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供这些服务。

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

（

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/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 。

5.3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

P1 10 3

P2 1 1

P3 2 3

P4 1 4

P5 5 2

假设在时刻0以进程P1，P2，P3，P4，P5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

d src="/pic/view?ih=rn=100doc\_id=07b3daa3b0717fd5360cdc55o=jpg\_6\_0\_\_\_\_\_\_\_pn=1iw=ix=sign=52d84f5fbae08171ab032dca33c36693type=1iy=aimw=app\_ver=2.9.8.2ua=bd\_800\_800\_IncredibleS\_2.9.8.2\_2.3.7bid=1app\_ua=IncredibleSuid=cuid=fr=3Bdi\_bear=WIFIfrom=3\_10000bduss=pid=1screen=800\_800sys\_ver=2.3.7" alt="操作系统概念第七版习题答案(中文版)" />

d src="/pic/view?ih=rn=100doc\_id=07b3daa3b0717fd5360cdc55o=jpg\_6\_0\_\_\_\_\_\_\_pn=1iw=ix=sign=52d84f5fbae08171ab032dca33c36693type=1iy=aimw=app\_ver=2.9.8.2ua=bd\_800\_800\_IncredibleS\_2.9.8.2\_2.3.7bid=1app\_ua=IncredibleSuid=cuid=fr=3Bdi\_bear=WIFIfrom=3\_10000bduss=pid=1screen=800\_800sys\_ver=2.3.7" alt="操作系统概念第七版习题答案(中文版)" />

c.等待时间

d src="/pic/view?ih=rn=100doc\_id=07b3daa3b0717fd5360cdc55o=jpg\_6\_0\_\_\_\_\_\_\_pn=1iw=ix=sign=52d84f5fbae08171ab032dca33c36693type=1iy=aimw=app\_ver=2.9.8.2ua=bd\_800\_800\_IncredibleS\_2.9.8.2\_2.3.7bid=1app\_ua=IncredibleSuid=cuid=fr=3Bdi\_bear=WIFIfrom=3\_10000bduss=pid=1screen=800\_800sys\_ver=2.3.7" alt="操作系统概念第七版习题答案(中文版)" />

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

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Pi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。

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。

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. \_n

i =1 Maxi < m + n

b. Maxi ≥ 1 for all i

Proof: Needi = Maxi − Alloca tioni

If there exists a deadlock state then:

c. \_n

i =1 Alloca tioni = m

Use a. to get:\_ Needi + \_ Alloca tioni = \_ Maxi < m + n

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

Rewrite to get:\_n

i =1 Needi < 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

本文档下载自文档之家，www.doczj.com-免费文档分享平台，众多试卷、习题答案、公务员考试、英语学习、法语学习、人力资源管理、电脑基础知识、学习计划、工作计划、工作总结、活动策划、企业管理等文档分类免费下载；乐于分享，共同进步，转载请保留出处:http://www.doczj.com/doc/07b3daa3b0717fd5360cdc55.html