给已经恢复到原值的事务有什么影响？对于新进入系统进程的事务，其所赋予的时间戳是如何大于原先事务的时间戳的？

在原先事务的访问变量改变后执行事务，那么相应的事务也恢复到原先的值。如果他们没有执行此项操作（也就是说没有重复的原先事务的访问变量值），那么这些操作在适当的时候就不会受到约束。

6.27

假设数目有限的资源中的一个单一的资源型必须加以管理。进程需要一定数量的这种资源，一旦用完将释放它们。例如，许多商业软件包提供了一定数量的许可证，这表明一些应用程序可以同时运行.当应用程序启动时，许可证的计数递减。当申请终止，许可证计数递增。如果所有的许可证都在使用，那么要求启动该应用程序的申请被剥夺了。只有当现有的许可证持有人终止申请并切许可证已经返还，那么这种申请将被授予.下列程序段是用来管理一个数目有限的情况下的可用资源。最多的资源数量和一些可用的资源数量如下所示：

```
#define MAX RESOURCES 5
int available resources = MAX RESOURCES;
When a process wishes to obtain a number of resources, it invokes the
decrease count()function:
/* decrease available resources by count resources */
/* return 0 if sufficient resources available, */
/* otherwise return -1 */
int decrease count(int count) {
if (available resources < count)
return -1;
else {
available resources -= count;
return 0;
}
```

```
}
When a process wants to return a number of resources, it calls the de-
crease count()function:
/* increase available resources by count */
int increase count(int count) {
available resources += count;
return 0;
}
```

前面的程序段将会产生一个竞争的条件。如下：

a. 确定数据参与竞争

b. 当竞争的条件发生时，确定代码段的位置（或是区域）

c. 利用 Java 同步，确定竞争的条件，同时修改 decrease Count（ ）以使一个线程在没有足够的现有的资源下阻塞。

a. 确定数据参与竞争：可以利用的变量资源

b. 当竞争的条件发生时，确定代码段的位置（或是区域）：代码使现有的资源递减和代码现有资源递增的声明可以放在竞争的条件。

c. 使用信号量，确定竞争条件:使用信号量表示当前可用资源变量，并且用信号量递增和信号量递减的操作代替递增和递减的操作。

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。

7.3

一种可能以防止死锁的解决办法是要有一个单一的，优先于任何其他资源的资源。例如，如果多个线程试图访问同步对象 A•…E，那么就可能发生死锁。（这种同步对象可能包括互斥体，信号量，条件变量等），我们可以通过增加第六个对象来防止死锁。每当一个线程希望获得同步锁定给对象 A• • •E，它必须首先获得对象 F 的锁.该解决方案被称为遏制：对象 A• • •E 的锁内载对象 F 的锁。对比此方案的循环等待和 Section7.4.4 的循环等待。

这很可能不是一个好的解决办法，因为它产生过大的范围。尽可能在狭隘的范围内定义死锁政策会更好。

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。

7.5

在一个真实的计算机系统中，可用的资源和进程命令对资源的要求都不会持续很久是一致的长期（几个月）。资源会损坏或被替换，新的进程会进入和离开系统，新的资源会被购买和添加到系统中。如果用银行家算法控制死锁，下面哪

些变化是安全的（不会导致可能的死锁） ，并且在什么情况下发生？

a. 增加可用资源（新的资源被添加到系统）

b. 减少可用资源（资源被从系统中永久性地移出）

c. 增加一个进程的 Max（进程需要更多的资源，超过所允许给予的资源）

d. 减少一个进程的 Max（进程不再需要那么多资源）

e. 增加进程的数量

f. 减少进程的数量

a. 增加可用资源（新的资源被添加到系统）：这个可以在没有任何问题的情况下安全地改变

b. 减少可用资源（资源被从系统中永久性地移出）：这可能会影响到系统，并导致可能性死锁因为系统的安全性假定其拥有一定数量的可用资源

c. 增加一个进程的 Max（进程需要更多的资源，超过所允许给予的资源）：这可能会影响到系统，并可能导致死锁

d. 减少一个进程的 Max（进程不再需要那么多资源）：这个可以在没有任何问题的情况下安全地改变

e. 增加进程的数量：如果允许分配资源给新进程，那么该系统并没有进入一个不安全的状态。

f. 减少进程的数量：这个可以在没有任何问题的情况下安全地改变

7.6

假设系统中有四个相同类型的资源被三个进程共享。每个进程最多需要两个资源。证明这个系统不会死锁。

假设该系统陷入死锁。这意味着，每一个进程持有一个资源，并且正等待另一个资源。因为有三个进程和四个资源，一个进程就必须获取两个资源。这一进程并不需要更多的资源，因此当其完成时会返回其资源。

7.7 假设一个系统有 m 个资源被 n 个进程共享，进程每次只请求和释放一个资源。证明只要系统符合下面两个条件，就不会发生死锁：

a.每个进程需要资源的最大值在 1 到 m 之间

b.所有进程需要资源的最大值的和小于 m+n

Answer:使用 Section7.6.2 的术语,可以有：

a. $\__{i=1}^{n} Max_i < m + n$

b. $Max_i \geqslant 1$ for all i

Proof: $Need_i = Max_i - Alloca\ tion_i$

If there exists a deadlock state then:

c. $\__{i=1}^{n} Alloca\ tion_i = m$

Use a. to get:$\_\ Need_i + \_\ Alloca\ tion_i = \_\ Max_i < m + n$

Use c. to get:$\_\ Need_i + m < m + n$

Rewrite to get:$\__{i=1}^{n} Need_i < n$ //*符号打不出来，大家自己看答案*

这意味着存在一个Pi的进程，其Needi=0.如果Maxi>=1，那么Pi进程至少有一个资源可以释放。从而系统就不会进入死锁状态。

7.8 假设哲学家进餐问题中，筷子被摆放在桌子的中央，它们中的任何一双都可以被哲学家

使用。假如每次只能请求一根筷子，试描述一种在没有引起死锁的情况下，一个特殊的请求请求能否被满足的简单的规则，将筷子分配给哲学家。

Answer:以下规则避免了死锁：当一个哲学家发出一个需要第一根筷子的请求时，如果没有别的哲学家有两根筷子或者只留有一根筷子时，这个请求就不被允许。

7.9 与上一题目中所给的环境相同。假如现在每个哲学家请求三根筷子来吃饭，而且这种资源请求仍旧是分开发生的。试描述一种类似的在没有引起死锁的情况下，一个特殊的请求请求能否被满足的简单的规则，将筷子分配给哲学家。

Answer: 当一个哲学家发出一个需要第一根筷子的请求时，满足其情况，如果1）那个哲学家已经有2根筷子，并且还有2根筷子剩余，2) 那个哲学家已经有1根筷子，并且还有2根筷子剩余，3）最少有1根筷子剩余，并且最少有一个哲学家拥有3根筷子，4）那个哲学家没有筷子，但有2根筷子剩余，并且最少存在另外一个拥有2根筷子的哲学家放下他的筷子。

7.10 我们可以通过把数组的维度减少到 1，而从一般的银行家算法中得到一个单一资源类型的银行家算法。试通过一个例子说明对于每个资源类型，多资源类型的银行家方案不能通过单一资源类型方案的单独运用来实现。

Answer:假设一个系统有资源A,B,C和进程P_0, P_1, P_2, P_3, P_4, 并按照下图来分配

ALLOCATION			
	A	**B**	**C**
P_0,	**0**	**1**	**0**
P_1,	**3**	**0**	**2**
P_2	**3**	**0**	**2**
P_3	**2**	**1**	**1**
P_4	**0**	**0**	**2**

还需要下列资源的数量

Need			
	A	**B**	**C**
P_0	**7**	**4**	**3**
P_1	**0**	**2**	**0**
P_2	**6**	**0**	**0**
P_3	**0**	**1**	**1**
P_4	**4**	**3**	**1**

如果可利用的资源是（2 3 0），我们可以看到，进程P_0请求（0，2，0）是不能被满足的，因为它比Availiable少（2 1 0），从而导致没有一个进程可以被完成。

然而，如果我们把三种资源看做是三个独立资源类型的银行家算法，可以得到以下各表：

对于资源A

	Allocated	**Need**
P_0,	**0**	**7**
P_1	**3**	**0**
P_2	**3**	**6**
P_3	**2**	**0**
P_4	**0**	**4**

在次序P_1, P_3, P_4, P_2, P_0下，各进程可以被满足。

对于资源B

	Allocated	**Need**
P_0,	**3**	**2**
P_1	**0**	**2**
P_2	**0**	**0**
P_3	**1**	**1**
P_4	**0**	**3**

在次序P_2, P_3, P_1 , P_0, P_4下，各进程可以被满足。

对于资源C

	Allocated	**Need**
P_0,	**0**	**3**
P_1	**2**	**0**
P_2	**2**	**0**
P_3	**1**	**1**
P_4	**2**	**1**

在次序P_1, P_2, P_0,P_3, P_4下，各进程可以被满足。

我们可以看出，如果我们使用多重资源类型的银行家算法，对于进程P_0的请求（0 2 0）是无法满足的，因为它使系统处于一个不安全的状态，然而，如果我们使用单一资源类型的银行家算法，把它们看做是三个分开的资源，这个请求是允许的。同时，如果我们有多重资源类型，我们则必须使用多重资源类型的银行家算法。

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。

7.12 在死锁检测算法中，乐观假设是什么？这种假设怎样可以被违反？

Answer：乐观假设是在资源分配方面和进程请求资源的过程中，不存在任何形式的循环等待。如果在实际过程中，一个循环等待确实发生，这种假设可以被违反。

8.1 解释内部碎片和外部碎片的区别？

Answer：内部碎片是某一区域或某一页中，未被占据其位置的作业所使用的区域。直到作业完成，释放页或区域，这个空间才能被系统所利用。

8.2 考虑下面产生二进制的过程。编译器是用来为每个独立单元产生目标代码，连接编辑器是用来联合各个部分的目标单元组成一个单一的程序二进制。连接编辑器是怎样对内存地址改变指令和数据的捆绑？从编译器到连接编辑器，什么信息需要被通过，而使内存绑定连接编辑器作业比较容易？

Answer：连接编辑器不得不将分解的符号地址替换为在最终的程序二进制中，与变量相联系的实际地址。为了完成这个，单元必须追踪那些查阅到的未分解的符号指令。在连接期间，全部程序二进制中的每个单元会被分配到一序列的地址空间，当它完成时，对于未分解的符号关系，可以通过这个二进制输出，当每个另外单元包含一系列需要修复的指令时，这个二进制可以在另外单元被修复。

8.3 按顺序给出 5 个部分的内存，分别是 100KB,500KB,200KB,300KB 和 600KB，用 first-fit,best-fit 和 worst-fit 算法，能够怎样按顺序分配进程 212KB,417KB,112KB,426KB 和 426KB？哪个算法充分利用了内存空间？

Answer：

a. First-fit:

b. 212K is put in 500K partition

c. 417K is put in 600K partition

d. 112K is put in 288K partition (new partition 288K = 500K − 212K)

e. 426K must wait

f. Best-fit:

g. 212K is put in 300K partition

h. 417K is put in 500K partition

i. 112K is put in 200K partition

j. 426K is put in 600K partition

k. Worst-fit:

l. 212K is put in 600K partition

m. 417K is put in 500K partition

n. 112K is put in 388K partition

o. 426K must wait

Best-fit: 算法充分利用了内存空间。

8.4 在运行过程中，许多系统允许程序分配更多的内存给它的地址空间。在程序堆中的数据分配是这种分配方式的一个实例。在下面的方案中，为了支持动态内存分配的要求是什么？
a.连续内存分配 b.纯段式分配 c.纯页式分配

Answer：a. 连续内存分配:当没有足够的空间给程序去扩大它已分配的内存空间时，将要求重新分配整个程序。

b. 纯段式分配：当没有足够的空间给段去扩大它的已分配内存空间时，将要求重新分配整个段。

c. 纯页式分配：在没有要求程序地址空间再分配的方案下，新页增加的分配是可能的。

8.5 比较在主存组织方案中，连续内存分配，纯段式分配和纯页式分配在下面问题中的关系。
a.外部碎片 b.内部碎片 c.通过进程分享代码的能力

Answer：连续内存分配会产生外部碎片，因为地址空间是被连续分配的，当旧进程结束，新进程初始化的时候，洞会扩大。连续内存分配也不允许进程共享代码，因为一个进程的虚

拟内存段是不被允许闯入不连续的段的。纯段式分配也会产生外部碎片，因为在物理内存中，一个进程的段是被连续放置的，以及当死进程的段被新进程的段所替代时，碎片也将会产生。然而，段式分配可以使进程共享代码；比如，两个不同的进程可以共享一个代码段，但是有不同的数据段。纯页式分配不会产生外部碎片，但会产生内部碎片。进程可以在页granularity中被分配，以及如果一页没有被完全利用，它就会产生内部碎片并且会产生一个相当的空间浪费。在页granularity，页式分配也允许进程共享代码。

8.6 在一个页式分配系统中，为什么一个进程不被允许进入它所不拥有的内存？操作系统怎么能被允许进入其它内存？它为什么应当可以或不可以进入？

Answer：地址在页式分配系统上是一个逻辑页号和一个偏移量。在逻辑页号的基础上产生一个物理页号，物理页通过搜索表被找到。因为操作系统控制这张表的内容，只有在这些物理页被分配到进程中时，它可以限制一个进程的进入。一个进程想要分配一个它所不拥有的页是不可能的，因为这一页在页表中不存在。为了允许这样的进入，操作系统只简单的需要准许入口给无进程内存被加到进程页表中。当两个或多个进程需要交换数据时，这是十分有用的。------它们只是读和写相同的物理地址（可能在多样的物理地址中）。在进程内通信时，这是十分高效的。

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。

8.8 在许多系统中的程序二进制的一般构造如下：代码被存储在较小的固定的地址中，比例0。代码段后紧跟着被用来存储程序变量的数据段。当这个程序开始运行，栈被分配到虚地址空间的另一个端末尾，并被允许向较低的虚地址扩张。上述结构在下列方案中具有什么意义: a.连续内存分配 b.纯段式分配 c.纯页式分配

Answer：1）当程序开始运行时，连续内存分配要求操作系统给程序分配最大限度的虚地址空间。这可能造成比进程所需要的实际内存大很多。2）纯段式分配，在程序开始运行时，给每个段分配较小的空间，而且能随着段的扩展而扩大，这就给操作系统提供了灵活性。3）纯页式分配在一个进程开始运行时，就不需要操作系统给进程分配最大的虚地址空间。当一个程序需要扩展它的堆或栈时，它需要分配一个新的页，但是相关的页表入口被提前分配了。

8.9 考虑一个分页系统在内存中存储着一张页表。a.如果内存的查询需要 200 毫秒，那么一个分页内存的查询需要多长时间？b.如果我们加上相关联的寄存器，75%的页表查询可以在相关联的寄存器中找到，那么有效的查询时间是多少？（假设如果入口存在的话，在相关的寄存器中找到页表入口不花费时间）

Answer：a.400 毫秒：200 毫秒进入页表，200 毫秒进入内存中的字

b.有效进入时间=0.75*200 毫秒+0.25*400 毫秒=250 毫秒

8.10 为什么有时候段式分配存储与页式分配存储可以联合成一种方案？

Answer：段式存储与页式存储经常结合在一起是为了提高它们两个中的每一个存储方式。当页表变的十分大时，段式存储是十分有用的。一大段连续的页表是不习惯被分解成为一个以 0 为段表地址的单一段表入口。分页的段式存储句柄有一个非常大的段的时侯，就需要很多时间来进行分配。通过把段分页，我们降低了由于外部碎片而造成的内存浪费，而且也简化了分配。

8.11 解释为什么当共享一个使用段式存储的 reenteant 单元时比纯页式存储时这样做要来的容易？

Answer：因为段式存储是以内存的逻辑共享为基础的，而不是物理的，任何大小的段在段

表中，被每个只具有一个入口的用户所共享。而分页必须在页表中对每个被共享的页有相同的入口。

8.13 问:页表分页的目的是什么?

答：在某些情况下,分页的页表可以变得足够大，可以简化内存分配问题（确保全部可以分配固定大小的网页，而不是可变大小的块），确保当前未使用的部分页表可以交换。

8.14 问：考虑分层分页方案，使用 VAX 架构。当用户程序执行一个内存装载程序时，有多少个内存操作要执行？

答：当一个内存装载程序完成时，有三个内存操作可以完成，一个是说明能够被打到的页表的位置。第二个是页表进入自己。第三个是现实的内存装载操作。

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。

8.16 问：假设 Intel 的地址转换方案如图 8.22 所示

A．描述 Intel80836 将逻辑地址转换成物理地址所采用的所有步骤。

B．使用这样复杂的地址转换硬件对硬件系统有什么好处？

C．这样的地址转换系统有没有什么缺点？如果有，有哪些？如果没有，为什么不是每个制造商都使用这种方案。

答：A。选择符是段描述符表的标志，段描述符的结果加上原先的偏移量构成页表，再加上目录、偏移量构成页表，构成线性地址。这个目录是页目录的标志。目录项选择页表，页表域是页表的索引。页表项再加上偏移量，构成物理地址。

B.这样的页表转换系统提供了灵活性，允许大多数操作系统在硬件上执行内存工具，而不是实施部分硬件和一些软件。因为，它可以在硬件上实施，更有效率（内核更简单）

C.地址转换在查找多样表时更花时间，缓存帮助，仍会导致缓存丢失。

9.1 问。举一个例子，IBM360/370 的资源和目的地区重叠时说明，（MVC）重新启动移动块的问题。

答：假设页面边缘为 1024，移动空间从资源区 800：1200 到目标区 700：1100，假设当页表在 1024 边缘发生故障访问错误，这时候的位置 800：923 已覆盖新的值，因此，重新启动区块移动指令会导致在 800：923 到 700：823 之间复制新的值，而这是不正确的。

9.2 问：考虑支持请求页面调度的硬件需求。

答：对于每一个内存访问操作，页表需要检查相应的页表驻留与否和是否计划已经读取或写入权限访问页面，一个 TLB 可以作为高速缓存和改善业绩的查询操作。

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。

9.4 问：某个计算机给它的用户提供了 2^{32} 的虚拟内存空间，计算机有 2^{14}B 的物理内存，虚拟内存使用页面大小为 4094B 的分页机制实现。一个用户进程产生虚拟地址 11123456，现在说明一下系统怎么样建立相应的物理地址，区分一下软件操作和硬件操作。（第六版有翻译）

答：该虚拟地址的二进制形式是 0001 0001 0001 0010 0011 0100 0101 0110。由于页面大小为 212，页表大小为 220，因此，低 12 位的“0100 0101 0110 ”被用来替换页（page），而前 20 位“0001 0001 0001 0010 0011”被用来替换页表(page table)。

9.5 假设有一个请求调页存储器，页表放在寄存器中：处理一个页错误，当有空的帧或被置换的页设有被修改过时要用 8ms，当被置换的页被修改过明用 20ms，存储器访问时间为 100ns。

假设被置换的页中有 70％被修改过，有效访问时间不超过 200ns 时最大可接受的页错误率是多少？（第六版有翻译）

答：0.2 _sec = $(1 - P) \times 0.1$ _sec + $(0.3P) \times 8$ millisec + $(0.7P) \times 20$ millisec

$0.1 = -0.1P + 2400\ P + 14000\ P$

0.1 _ 16,400 P

P _ 0.000006

9.6 问：假设正在监测的速度指针在时钟算法（表明侯选页面更换），如果发生以下行为，系统会怎么样？

A.指针快速前进　B.指针移动缓慢

答：如果指针运行快，则该程序同时访问大量页面，当指针在对应的页面上清理与检查时，这是最可能发生的，因此不能被取代，这样做的结果是受害页面被发现之前，扫描很多页面。如果指针运行慢，在虚拟内存找寻候选页表更换极为有效，表明许多常驻页面不会被窃取。

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。

9.8 问：讨论在哪一种情况下，MFU（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：考虑可容纳 4 页的内存：1 2 3 4 4 4 5 1，MFU 算法会用 5 替换 4，而 LRU 算法刚用 5 替换 1，实践中不可能发生，对于串：1 2 3 4 4 4 5 1，LRU 算法做得更正确。

9.9 问：在 VAX/VMS 系统对驻留页采用先进先出算法，在空闲帧给最近最少使用页面，假设在空闲帧使用 LRU 算法，回答下列问题

A.如果页表出错和页面不存在空闲帧如何产生新空间给亲要求页面

B.如果页面出错和页面存在空闲帧，如何驻留页面，空闲帧怎么样分配给新要求页表。

C.如果驻留页面只有一个，系统如何决定

D.如果没有空闲帧，系统如果决定

答：A.在这种情况下，空闲帧中的一个页面被替换到磁盘上，为驻留页面创建一个空间，再转移到空闲帧里，浏览页面时，又被称动到驻留页面上。

B.引进一套驻留页面，并将页面搬进空闲帧

C.系统在空闲帧里使用页置换法通常是 LRU 算法

D.系统进行 FIFO 算法

9.10 问：假设一个具有下面时间度量利用率的请求调页系统：

CPU 利用率 20％，分页磁盘 97.7％，其他 I/O 设备，5％

说明下面哪一个（可）能提高 CPU 的利用率，为什么？

A 安装一个更快的 CPU

B 安装一个更大的分页磁盘

C 提高多道程序设计程序

D 降低多道程序设计程度

E 安装更多内存

F 安装一个更快的硬盘，或对多个硬盘使用多个控制器

G 对页面调度算法添加预取页

H 增加页面大小。

答：该系统显然花费了许多时间进行分页，显示过度分配的内存，如果多级程序水平减少驻地进程，将页面错误变少和提高 CPU 利用率。另一种方式来提高利用率是获得更多的物理内存或更快的分页鼓。

ABC 都不行，D 可以

E.可能提高 CPU 利用率为更多页面保持驻地，而不需要分页或磁盘。

F.另一个改进，因为磁盘的瓶颈是删除更快的响应，和更多的磁盘容量，CPU 将会获得更多的数据传输速度

G.CPU 将获得更快的数据传输率，所以更多地被使用。如果分页服从预调（即一些访问顺序）这只是一个方面。

H.增加页面大小将导致减少页面错误，如果数据进行是随机的，则分页可以随之，因为较少页面可保存在内存上，更多的数据转移到页面错误 上，这种 变化可以减少 CPU 利用率或者增加 CPU 利用率。

9.11 假设一台机器使用一级间接引用方法提供可以访问内存位置的指令。当一个程序的所有页未驻留，程序的第一条指令是一个间接内存 load 操作时，将会出现什么页错误？当操作系统正在使用一个单进程帧分配技术，只有两个页被分配至此进程时，将会发生什么？

Answer:

出现以下页错误：访问指令的页错误，访问包含一个指向目标内存位置指针的内存位置的页的错误，访问目标内存位置的页错误。第三页置换包含指令的页，操作系统将产生三个页错误。如果需要再次取出指令，重复被陷指令，那么，页错误将无限期地继续下去。如果指令在寄存器中缓存，那么将能在第三页错误后完全执行。

9.12 假设你的置换策略（在分页系统中）是有规律地检查每个页并将最近一次检测后没有再被引用的页丢弃。与 LRU 或二次机会置换算法相比，使用这种策略有哪些好处和坏处？

Answer:

这种算法可以靠引用位的使用来实现。每次检查过后，置位为 0；如果页被引用，置位为 1。然后，该算法将从自上次检查后未使用过的页中选择任意页来置换。

这种算法的优点是算法比较简单——只需保持一个引用位。这种算法的缺点是，只能使用很短的时间帧来决定是否置换一页，从而忽略了局部性。例如，一个页可能是一个进程工作集合的一部分，但因为自上次检查后未被引用而被置换。（即不是所有工作集合中的页可以在检查之间被引用）

9.13 一个页面置换算法应使发生页错误的次数最小化。怎样才能通过将使用频率高的页平均分配到整个内存而不只是竞争少数几个页帧页来达到这种最小化。可以对每个页帧设置一个计数器来记录与此帧相关的页数。那么当置换一

个页时，就可以查找计数器值最小的页帧

Answer:

a.定义一个页面置换算法解决问题：

Ⅰ.计数器初始值——0；

Ⅱ.计数器值增加——每当新的一页与此帧相关联；

Ⅲ.计数器值减少——每当与此帧相关联的一个页不再需要；

Ⅳ.怎样选择要被置换的页——找到带有最小计数器值的帧。使用先进先出算法解除其关系

b.14 个页错误

c.11 个页错误

9.14 假设一个请求调页系统具有一个平均访问和传输时间为 20ms 的分页磁盘。地址转换是通过在主存中的页表来进行的，每次内存访问时间为 1μs。这样，每个通过页表进行的内存引用都要访问内存两次。为了提高性能，加入一个相关内存，当页表项在相关内存中时，可以减少内存引用的访问次数。

假设 80%的访问发生在相关内存中，而且剩下中的 10%（总量的 2%）会导致页错误。内存的有效访问时间是多少？

Answer:

有效访问时间= (0.8) × (1 μsec)+ (0.1) × (2 μsec) + (0.1) × (5002 μsec)

= 501.2 μsec

= 0.5 millisec

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。

9.16 一个进程可能有两个工作集合吗，一个代表数据，另一个代表代码？请解释。

Answer:

是的，事实上，许多处理器因为这个原因提供两个 TLB。举个例子，一个进程访问的代码可长时间地保留同样的工作集合。然而，代码访问的数据可能改变，这样为工作集合的数据访问映射了一个改变。

9.17 假设使用参数 Δ 定义工作集合模型下的工作集合窗口。设置 Δ 为一个较小值，其表示页错误频率和系统中当前正在执行的活动页（非暂停的）进程数量，则影响如何？当设置 Δ 为一个非常大的值呢？

Answer:

当设置 Δ 为一个较小的值，那么有可能低估一个进程的驻留页集合，允许安排一个进程，即使其所需的所有页未驻留。这可能导致大量的页错误。当设置 Δ 为较大的值，那么将高估一个进程的驻留集合，这可能阻止许多进程被安排，尽管他们需要的页驻留。然而，一旦一个进程被安排，高估驻留集合后就不可能

产生页错误

9.18 假设有一个初始值为 1024 kB 的段，使用伙伴(Buddy system)系统分配其内存。以图 9.27 为指导，画出树来说明下列内存请求是如何分配的：

- **请求 240 字节**
- **请求 120 字节**
- **请求 60 字节**
- **请求 130 字节**

下一步，为下列内存释放修改树。只要有可能便执行接合：

- **释放 250 字节**
- **释放 60 字节**
- **释放 120 字节**

Answer:

伙伴(Buddy system)系统进行了下列分配：为 240 字节的请求分配一个 256 字节段。为 120 字节的请求分配一个 128 字节段，为 60 字节的请求分配一个 64 字节段，为 130 字节的请求分配一个 256 字节段。分配后，下列大小的段是可用的：64 bytes，256 bytes，1K, 2K, 4K, 8K, 16K, 32K, 64K, 128K, 256K, 和 512K。内存释放后，仅剩包含 130 字节数据的 256 字节段在使用。下列段将是空闲的：256 bytes, 512 bytes, 1K, 2K, 4K, 8K, 16K, 32K, 64K, 128K, 256K, 和 512K。

9.19 slab 分配算法为每个不同的对象类型使用一个单独的缓存。假设每一个对象类型都有一个缓存，试解释，为什么这不与多个 CPU 较好地协调。怎么做才能解决这个可扩展性问题？

Answer:

这一直是 slab 分配存在的一个问题——多 CPU 存在时的较差可扩展性。这个问题产生于必须锁定正被访问的全局缓存。这影响多处理器系统的序列缓存访问。Solaris 操作系统凭借引进一个单 CPU 缓存，而非一个单一的全局缓存解决了这个问题

9.20 假设一个为其进程分配不同大小页的系统。这种页面调度方法有何优点？虚拟内存系统提供此功能时进行了哪些修正？

Answer:

程序可能有大量的代码段或使用大型的数组作为数据。这部分程序可被分配于较大的页，从而减少与页表相关的内存开销。考虑到不同的内存大小，虚拟内存系统就必须保持多个不同大小的空闲页链表，为了地址翻译也需要有复杂的代码。

10.1 假设有一个文件系统，它里面的文件被删除后，当连接到该文件的链接依然存在时，文件的磁盘空间会再度被利用。如果一个新的文件被创建在同一个存储区域或具有同样的绝对路径名，这会产生什么问题？如何才能避免这些问题？

Answer:

令旧的文件为 F1，新文件为 F2。用户希望通过存在的链接进入 F1，实际上将进入 F2。注意，使用的是文件 F1 的访问保护，而不是 F2 的。这个问题可以通过确保所有被删除文件的链接也被删除来避免。可以从几个方面来完成：

a. 保存一份文件所有链接的列表，删除文件时，删除每一个链接
b. 保留链接，当试图访问一个已删除文件时删除他们
c. 保存一个文件引用清单（或计数器），只有在文件所有链接或引用被删除后，删除该文件

10.2 打开文件表被用以保持当前打开文件的信息，操作系统应该为每个用户保持一个单独的表吗？或者只是保持一个包含当前所有用户访问文件的引用的表？如果两个不同程序或用户访问同样的文件，在打开文件表中应包含单独的条目吗？

Answer:

保持一个中央的打开文件表，操作系统可以执行下列操作，否则不可执行：假设一个当前有一个或一个以上进程访问的文件。如果该文件被删除，那么应该直到所有正在访问文件的进程关闭它时，它才能从磁盘上删除。只要有正在访问文件的进程数目的集中核算，该检查就可以执行。另一方面，如果两个进程正在访问该文件，则需要保持两个单独的状态来跟踪当前位置，其中部分文件正被两个进程访问。这就要求操作系统为两个进程保持单独的条目。

10.3 一个提供强制锁，而非使用由用户决定的咨询锁的进程有何优点和缺点？

Answer:

在许多情况下，单独的程序可能愿意容忍同时访问一个文件，而不需要获得锁，从而确保文件的相互排斥。其他程序结构也可以确保相互排斥，如内存锁；或其他同步的形式。在这种情况下，强制锁将限制访问文件的灵活性，也可能增加与访问文件相关的开销。

10.4 在文件的属性中记录下创建程序的名字，其优点和缺点是什么？（在 Macintosh 操作系统中就是这样做的）

Answer:

记录下创建程序的名字，操作系统能够实现基于此信息的功能（如文件被访问时的程序自动调用）。但它会增加操作系统的开销，需要文件描述符的空间。

10.5 有些系统当文件第一次被引用时会自动打开文件，当作业结束时关闭文件。论述这种方案与传统的由用户显式地打开和关闭文件的方案相比有什么有点和缺点?

Answer:

文件的自动打开和关闭免除了用户对这些功能的调用，从而使它更方便用户；但它比显式打开和关闭需要更多的开销

10.6 如果操作系统知道某一应用将以顺序方式访问文件数据，将如何利用此信息来提高性能？

Answer:

当访问一个块时，文件系统可以预取随后的块，预计未来对于这些块的要求，这种预取优化将减少未来进程将经历的等待时间。

这个预取的优化将会为未来的要求减少等候所经历的时间。（10.6 最后一句翻译）

10.7举一个应用程序的例子，它能够受益于操作系统支持的随机存取，以建立索引的档案。

答：一个应用程序，它维持的一个数据库的条目可以受益于这种一种支持：举个列子, 如果某程序是维护一个学生数据库，则访问的数据库不能被任何预先确定的访问模式模拟，这种获得记录是随机的，而且该记录的定位，如果作业系统是提供某种形式的树为基础的指数，将会更有效。

10.8讨论支持联系档案，两岸装入点的优点和缺点（即链接文件指的是文件在不同体积存储） 。

答：其优点是，有更大的透明度，也就是说，用户并不需要知道装入点和建立联系的所有情景。但缺点是文件系统包含的链接可能会展开而安装的文件系统包含目标文件可能不会，因此，在这种情况下不能提供透明的访问该文件，错误的条件会使该用户的联系是一条走不通的链接，而且链接确实跨越了文件系统的界限。

10.9 有些系统文件提供文件共享时候只保留文件的一个拷贝，而另外的一个系统则是保留多个拷贝，对共享文件的每一个用户提供一个拷贝，论述这种方法的相对优点。

答：在一个单一的复制，同时更新了一个文件可能会导致用户获得不正确的信息，文件被留在了不正确的状态. 随着多份拷贝，它会浪费存储而且各种副本可能不一致。

10.10讨论交往远程文件系统（存储在文件服务器）从一套不同的失败语义相关的本地文件系统的优点和缺点。

答：其优点是，如果在获得的文件存储在一个远程文件系统认识到它发生了一个错误，应用程序可以在处理故障状况时候提供一个更加智能化的方式，，而举例来说，一个文件打开文件可能简单的就失败了，而不是简单地挂在访问远程文件的一个失败的服务器和应用程序可以尽可能以最好的方式处理失败; 如果运行只是外挂起，那么整个应用程序应该外挂起，这是不可取的。然而，由于在失败语义缺乏统一以致由此导致应用程序代码更复杂。

10.11什么是影响一致支持共享访问这些存储在远程文件系统的文件的UNIX语义的含义？

答：UNIX的一致性语义需要更新文件立即提供给其他进程。支持这种语义的共享文件远程文件系统可能会导致以下的低效率: 所有更新的客户，必须立即上报文件服务器，而不是批处理（如果更新到一个临时文件将会被忽略，），而且更新的都必须送交到文件服务器向客户的数据进行高速缓存，使得立即构造成更多的联系。

11.1 试想一个文件系统，采用改进的连续分配计划的支持程度. 其档案收集的每个程度相当于毗连的区块。一个关键的问题是在这样的系统存在一定的变异的大小程度。如下计划有哪些优点和缺点？

a. 所有程度都是同样的尺寸，大小是预先确定的。

b. 程度上可以任意大小和动态分配。
c. 程度上可以几个固定的尺寸，这些尺寸是预先确定的。
答：如果所有的程度上是相同的尺寸，大小是预先确定的，那么它简化了块分配计划。一个简单的位图或空闲状态的名单程度就足够了。如果程度上可以任意大小和动态分配，则更复杂的分配计划是必需的。这可能很难找到一个程度的适当规模而且它可能会存在外部碎裂.。人们可以使用好友系统分配器讨论了前几章设计一个适当的分配器。当程度上可以几个固定的尺寸，而这些尺寸可以预定，人们必须保持一个单独的位图或为每个名单给以可能大小.. 这项计划是中级复杂性和中级灵活性的比较。

11.2使用FAT链合作区块的档案来进行变化相联系的分配有哪些优势？
答：它的优势是，在访问块是储存在中间的文件时候，在 FAT 里跟踪指针可以决定它的位置，而不是访问所有个别区块中的档案顺序的方式找到指针的目标块。
通常情况下，大多数的 FAT 可缓存在存储器里 ，因此，指针可以通过记忆体确定，而不用通过磁盘块。

11.3假设有一个系统，它的空闲空间保存在空闲空间链表中：
a. 假设指向空闲空间链表的指针丢失了，系统能不能重建空闲空间链表，为什么？
b. 试想一个文件系统类似 UNIX 的使用与分配索引，有多少磁盘 I / O 操作可能需要阅读的内容，一个小地方的档案在 a/b/c？假设此时没有任何的磁盘块，目前正在缓存。
c.设计一个方案以确定发生内存错误时候总不会丢失链表指针
答：
a.为了重建自由名单，因此有必要进行“垃圾收集” 。这就需要搜索整个目录结构，以确定哪些网页已经分配给了工作。这些剩余的未分配的网页可重新作为自由空间名单。
b.在自由空间列表里指针可存储在磁盘上，但也许在好几个地方。
c.指针可以存储在磁盘上的数据结构里或者在非挥发性RAM （ NVRAM. ）

11.4有些档案系统允许磁盘存储将分配在不同级别的粒度。举例来说，一个文件系统可以分配4 KB的磁盘空间作为单一的一个4字节的块或8个512字节的块。我们如何能利用这种灵活性来提高性能？对自由空间管理做出哪些修改以支持这一功能?
答：此项计划将减少内部分裂。如果文件是5字节，然后可以分配4 KB的区块和两个毗连的512字节的块。除了维持一个位图的自由块，一个目前正在使用的区块内也将保持额外的状态。当所有的分块成为空闲时候，该分配器将不得不审查这笔额外分配状态分块和凝聚的分块，以获取更大的块。

11.5讨论一旦难以维持的一致性的系统导致计算机崩溃，如何性能优化的文件系统。
答：由于延迟更新数据和元数据可能出现最主要的困难。在希望同样的数据可能被更新时候更新可能会推迟，或更新的数据可能是临时性的，而且在不久后可能会被删除。
但是，如果系统崩溃，则不必致力于延迟更新，文件系统的一致都将被破坏。

11.6 设想一个在磁盘上的文件系统的逻辑块和物理块的大小都为512B。假设每个文件的信息已经在内存中，对3种分配方法（连续分配，链接分配和索引分配），分别回答下面的问题：
A，逻辑地址到物理地址的映射在系统中怎么样进行的？（对于索引分配，假设文件总是小于512块长）
B，假设现在处在逻辑块10（最后访问的块是块10），限制想访问块4，那么必须从磁盘上读

多少个物理块）
答：设想Z是开始文件的地址（块数），
a.毗连。分裂逻辑地址由512的X和Y所产生的份额和其余的分别。
1：将X加入到Z获得物理块号码。 Y是进入该区块的位移。
2.：1
b.联系。分裂逻辑地址由511的X和Y所产生的份额和其余的分别。
1.：找出联系名单（将X + 1块） 。 Y + 1是到最后物理块的位移
2.：4
c.收录。分裂的逻辑地址由512的X和Y所产生的份额和其余的分别。
1.：获得该指数块到内存中。物理块地址载于该指数在所在地块10， Y是到理想的物理块的位移。
2.：2

11.7 一个存储设备上的存储碎片可以通过信息再压缩来消除，典型的磁盘设备没有重新定位或基址寄存器（像内存被压缩时用的一样），怎样才能重定位文件呢？给出为什么文件再压缩和重定位常常被避免使用的3个原因。
答：移动文件二级存储涉及大量开销—数据块都必须读入内存，并写回了他们的新地点. 此外，移动登记册只适用于连续的文件，和许多磁盘文件不是一种顺序。相同的，许多新的档案不需要连续的磁盘空间; 如果是以逻辑顺序区块保持的磁盘系统，甚至连续档案可用于非相邻块之间的联系。

11.8在何种情况下会使用内存作为RAM磁盘更加有用而不是用它作为一个磁盘高速缓存？
答：在用户（或系统）确切地知道什么样的数据情况下将是必要的。缓存的算法为基础的，而RAM磁盘是用户导向。

11.9试想增加下列远程文件访问协议。每个客户端保持一个名称缓存，缓存翻译的文件名，以对应相应的文件句柄。哪些问题我们在执行名称缓存应该考虑到？
答：其中的一个问题是保持一致的名称缓存。如果缓存条目变得不一致，则应该更新或应检测不一致的地方。如果发现不一致后，然后应该有一个兜底机制，以确定新的翻译名称。此外，另一个相关的问题是是否有名称查找执行的一个组成部分的时间为每个子目录的路径，或是否是在一个单一的射击服务器。如果是运行的一个组成部分的时间，则在用户端可能会获得更多的资料翻译所有的中间目录。另一方面，它增加了网络流量作为一个单一的名称查找原因序列部分名称查找。

11.10解释为什么记录元数据更新能确保文件系统从崩溃中恢复过来？
答：对于档案系统崩溃后的重恢复，它必须是一致的或必须能够取得一致。因此，我们必须证明，测井数据在不断更新的档案系统中是一致的或有能力达成一致的状态。对于一个不一致的文件系统，元数据必须不完全是书面的或在文件系统的数据结构是错误的。测井数据的写入的是一个连续的记录。

完整的交易是在它被转移到文件系统结构之前被写入的，如果在文件系统数据更新时系统崩溃，根据记录器里的信息，这些更新仍能完成。因此可以说，记录器确保了文件系统的变化的完成（不论是在系统崩溃前还是之后）。因为信息是连续写入到记录器中的，因此变化的

命令保证是正确的。如果改变未完全写入记录其中，它便未改变文件系统结构，便会被丢弃。因此，通过数据记录重播，结构是相容的或接近相容的。

11.11 设想下面的备份方法：
第一天：将所有的文件从磁盘拷贝到备份介质。
第二天：将从第一天开始变化的文件拷贝到另一介质。
第三天：将从第一天开始变化的文件拷贝到另一介质。
与 11.7.2 小节中的方法不同。并非将所有从第一次备份后改变的文件都拷贝。与小节中的方法相比有什么优点？有什么缺点？恢复操作是更简单还是更复杂了？为什么？

【答】还原比较容易，因为你可以去备份磁带上，而不是充分磁带。没有中间磁带需要读取。更多的磁带被用作多个文件变化。

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。

12.2 假设一个错哦盘驱动器有 5000 个柱面，从 0 到 4999，驱动器正在为柱面 143 的一个请求提供服务，且前面的一个服务请求是在柱面 125.按 FIFO 顺序，即将到来的请求队列是
86，1470，913，1774，948，1509，1022，1750，130
从现在磁头位置开始，按照下面的磁盘调度算法，要满足队列中即将到来的请求要求磁头总的移动距离（按柱面数计）是多少？

a. FCFS
b. SSTF
c. SCAN
d. LOOK
e. C-SCAN

【答】a. FCFS 的调度是 143 ， 86 ， 1470 ， 913 ， 1774 ， 948 ， 1509 ， 1022 ， 1750 ， 130 。总寻求距离是 7081 。

b. SSTF 的调度是 143 ， 130 ， 86 ， 913 ， 948 ， 1022，1470，1509，1750，1774。总寻求距离是 1745。

c. SCAN 的调度是 143 ， 913 ， 948 ， 1022，1470，1509，1750，1774 ， 4999 ， 130 ， 86 。总寻求距离是 9769 。

d. LOOK 的调度是 143 ， 913 ， 948 ， 1022，1470，1509，1750，1774，130 ， 86 。总寻求距离是 3319 。

e. C-SCAN 的调度是 143 ， 913 ， 948 ， 1022 ， 1470 ， 1509 ， 1750 ， 1774 ， 4999 ， 86 ， 130 。总寻求距离是 9813 。

f. C-LOOK 的调度是 143 ， 913 ， 948 ， 1022 ， 1470 ， 1509 ， 1750 ， 1774 ， 86 ， 130 。总寻求距离是 3363 。

12.3 。。。。。

【答】d. （ 65.20 -3 1.52） / 6 5.20= 0 .52。百分比加速超过营运基金 FCFS 是 52 ％ ，对争取时间。如果我们的开销包括转动延迟和数据传输，这一比例将加速减少。

12.4 假设题 12.3 中的磁盘以 200RPM 速度转动。
A:磁盘驱动器的平均旋转延迟时间是多少？
B:在 a 中算出的时间里，可以寻道多少距离？

【答】A 7200 转 120 轮换提供每秒。因此，充分考虑轮换 8.33 毫秒，平均旋转延迟（一个半轮换）需要 4.167 毫秒。

B 求解：吨= 0.7561 + 0.2439 √ L
t = 4.167
使 L 为 195.58 ，
因此，在一个平均旋转延迟的期间我们可以寻找超过 195 个轨道（约 4 ％的磁盘）。

12.6　假设对于同样均衡分发的请求，比较 C-SCAN 和 SCAN 调度的性能。考虑平均响应时间（从请求到达时刻到请求的服务完成之间的距离），响应时间的变化程度和有效带宽，问性能对于相关的寻道时间和旋转延迟的依赖如何?

【答】略

12.7　请求往往不是均衡分发的。例如，包含文件系统 FAT 或索引结点的柱面比仅包含文件内容的柱面的访问频率要高。假设你知道 50%的请求都是对一小部分固定数目柱面的。
A:对这种情况，本章讨论的调度算法中有没有那些性能特别好?为什么？
B:设计一个磁盘调度算法，利用此磁盘上的“热点”，提供更好的性能。
C:文件系统一般是通过一个间接表找到数据块的，像 DOS 中的 FAT 或 UNIX 中的索引节点。描述一个或更多的利用此类间接表来提高磁盘性能的方法。

【答】a.　SSTF 将采取情况的最大的优势。如果提到的“高需求”扇区被散置到遥远的扇区，FCFS 可能会引起不必要的磁头运动。

b.　以下是一些想法。将热数据放置于磁盘的中间附近。修改 SSTF，以防止饥饿。如果磁盘成为闲置大概 50 毫秒以上，则增加新的政策，这样操作系统就会对热点地区产生防患未然的寻求，因为接下来的要求更有可能在那里。

c.　主要记忆体缓存数据，并找到一个与磁盘上物理文件密切接近的数据和元数据。（ UNIX 完成后者的目标分配数据和元数据的区域称为扇区组。 ）

12.8　一个 RAID 1 组织读取请求是否可以比 RAID 0 组织实现更好的性能（非冗余数据带）？如果是的话，如何操作？

【答】是的，一个 RAID 1 级组织在阅读要求方面可以取得更好的性能。当执行一个读操作，一个 RAID 1 级系统可以决定应访问哪两个副本，以满足要求。这种选择可能是基于磁盘头的当前位置，因此选择一个接近目标数据的磁盘头可以使性能得到优化。

12.9 试想一个 RAID 5 级的组织，包括五盘，以
平等套 4 次盖帽 4 个磁盘存储的第五盘。该会有多少区块被访问以履行下列？
a. 一个区块数据的写入
b.多个毗连区块数据的写入？

【答】 a) 写一个块的数据需要满足以下条件：奇偶块的读取，存储在目标块中旧的数据的读取，基于目标区块上新旧内容的不同的新的奇偶的计算，对奇偶块和目标块的写入。

b) 假设 7 毗连区块在 4 块体边界开始。一个 7 个毗连区块的数据的写入可以以 7 个毗连区块的写入形式进行，写奇偶块的首个 4 块，读取 8 块，为下一组 4 块计算奇偶以及在

磁盘上写入相应的奇偶区块。

12.10 对比达到一个 RAID 5 级的组织 与所取得的一个 RAID 1 级安排的吞吐量如下：
a:在单一块上读取操作
b:在多个毗连区块读取操作

【答】
a)吞吐量的数额取决于在 RAID 系统里磁盘的数量。一个 RAID 5 由为每套的奇偶块的四张块延长的 5 个磁盘所组成，它可能同时支持四到五次操作。一个 RAID 1 级，包括两个磁盘可以支持两个同步行动。当然，考虑到磁盘头的位置，RAID 级别 1 有更大的灵活性的副本块可查阅，并可以提供性能优势。

b)RAID 5 为访问多个毗连区块提供更大的带宽，因为邻近的区块可以同时访问。这种带宽的改善在 RAID 级别 1 中是不可能的。

12.11 对比用一个 RAID 级别写入作业与用一个 RAID 级别 1 写入作业取得的业绩。

【答】RAID 级别 1 组织仅根据当前数据镜像便可完成写入， 另一方面，RAID 5 需要在阅读之前读取基于目标快新内容更新的奇偶块的旧内容。这会导致 RAID 级别 5 系统上更多间接的写操作。

12.12 假设您有一个混合组成的作为 RAID 级别 1 和 RAID 级别 5 的磁盘配置。假设该系统在决定该组织的磁盘用于存储特别是文件方面具有灵活性。哪个文件应存放在 1 级的 RAID 磁盘并在 5 级的 RAID 磁盘中用以优化性能？

【答】经常更新的数据需要存储在磁盘阵列 1 级的磁盘，而更经常被读取或写入的数据，应存放在 RAID 5 级的磁盘。

12.14 有没有一种方法可以实现真正的稳定存储？
答：真正的稳定存储永远不会丢失数据。最基本的稳定存储技术就是保存多个数据的副本，当一个副本失效时，可以用其它的副本。但是，对于任何一种策略，我们都可以想象一个足够大的灾难可能摧毁所有的副本。

12.15 硬盘驱动器的可靠性常常用平均无故障时间（MTBF）来描述。虽然称之为时间，但经常用设备小时来计算无故障时间。
a.如果一个大容量磁盘有 1000 个驱动器，每个的 MTBF 是 750 000 小时，一下哪个描述能最好地体现该大容量磁盘出错的概率？每千年一次，每百年一次，每十年一次，每年一次，每月一次，每周一次，每天一次，每小时一次，每分钟一次，还是每秒一次？
b.根据死亡统计资料，平均来说，20 至 21 岁的美国人死亡的概率是千分之一。推断出 MTBF 是 20 年。把这个数据从小时换成年。用 MTBF 来解释这个 20 年的寿命，可以得到什么？
c.如果一个厂商宣称某种型号的设备有 100 万小时的 MTBF。这对设备预期的寿命有什么影响？
答：a.750000 的平均无故障时间除以 1000，得到故障间隔为 750，所以是每月一次。
b.根据小时来计算，人的平均无故障时间是 8760（一年中的小时数）除以 0.001，得到

8760000 的 MTBF 值。8760000 小时约等于 1000 年。所以，对于一个与、预期寿命是 20 年的人来说，这并不能说明说明。

c.MTBF 与设备的寿命无关。硬盘的一般设计寿命是 5 年。即使一个硬盘真的有 100 万年的 MTBF，设备本身的寿命也达不到那么长时间。

12.16 讨论 sparing 扇区和 slipping 扇区的优点和缺点。

答：sparing 扇区会增加额外的换道时间和旋转延迟，可能使响应时间增加 8ms。sparing 扇区对将来的读盘有较小的影响，但在重映射的时候，需要读写所有道上的数据来跳过坏块。

12.17 描述为什么操作系统要知道块存储到磁盘的详细信息。操作系统这样通过这些来提高文件系统的性能？

答：当为文件分配物理块的时候，如果几何相邻的块有更多关于块物理方位的信息，就把这些块分配给文件。并且，可以在同一柱面的不同光盘面连续分配两个块，这使下一次的访问时间减到了最小。

12.18 操作系统常把移动磁盘当作共享文件系统，而一个磁盘上一次只能有一个应用。说出磁盘和磁带处理方式不同的 3 点原因。操作系统通过共享文件系统访问磁带，还需要什么特殊的支持。应用共享磁带，需要什么特殊的属性，能否把文件当作磁盘上的文件来使用？

答：a.磁盘有更快的随机访问时间，所以对交叉存储的文件有更好的性能。而磁带需要更多的定位时间。所以，当两个用户访问一个磁带时，驱动器的大部分时间都用作转换磁带和定位，只有少量的时间用于数据传输。这种情况类似于虚拟内存没有足够的物理内存而发生 thrashing。b.磁带的带盘是可以移动的。有时，可能需要把当前带盘的数据存放在拷贝中（远离电脑的地方），来防止电脑所在处发生火灾。c.磁带常常用于在生产者和消费者之间传送大量的数据，这些磁带不能作为不同的共享存储设备。

为了支持共享文件系统方式访问磁带，操作系统需要提供一般文件系统的功能，包括：管理所有磁带上的文件系统命名空间；空间回收；I/O 调度。访问磁带文件系统的应用需要能承受长时间的延迟。为了提高性能，这些应用要大量减少 I/O 操作来换取磁带调度算法较高的效率。

12.19 如果磁带设备每英尺存储的比特数与磁盘相同，对性能和价格会产生什么影响？

答：为了达到与磁盘相同的单位存储量，磁带的单位存储量会以 2 的指数级增长。这会使磁带比磁盘便宜。磁带的容量可能会大于 1GB，所以一个磁带就可以代替现在的一个磁带机，因而减少了花费。单位存储量不会对数据传输造成压力，但是大容量会减慢磁盘的转换。

12.20 通过简单的计算，比较由磁盘和引入第三方存储设备的 1 兆节的操作系统在花费和性能上的不同。假设磁盘的容量是 10GB，花费 1000 元，每秒传输 5MB 数据，平均访问延迟是 15 毫秒。假设磁带库每兆节花费 10 元，每秒传输 10MB,平均等待延迟是 20 秒。计算纯磁盘系统的总花费，最高数据传输率和平均等待延迟。现在假设有 5%的数据是经常读写的，把它们存放在磁盘中，其余 95%存放在磁带库中。所以 95%的请求由磁盘响应，5%的请求由磁带库响应。此时的总花费，最高数据传输率和平均等待延迟是多少？

答：首先计算纯磁盘系统。1TB=1024GB，大概地计算出需要 100 个磁盘，花费是 100000 元，加上 20%的电缆，电源，其他开销，总花费再 120000 元左右。总的数据访问速度是 500 MB/s，平均等待时间与工作量有关。如果要求传输的大小是 8KB，请求的数据随机分布在磁盘上。如果系统是轻负荷，请求会到达一个空闲的磁盘，所以响应时间是 15ms 的访问时间加上 2ms 的传输时间。如果系统是重负荷，延迟会随着队列长度的增加而增加。

再考虑分层存储系统。总共需要的磁盘空间是 50GB，所以需要 5 个磁盘，花费是 5000 元（增加 20%的其他费用就是 6000 元）。950GB 的磁带库花费是 9500 元，总开销 15500 元。最大是数据传输率以来与磁带库中驱动器的数量。假设只有一个驱动器，总的速率就是 60 MB/s。对于轻负荷系统来说，95%的请求由磁盘响应，延迟是 17ms，其余 5%的请求由磁带库响应，延迟 20 秒。所以平均延迟是 (95×0.017+5×20)/100，约等于 1 秒。即使磁带库的请求队列是空的，磁带库的延迟也是造成系统延迟的主要原因，因为有 1/20 的工作都在延时为 20 秒的设备上完成。如果系统重负荷，延迟会随磁带库等待队列的增长而增长。

层次存储系统更便宜。由于 95%的请求由磁盘响应，所以性能与纯磁盘系统差不多。但是层次存储系统的最大数据传输率和平均等待时间不如纯磁盘系统。

12.21 假设现在发明了一种全息照相存储器，它花费 10000 元，平均访问速度是 40 毫秒。如果它用 100 美元的 CD 大小的胶卷，胶卷可以保存 40000 张图片，每张图片都是黑白正方形的，分辨率是 6000 × 6000 像素（每像素 1bit）。假设驱动器 1 毫秒可以读写 1 张图片。

a.这个设备有什么作用？

b.这个设备会对操作系统的 I/O 操作产生什么影响？

c.其他存储设备会不会因为这种设备的发明而被淘汰？

答：先计算这种设备的性能。传输速率是 4291 MB/s，远快于现在使用的硬盘（最快的硬盘也只能达到 40 MB/s）。以下的回答说明了这个设备不能储存小于 4MB 的块。

a.这一设备在存储图片，视频文件和数字媒体文件时会有大量需求。

b.假设与这种设备通信的时间与它的吞吐量匹配，大量的数字文件的读写就会优化。但是管理数字对象的时间不会改变。所以性能上会有很大的提高。

c.现在，图片大小的对象都存放在光存储设备上，如磁带，磁盘。如果层次存储系统可行的话，将会大量需要这种设备。在层次系统中，任何一种媒体设备是有用的，所以没有一种会被替代。磁带仍是用于小文件的随机访问，磁带用于定点存档和备份。光盘用于方便计算机之间的交流和大量的低价存储。

由于全息照相存储器的大小和省电，它可能会代替数码相机 MP3 和掌上电脑的存储芯片。

12.22 设单面 5.25 英尺的光盘单位存储量是每英尺 1GB。假设某种磁带的单位存储量是每英尺 20GB，0.5 英尺宽，1800 英尺长。如有一种光磁带有磁带的容量和磁盘的存储密度，这种光磁带可以储存多少数据？如果磁带的价格是 25 元，这种设备多少钱比较合理？

答：5.25 英尺光盘的面积大约是 19.625 平方英尺。假设核心 hub 的直径是 1.5 英尺，hub 占用的面积是 1.77 平方英尺，留下 17.86 平方英尺用作存放数据。所以光盘的存储量大约是 2.2 GB。

磁带的表面积是 10800 平方英尺，所以存储能力是 26GB。

如果 10800 平方英尺的存储密度是每英尺 1GB，总的存储量能达到 1350GB，即 1.3TB。如果单位存储量的价格与磁带相同，它的价格将是磁带的 50 倍，即 1250 元。

12.23 基于磁带的操作系统如何获得空闲列表？假只使用磁带存储技术，用磁带结束符来定位，隔离和读取位置指令。

答：由于只使用磁带存储技术，所以所有的看空闲空间都在磁带的后面部分。不需要存储这些空间的位置，因为可以用结束符来定位。结束符后的空闲空间大小可以用一个数值来记录。同时需要另一个数值来记录分配给文件但是已经在逻辑上被删除的空间的大小（由于只能用磁带存储技术，这些空间并没有被重新声明）。所以，当有需要时，可以把没有删除的文件一道另一张磁带上来时这些空间可以被重复使用。可以在磁盘上存储空闲和删除的空间大小

来方便访问。这些数据还会作为最后一个数据块存放在磁带上。当新分配存储区是就要重写这个数据块。