第3章

死锁

死锁是多道程序并发执行带来的另一个严重问题，它是操作系统乃至并发程序设计中最难处理的问题之一。进程死锁产生的根本原因有两个：一是竞争资源；二是进程间推进的顺序不合理。死锁处理不好将会导致整个系统运行效率下降，甚至不能正常运行。操作系统设计者必须高度重视死锁现象。死锁普遍存在，对于死锁，不存在完美的、彻底的解决方案，只能在众多可行方案中选择一个折中方案。

本章主要讲解死锁的基本概念、死锁的处理策略、死锁的预防、死锁的避免以及死锁的检测和解除。本章需要重点掌握：

- 了解死锁的基本概念；了解死锁的检测、死锁解除的方法；
- 理解死锁产生的原因；理解死锁预防与死锁避免的区别；
- 掌握死锁产生的4个必要条件；掌握系统安全状态及其判别方法；掌握处理死锁的方法；
- 学会银行家算法及其应用。

3.1 死锁的定义和产生原因


视频讲解

3.1.1 死锁的定义

死锁是指一组并发执行的进程彼此等待对方释放资源，而在没有得到对方占有的资源之前不释放自己所占有的资源，导致彼此都不能向前推进，称该组进程发生了死锁。

死锁产生后，若无外力干预，陷入死锁的各个进程都永远不能向前推进，导致这些进程不能正常结束。同时，要求共享使用死锁进程所占资源的其他进程，或者需要与死锁进行某种合作的其他进程也会受到牵连，也不能正常结束。最终可能导致系统瘫痪，给系统和用户带来极大损失。因此，操作系统设计者必须对死锁现象予以充分重视。

死锁问题不仅普遍存在于计算机系统中，在日常生活中也广泛存在。现实生活中交通死锁问题较为常见。例如，某交通路口恰有4辆汽车几乎同时到达，并相互交叉停了下来，如图3-1(a)所示。如果该路口没有采取任何交通管理措施，4辆车同时驶过十字路口，就会发生如图3-1(b)所示的场面。最终结果是4辆车都在等待对方车辆后退，但谁也不先让，所以都不能通过该路口，出现交通死锁现象。

在该例中，可把每辆汽车行驶过十字路口看作一个进程，系统中共有4个这样的进程并发执行。十字路口可看作4个临界资源，如图3-1(a)中标注的a、b、c和d所示。每个汽车进程在过路口都要依次申请其中的两个资源。1号汽车申请a和b资源；2号汽车申请b和c资源；3号汽车申请c和d资源；4号汽车申请d和a资源。出现死锁情况时，1号汽车占据a资源和申请b资源；2号汽车占据b资源和申请c资源；3号汽车占据c资源和申请

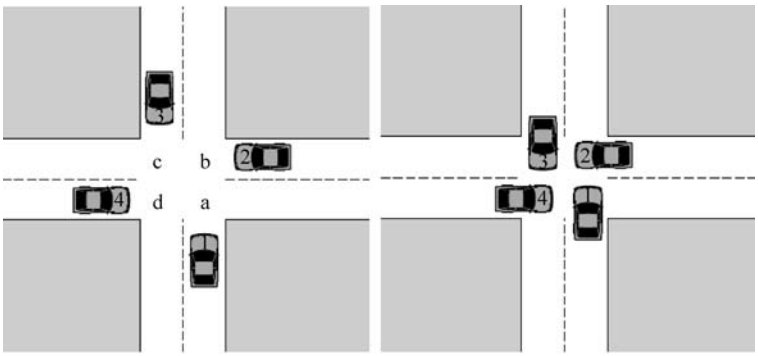

图 3-1 交通阻塞导致死锁示意图

d 资源；4 号汽车占据 d 资源和申请 a 资源。此时，若不采取外力措施干预，如交通警察未赶到现场进行疏导管理，4 辆汽车将永远互相等待。在死锁解除前，如果此后还有其他汽车想通过该十字路口，由于 4 个路口资源都已经分配，故也不能通过，因此造成更多进程阻塞。

在计算机系统中，凡是涉及临界资源(即互斥资源)申请的并发进程间都可能发生死锁。举一个计算机系统中的简单实例，如某系统中有 P_1 和 P_2 两个进程并发执行，P_1 和 P_2 两个进程在执行过程中都需要使用一台打印机和一台 CD-ROM 驱动器。该系统中只有一台打印机和一台 CD-ROM 驱动器，两者均为临界资源。假设进程 P_1 和 P_2 的执行过程如图 3-2 所示。

进程P_1	进程P_2
⋮	⋮
申请打印机	申请CD-ROM驱动器
⋮	⋮
申请CD-ROM驱动器	申请打印机
⋮	⋮
释放打印机	释放CD-ROM驱动器
⋮	⋮
释放CD-ROM驱动器	释放打印机
⋮	⋮

图 3-2 两个并发进程的执行情况图

当进程 P_1 申请打印机成功时，恰巧此时进程发生切换，调度程序选中 P_2 执行，P_1 暂时变为就绪态等待调度程序调度。进程 P_2 首先申请 CD-ROM 驱动器，此时该设备处于空闲状态，故系统把它分配给进程 P_2。这时，P_1 和 P_2 两个进程都不能向前继续推进。进程 P_1 申请 CD-ROM 驱动器，但 CD-ROM 驱动器已被 P_2 占用，只能被阻塞，等待进程 P_2 使用完毕后，释放 CD-ROM 驱动器给它；进程 P_2 向前推进时，申请打印机，但打印机此时被进程 P_1 占用，P_2 也只好阻塞，等待进程 P_1 使用完毕后释放。因此 P_1 和 P_2 两个进程都陷入了相互等待的状态，形成了死锁。

通过上面介绍的例子可以发现，死锁具有以下特点。

(1) 陷入死锁的进程是系统并发进程中的一部分，且至少要有两个进程，单个进程不会

形成死锁。

(2) 陷入死锁的进程彼此都在等待对方释放资源,形成一个循环等待链。

(3) 死锁形成后,在没有外力干预时,陷入死锁的进程不能自己解除死锁,死锁进程无法正常结束。

(4) 如不及时解除死锁,死锁进程占有的资源不能被其他进程所使用,导致系统中更多进程阻塞,造成资源利用率下降。

3.1.2 死锁产生的原因

产生死锁的原因可归结为两点。

(1) 竞争资源。当系统中供多个进程所共享的资源不足以同时满足它们的需要时,引起它们对资源的竞争而产生死锁。

(2) 进程推进顺序不当。进程在运行过程中,请求和释放资源的顺序不当,导致了进程死锁。

1. 竞争资源

死锁产生的根本原因是资源竞争且分配不当。因为多道程序并发执行,造成多个进程在执行过程中所需要的资源数远远大于系统能提供的资源数。如图 3-2 所示例子中的进程 P_1 和 P_2 两个进程之所以产生死锁,就是因为系统中的打印机和 CD-ROM 驱动器不够用,如果为了防止死锁而配置多台打印机和 CD-ROM 驱动器,从成本角度看是不现实的。由于各个进程对资源的需求量是动态变化的,虽然有多个并发执行的进程,但某个时刻它们对打印机提出的请求可能只有一个,甚至没有,系统配置多台打印机就造成了资源的浪费。

计算机系统中有很多资源,按照占用方式来分,可分为可剥夺资源与不可剥夺资源。

(1) 可剥夺资源。某进程在获得这类资源后,即使该进程没有使用完,该类资源也可以被其他进程剥夺使用。例如,优先级高的进程可以抢占优先级低的进程的处理器;如可把一个进程从一个存储区转移到另一个存储区,在主存紧张时,还可将一个进程从主存调出到辅存上,即剥夺该进程在主存的空间。可见,CPU、主存和磁盘均属于可剥夺资源,竞争可剥夺资源不可能出现死锁。

(2) 不可剥夺资源。当系统把这类资源分配给某进程后,不能强行收回,只能在进程使用完后自行释放,然后其他进程才能使用。例如,当一个进程已开始刻录光盘时,如果突然将刻录机分配给另一个进程,其结果必然会损坏正在刻录的光盘,因此只能等刻录好光盘后由进程自身释放刻录机。另外,打印机、刻录机和 CD-ROM 驱动器等都属于不可剥夺资源。

2. 进程推进的顺序不当

并发执行的进程在运行中存在异步性,彼此之间相对执行速度不定,存在着多种推进顺序。并发进程间推进顺序不当时会引起死锁。

为了更好地描述进程 P_1 和进程 P_2 的推进顺序,图 3-3 给出了两者的推进顺序示意图。横轴表示进程 P_1 的执行进展,纵轴表示进程 P_2 的执行进展,R_1 和 R_2 表示资源。从原点出发的不同路径分别表示两个进程以不同的速度向前推进。图中给出了 4 种不同的执行路径,表示 4 种不同的进程间推进顺序。

(1) 进程 P_1 申请并获得资源 R_1 和资源 R_2,执行结束后释放资源 R_1 和资源 R_2。然后进程 P_2 被调度执行,它也申请并获得资源 R_2 和资源 R_1,执行结束后也释放资源 R_2 和资源

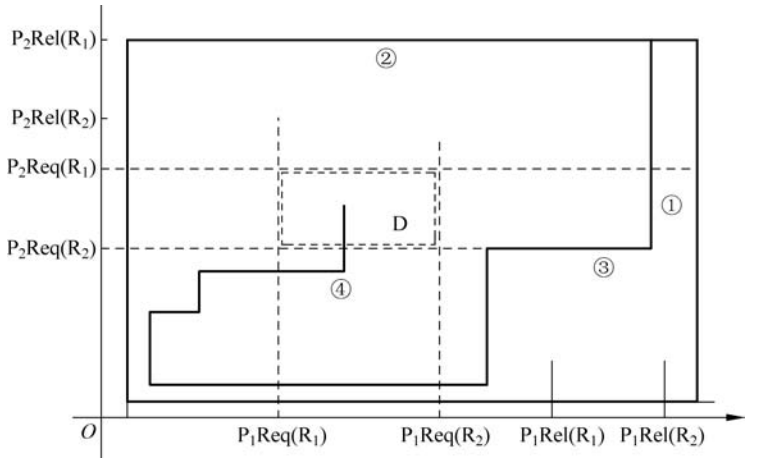

图 3-3 进程推进顺序示意图

R_1。进程 P_1 和进程 P_2 均顺利执行完毕。

(2) 进程 P_2 先调度执行,申请并获得资源 R_2 和资源 R_1,执行结束后释放资源 R_2 和资源 R_1。然后进程 P_1 申请并获得资源 R_1 和资源 R_2,执行结束后也释放资源 R_1 和资源 R_2。进程 P_1 和进程 P_2 均顺利执行完毕。

(3) 进程 P_1 申请并获得资源 R_1 和资源 R_2,然后进程 P_2 申请资源 R_2,由于资源 R_2 已经分配给进程 P_1,进程 P_2 只能阻塞。进程 P_1 执行结束后释放资源 R_1 和资源 R_2,此时 P_2 进程获得了资源 R_2,由阻塞状态变为就绪状态,然后 P_2 进程又申请资源 R_1 并获得资源 R_1,因此 P_2 进程被调度执行,执行完毕后释放资源 R_2 和资源 R_1,执行完毕。

(4) 进程 P_1 申请并获得资源 R_1,进程 P_2 申请并获得资源 R_2,此时,进程 P_1 和进程 P_2 进入到 D 区,无论进程 P_1 还是进程 P_2 被调度选中执行,都会出现阻塞。进程 P_1 因申请资源 R_2 不能获得而阻塞,进程 P_2 因申请资源 R_1 不能获得也阻塞。两个进程相互等待对方释放资源,陷入死锁。


视频讲解

3.2 产生死锁的必要条件

要解决死锁问题,应分析在什么情况下可能发生死锁。Coffman 首先提出死锁产生的四个必要条件——称为 Coffman 条件。由于这四个条件是必要条件,一旦死锁出现,这四个必要条件都必然成立。因此,只要有一个必要条件被系统或用户破坏,就不会出现死锁现象。系统产生死锁的四个必要条件如下。

(1) 互斥条件:进程应互斥使用资源,任一时刻一个资源仅为一个进程独占,若另一个进程请求一个已被占用的资源时,它被置成等待状态,直到占用者释放资源。

(2) 请求和保持条件:一个进程请求资源得不到满足而等待时,不释放已占有的资源。

(3) 不剥夺条件:任一进程不能从另一进程那里抢夺资源,即已被占用的资源,只能由占用进程自身来释放。

(4) 循环等待条件:存在一个循环等待链,其中,每一个进程分别等待另一个进程所占有的资源,造成永久等待。

这四个条件仅是必要条件而不是充分条件，即只要发生死锁则这四个条件一定会同时成立，但反之则不然。循环等待条件隐含着前三个条件，即只有前三个条件成立，第四个条件才会成立。特别注意的是：环路等待条件只是死锁产生的必要条件，而不是等价定义。死锁一旦产生，则死锁进程间必存在循环等待链，但存在循环等待链不一定产生死锁。例如：某系统中有两个 R_1 资源和一个 R_2 资源。假设系统中有三个进程 P_1、P_2 和 P_3 并发执行，存在一个环路，进程 P_1 等待 P_2 所占有资源 R_1，进程 P_2 等待 P_1 所占有资源 R_2，此时进程 P_3 占有另一个 R_1 资源，显然进程 P_1 和进程 P_2 已陷入死锁。但是，如果进程 P_3 以后的执行过程中没有提出新的关于 R_1 或 R_2 的请求，且顺利执行完毕，释放其占有的资源 R_1 给进程 P_1，则进程 P_1 和 P_2 形成的循环等待链将被打破，死锁解除。

3.3 死锁的处理方法

死锁普遍存在于并发执行进程间，处理死锁的方法很多种。其中最简单的处理方法就是忽略死锁。就像鸵鸟遇到无法避免的危险时就把头埋在沙子里那样，对出现的危险不管不顾。操作系统处理死锁的一种策略就是不预防、不避免，对可能出现的死锁采取放任的态度，称作鸵鸟算法。

如果从计算机系统的核心目的考虑，鸵鸟算法这样做也是合理的，计算机的核心目的是提高系统的吞吐量，降低开销。因此，下列两个因素决定了操作系统可以采用鸵鸟算法。首先死锁的检测、恢复和预防的算法编写、测试和调试都很复杂。其次，它们在很大程度上了降低了系统的运行速度，开销比较大。因此，对于很少发生死锁的计算机系统而言，采用鸵鸟算法也是合理和有效的。即使进程发生了死锁，也可以重新启动该进程，以重新启动该进程的较小开销避免了死锁处理的大量开销。

鸵鸟算法的意义在于，当出现死锁的概率很小，并且出现之后处理死锁会花费很大的代价时，执行死锁避免的开销很大，还不如不做处理。因此，鸵鸟算法是平衡性能和复杂性的一种方法，是目前通用操作系统中采用最多的方法。

如果涉及死锁处理的是高可靠性系统或实时控制系统，则不适宜采用鸵鸟算法。因此，要采取各种措施预防和避免死锁，一旦出现死锁，系统要能解除死锁。

按照死锁处理的时机划分，可把死锁处理的方法分成四种。

(1) 预防死锁。预防死锁是指在系统运行之前就采取相应措施，消除发生死锁的任何可能性。通过消除死锁发生的必要条件可预防死锁。破坏产生死锁的四个必要条件中的一个或几个来预防产生死锁。预防死锁是处理死锁的静态策略，它虽比较保守、资源利用率低，但因简单明了、较易实现，现仍被广泛使用。

(2) 避免死锁。避免死锁是为了克服预防死锁的不足而提出的动态策略。避免死锁与预防死锁的策略不同，它并不是事先采取各种限制措施，去破坏产生死锁的四个必要条件，而是在资源动态分配过程中，用某种方法防止系统进入不安全状态，从而可以避免发生死锁。避免死锁的方法虽好，但也存在两个缺点：一是对每个进程申请资源分析计算较为复杂且系统开销较大；二是在进程执行前，很难精确掌握每个进程所需的最大资源数。

(3) 检测死锁。这种方法无须事先采取任何限制性措施，允许进程在运行过程中发生死锁。但可通过检测机构及时地检测出死锁的发生，然后采取适当的措施，把进程从死锁中

解脱出来。死锁检测不延长进程初始化时间,允许对死锁进行现场处理,其缺点是通过剥夺解除死锁,给系统或用户造成一定的损失。

(4) 解除死锁。当检测到系统中已发生死锁时,就采取相应措施,将进程从死锁状态中解脱出来。常用的方法是撤销一些进程,回收它们的资源,将它们分配给已处于阻塞状态的进程,使其能继续运行。在实际执行中,由于并发进程推进顺序的多样性,系统很难做到有效地解除死锁。

上述四种方法对于死锁的防范程度逐渐减弱,但相对应的是资源利用率的提高,以及进程因资源因素而阻塞的频度下降(即并发程度提高)。处理死锁的基本方法的比较如表 3-1 所示。

表 3-1 处理死锁的基本方法比较

方 法	资源分配策略	各种可能模式	主要优点	主要缺点
死锁的预防	保守的;宁可资源闲置	一次性请求所有资源 资源被剥夺 资源按序申请	适用于作突发式处理的进程,不用被剥夺 适用于状态可以保存和恢复的资源 可以在编译时(而不必在运行时)就行检查	效率低;进程初始化时间延长 剥夺次数过多;多次对资源重新启动 不便灵活申请新资源,申请序号很难确定
死锁的避免	在运行时动态地分配资源,判断系统是否是安全状态	寻找安全序列	不会进行剥夺	必须知道将来的资源需求;进程可能会长时间阻塞
死锁的检测和恢复	宽松的;只要允许,就分配资源	定期检查系统是否发生死锁	不延迟进程初始化时间;允许对死锁进程进行现场处理	通过剥夺解除死锁,造成损失

视频讲解

3.4 死锁的预防

预防死锁的方法是通过破坏产生死锁的四个必要条件中的一个或几个,以避免发生死锁。由于互斥条件是非共享设备所必须的,不仅不能改变,还应加以保证,因此主要是破坏产生死锁的后三个条件。

3.4.1 破坏“请求”条件和“保持”条件

破坏这个条件的办法很简单,可采用预分配资源方法。所有进程在运行前,系统一次性地分配其运行所需要的全部资源。进程在运行期间,不会再提出资源要求,从而破坏了“请求”条件。系统在分配资源时,只要有一种资源不能满足进程的要求,即使其他所需的各种资源都空闲也不能分配给该进程,而让该进程等待。由于该进程在等待期间未占有任何资源,于是破坏了“保持”条件,从而可以预防死锁的发生。

这种方法虽然简单、易行且安全,但是系统的资源浪费严重,进程经常会发生饥饿现象。

例如，某进程在开始时从 CD-ROM 驱动器中读入初始数据，然后进行长达数小时的计算，最后几分钟通过打印机把运算结果输出。

3.4.2 破坏“不剥夺”条件

破坏“不剥夺”条件就是采用可剥夺的资源分配方式，即允许对系统资源进行抢占。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得到满足时，它必须释放已经保持的所有资源，待以后需要时再重新申请。这就意味着进程已占有的资源会被暂时地释放，或者被抢占了，从而破坏了“不剥夺”条件。

这种方法实现起来比较复杂，并且付出了很大的代价，适用于资源状态易于保留和恢复的环境中，如 CPU 寄存器和主存空间，但一般不能用于打印机和磁带机这类资源。

3.4.3 破坏“循环等待”条件

一个能保证“循环等待”条件不成立的方法是，对系统所有资源类型进行线性排序，并赋予不同的序号，然后按序号进行分配，规定进程不能连续两次申请同类资源，这样一来，进程在申请、占用资源时就不会形成资源申请环路，也就不会产生循环等待。

假设 $R=\{r_1, r_2, \cdots, r_m\}$，表示一组资源类型，定义一组函数 $F: R \rightarrow N$，式中 N 是一组自然数。例如，一组资源包括磁带机、磁盘机和打印机。函数 F 可定义如下：

F(磁带机)=1，F(磁盘机)=5，F(打印机)=12

为了预防死锁，进行如下约定：所有进程对资源的申请严格按照序号递增的次序进行，即一个进程最初可以申请任何类型的资源，例如，r_i 以后该进程可以申请一个新资源 r_j（当且仅当 $F(r_j)>F(r_i)$）。例如，按上述规定，一个希望同时使用磁带机和打印机的进程，必须首先申请磁带机，然后再申请打印机。

另一种申请办法也很简单：先弃大，再取小。也就是说，无论何时，一个进程申请资源 r_j，它应释放所有满足 $F(r_i)>F(r_j)$ 关系的资源 r_i。

这两种办法都是可行的，不会产生循环等待条件。这种策略使资源利用率和系统吞吐量都有很大的提高，但是限制了进程对资源的请求，同时给系统中所有资源合理编号也较为困难，并且会增加系统开销。为了遵循按序号申请的次序，暂不使用的资源也需要提前申请，从而增加了进程对资源的占用时间。

3.5 死锁的避免

死锁的预防是静态策略，对进程申请资源的活动进行严格限制，以保证死锁不会发生。死锁的避免和死锁的预防不同，系统允许进程动态地申请资源，系统在进行资源分配之前，对进程发出的资源申请进行严格检查，如满足该申请后，系统仍处于安全状态，则分配资源给该进程，否则拒绝此申请。

3.5.1 系统安全状态


视频讲解

所谓安全状态是指系统能够按照某种进程执行序列，如$<P_1, P_2, P_3, \cdots, P_n>$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,使得每个进程都可顺利地完成。此时,称系统处于系统安全状态,进程执行序列<P_1,P_2,P_3,…,P_n>为当前系统的一个安全序列。如果系统无法找到这样一个安全序列,则称当前系统处于不安全状态。

【例1】 现有12个同类资源供3个进程共享,进程P_1总共需要9个资源,但第一次先申请2个资源,进程P_2总共需要10个资源,第一次要求分配5个资源,进程P_3总共需要4个资源,第一次请求2个资源。经第一轮的分配后,系统中还有3个资源未被分配,现在的分配情况如表3-2所示。

表3-2 资源分配状态

进　程	已占资源数	最大需求数
P_1	2	9
P_2	5	10
P_3	2	4

这时,系统处于安全状态。因为还剩余3个资源,可把其中的2个资源再分配给进程P_3,系统还剩余1个资源。进程P_3已经得到了所需的全部资源,能执行到结束,且归还所占的4个资源。现在系统共有5个空闲的资源,可分配给进程P_2。同样地,进程P_2得到了所需的全部资源,执行结束后可归还10个资源。最后进程P_1也能得到尚需的7个资源而执行到结束,然后归还9个资源。这样,三个进程都能在有限的时间内得到各自所需的全部资源,执行结束后,系统可收回所有资源。

但是,在第一轮的分配后,若进程P_1先提出再分配一个资源的要求,系统从剩余的资源中分配1个给进程P_1后,尚剩余2个资源,现各进程占用资源情况如表3-3所示。

表3-3 资源分配状态

进　程	已占资源数	最大需求数
P_1	3	9
P_2	5	10
P_3	2	4

虽然剩余的2个资源可满足进程P_3的需求,但当进程P_3得到全部资源且执行结束后,系统最多只有4个由进程P_3归还的资源,而进程P_1和进程P_2还分别需要6个资源和5个资源。显然,系统中的资源已不能满足这两个进程的需求了,也就是说,这两个进程已经不能在有限的时间里得到需要的全部资源。系统已从安全状态转为不安全状态,这是由于资源分配不得当造成的。

应注意的是,“不安全状态”与“死锁”两者并不是等同的,上述的分配情况使系统进入了不安全状态,但死锁尚未发生。如果进程P_1提出再申请5个资源,则系统不能满足它的要求,从而让进程P_1成为等待资源状态。类似地,进程P_2请求分配尚需的6个资源时,也成为了等待资源状态。此时,进程P_1和P_2的等待永远结束不了,它们就处于“死锁”状态了。可见,不安全状态隐含着将发生死锁。

只要能保持系统处于安全状态就可避免死锁的发生,故每当有进程提出分配资源的请

求时，系统应分析各进程已占资源数、尚需资源数和系统中可以分配的剩余资源数，然后决定是否为当前的申请分配资源。如果能维持系统的安全状态，则可为进程分配资源，否则暂不为申请者分配资源，直到有其他进程归还资源后，再分配给它。


视频讲解

3.5.2 银行家算法

1. 银行家算法的基本思想

最有代表性的避免死锁的算法就是 Dijkstra 的银行家算法。其基本思想是：在资源分配前，判断系统是否处于安全状态，如处于安全状态则把资源分配给申请进程，如处于不安全状态则令申请资源的进程阻塞，不响应其资源申请。

这和现实社会中的银行家很相似，可以用现实生活中的银行贷款实例来类比银行家算法的执行过程。例如，银行家有一笔资金 m 万元，n 个客户需要贷款，他们都和银行签订了贷款协议，每个客户所需的资金不同，且都不超过 m 万元，但客户们的贷款总和远远超过 m 万元。协议中规定，银行根据自身情况向各个客户发放贷款。客户只有在获得全部贷款后，才能在一定的时间内将全部资金归还给银行家。银行家并不一定批准客户每次的贷款请求，在每次发放贷款时，银行家都要考虑发放该笔贷款是否会使得银行无法正常运转。只有在批准贷款请求不会导致银行银根不足时，该贷款请求才被批准。

在此实例中，银行家采用的策略就是死锁的动态避免策略。银行家类似于操作系统中的资源分配程序，m 万元类似于系统中可供分配的空闲资源，每个贷款客户类似于并发执行的进程，贷款金额就是该进程所需的最大资源数。每个进程在执行过程中动态地向系统提出资源请求，只有全部资源请求满足后，才能执行完毕，归还其所占有的全部系统资源。

综上所述，银行家算法的核心理念就是把资源分配给那些最容易执行完成的进程，保证系统中各进程最终都能正常完成。

2. 银行家算法的数据结构

为了实现银行家算法，在系统中必须设置 4 个数据结构，分别用来描述系统中可利用的资源、所有进程对资源的最大需求、系统中的资源分配，以及所有进程还需要多少资源的情况。

(1) 可利用资源向量 Available。这是一个含有 m 个元素的数组，其中每一个元素代表一类可利用的资源数目，其初始值是系统中所配置的该类全部可用资源的数目，其数值随该类资源的分配和回收而动态地改变。如果 Available$[j]=K$，则表示系统中现有 R_j 类资源共 K 个。

(2) 最大需求矩阵 Max。这是一个 $n\times m$ 的矩阵，它定义了系统中的 n 个进程中的每一个进程对 m 类资源的最大需求。如果 Max$[i,j]=K$，则表示进程 i 需要 R_j 类资源的最大数目为 K。

(3) 分配矩阵 Allocation。这是一个 $n\times m$ 的矩阵，它定义了系统中每一类资源当前已分配给每一进程的资源数。如果 Allocation$[i,j]=K$，则表示进程 i 当前已分得 R_j 类资源的数目为 K。

(4) 需求矩阵 Need。这是一个 $n\times m$ 的矩阵，它用以表示每一个进程尚需的各类资源数。如果 Need$[i,j]=K$，则表示进程 i 还需要 K 个 R_j 类资源才能完成其任务。

上述三个矩阵间的关系是：

```
Need[i,j] = Max[i,j] - Allocation[i,j]
```

3. 银行家算法

当进程 P_i 申请资源时，向系统提交一个资源申请向量 $Request_i[j]$，如果 $Request_i[j]=k$，表示进程 P_i 申请 k 个 j 类资源。

银行家算法按照下述流程进行检查，判断是否把 k 个 j 类资源分配给进程。

(1) 如果 $Request_i[j]+Allocation[i,j]<=Max[i,j]$成立，转向步骤(2)；如不成立，则说明进程的 j 类资源申请超过了其最大需求量，报错中断返回。

(2) 如果 $Request_i[j]<=Available[j]$成立，转向步骤(3)；如不成立，则说明系统中现有的 j 类资源不能满足进程 P_i 的资源申请。该请求不能满足，进程 P_i 被阻塞，结束算法返回。

(3) 系统试着把资源分配给进程，并修改下面数据结构中的值：

```
Available[j] = Available[j] - Requesti[j]
Allocation[i,j] = Allocation[i,j] + Requesti[j]
Need[i,j] = Need[i,j] - Requesti[j]
```

(4) 调用系统安全性算法，检查此次资源分配后系统是否处于安全状态。若安全，满足进程 P_i 的资源申请；否则，将本次试探分配作废，恢复原来资源分配状态，不响应进程 P_i 的资源申请，让进程 P_i 等待。

4. 安全性算法

系统所执行的安全性算法可描述如下。

(1) 设置两个向量：

① 工作向量 Work，它表示系统可提供给进程继续运行所需的各类资源数目，它含有 m 个元素，在执行安全算法开始时，Work=Available；

② Finish，它表示系统是否有足够的资源分配给进程，使之运行完成。开始时先做 Finish[i]=false；当有足够资源分配给进程时，再令 Finish[i]=true。

(2) 从进程集合中找到一个能满足下述条件的进程：

① Finish[i]=false；

② Need[i,j]<=Work[j]；

若找到，则执行步骤(3)；否则，执行步骤(4)。

(3) 当进程 P_i 获得资源后，可顺利执行，直至完成，并释放出分配给它的资源，故应执行：

Work[j]= Work[j]+ Allocation[i,j]；

Finish[i]=true；

Go to step 2；

(4) 如果所有进程的 Finish[i]=true 都满足，则表示系统处于安全状态；否则，系统则处于不安全状态。

5. 银行家算法举例

【例 2】 假设系统中有五个进程{P_0、P_1、P_2、P_3、P_4}和三类资源{A,B,C}，各种资源的数量分别为 10、5、7，在 T_0 时刻的资源分配情况如表 3-4 所示。

表 3-4 T_0 时刻的资源分配情况表

资源情况 / 进程	Max			Allocation			Need			Available		
	A	B	C	A	B	C	A	B	C	A	B	C
P_0	7	5	3	0	1	0	7	4	3	3	3	2
										(2	3	0)
P_1	3	2	2	2	0	0	1	2	2			
				(3	0	2)	(0	2	0)			
P_2	9	0	2	3	0	2	6	0	0			
P_3	2	2	2	2	1	1	0	1	1			
P_4	4	3	3	0	0	2	4	3	1			

(1) T_0 时刻的安全性：利用安全性算法对 T_0 时刻的资源分配情况进行分析，如表 3-5 所知，在 T_0 时刻存在着一个安全序列{P_1，P_3，P_4，P_2，P_0}，故系统是安全的。

表 3-5 T_0 时刻的安全序列

资源情况 / 进程	Work			Need			Allocation			Work+Allocation			finish
	A	B	C	A	B	C	A	B	C	A	B	C	
P_1	3	3	2	1	2	2	2	0	0	5	3	2	true
P_3	5	3	2	0	1	1	2	1	1	7	4	3	true
P_4	7	4	3	4	3	1	0	0	2	7	4	5	true
P_2	7	4	5	6	0	0	3	0	2	10	4	7	true
P_0	10	4	7	7	4	3	0	1	0	10	5	7	true

(2) P_1 请求资源：P_1 发出请求向量 $Request_1(1,0,2)$，系统按银行家算法进行检查：

① $Request_1(1,0,2) <= Need_1(1,2,2)$；

② $Request_1(1,0,2) <= Available_1(3,3,2)$；

③ 系统先假定可为 P_1 分配资源，并修改 Available、$Allocation_1$ 和 $Need_1$ 向量，由此形成的资源变化情况如表 3-4 中的圆括号所示；

④ 再利用安全性算法检查此时系统是否安全，如表 3-5 所示。

由所进行的安全性检查得知，可以找到一个安全序列{P_1，P_3，P_4，P_0，P_2}。因此，系统是安全的，可以立即将 P_1 所申请的资源分配给它，如表 3-6 所示。

表 3-6 P_1 申请资源时的安全性检查

资源情况 / 进程	Work			Need			Allocation			Work+Allocation			finish
	A	B	C	A	B	C	A	B	C	A	B	C	
P_1	2	3	0	0	2	0	3	0	2	5	3	2	true
P_3	5	3	2	0	1	1	2	1	1	7	4	3	true
P_4	7	4	3	4	3	1	0	0	2	7	4	5	true
P_0	7	4	5	7	4	3	0	1	0	7	5	5	true
P_2	7	5	5	6	0	0	3	0	2	10	5	7	true

(3) P_4 请求资源：P_4 发出请求向量 $Request_4(3,3,0)$，系统按银行家算法进行检查：

① $Request_4(3,3,0) <= Need_4(4,3,1)$；

② $Request_4(3,3,0) > Available(2,3,0)$，让 P_4 等待。

(4) P_0 请求资源：P_0 发出请求向量 $Request_0(0,2,0)$，系统按银行家算法进行检查：

① $Request_0(0,2,0)<=Need_0(7,4,3)$;

② $Request_0(0,2,0)<=Available(2,3,0)$;

③ 系统暂时假定可为 P_0 分配资源,并修改有关数据,如表 3-7 所示。

表 3-7 为 P_0 分配资源后的有关资源数据

资源情况 / 进程	Allocation			Need			Available		
	A	B	C	A	B	C	A	B	C
P_0	0	3	0	7	2	3	2	1	0
P_1	3	0	2	0	2	0			
P_2	3	0	2	6	0	0			
P_3	2	1	1	0	1	1			
P_4	0	0	2	4	3	1			

(5) 进行安全性检查:可用资源 Available(2,1,0)已不能满足任何进程的需要,故系统进入不安全状态,此时系统不分配资源。

通过这个例子可以看到,银行家算法确实能保证系统时时刻刻都处于安全状态,但它要不断检测每个进程对各类资源的占用和申请情况,需花费较多的时间。

视频讲解

3.6 死锁的检测

死锁的静态预防和动态避免都难以完全实现,且都不利于各进程对系统资源的充分共享。在实际中,死锁现象并不是经常在系统中出现,以至于大多数系统都不进行死锁的预防和避免。解决死锁问题的另一途径就是死锁检测和解除。死锁的检测和解除用于系统中定时运行一个“死锁检测”程序,判断系统内是否已出现死锁。一旦出现死锁,采取相应的措施解除它。

3.6.1 资源分配图

操作系统中的每一时刻的系统状态都可以用资源分配图(Resource Allocation Graph)来表示,资源分配图是描述进程和资源间申请及分配关系的一种有向图,用以检测系统是否处于死锁状态。设一个计算机系统中有许多类资源和许多个进程。每一个资源类用一个方框表示,方框中的黑圆点表示该资源类中的各个资源,每个进程用一个圆圈表示,用有向边来表示进程申请资源和资源被分配的情况。约定 $P_i \rightarrow R_j$ 为请求边,表示进程 P_i 申请资源类 R_j 中的一个资源得不到满足而处于等待 R_j 类资源的状态,该有向边从进程开始指到方框的边缘,表示进程 P_i 申请 R_j 类中的一个资源。反之,$R_j \rightarrow P_i$ 为分配边,表示 R_j 类中的一个资源已被进程 P_i 占用,由于已把一个具体的资源分给了进程 P_i,故该有向边从方框内的某个黑圆点出发指向进程。图 3-4 是进程资源分配图的一个例子,其中共有三个资源类,每个进程的资源占有及申请情况已表示在图中。这个例子中,由于存在占有和等待资源的环路,导致一组进程永远处于等待资源状态,发生了死锁。

进程资源分配图中存在环路,并不一定发生死锁。因为循环等待条件仅是死锁发生的必要条件,而不是充分条件,图 3-5 便是一个有环路而无死锁的例子。虽然进程 P_1 和进程 P_3 分别占有了一个资源 R_1 和一个资源 R_2,并且等待另一个资源 R_2 和另一个资源 R_1 形成

了环路，但进程 P_2 和进程 P_4 分别占有了资源 R_1 和资源 R_2 中的一个，它们申请的资源已得到了全部满足，因而能够在有限时间内归还占有的资源，于是进程 P_1 和进程 P_3 分别能获得另一个所需资源，这时进程资源分配图中减少了两条请求边，环路不再存在，系统中也就不存在死锁了。

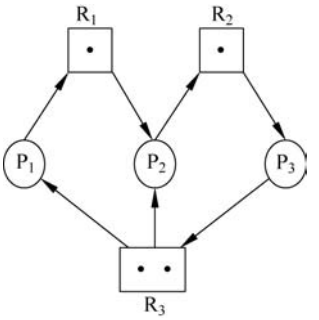

图 3-4 进程资源分配图的一个例子

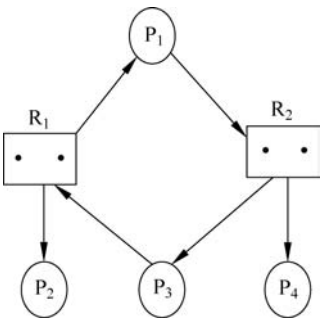

图 3-5 有环路而无死锁的一个例子

3.6.2 死锁定理

可以利用下列步骤运行一个“死锁检测”程序，对进程资源分配图进行分析和简化，以此方法来检测系统是否处于死锁状态。

(1) 如果进程资源分配图中无环路，则此时系统没有发生死锁。

(2) 如果进程资源分配图中有环路，且每个资源类中仅有一个资源，则系统中发生了死锁，此时，环路是系统发生死锁的充分条件，环路中的进程便为死锁进程。

(3) 如果进程资源分配图中有环路，且涉及的资源类中有多个资源，则环路的存在只是产生死锁的必要条件而不是充分条件，系统未必会发生死锁。如果能在进程资源分配图中找出一个既不阻塞又非独立的进程，它在有限的时间内有可能获得所需资源类中的资源继续执行，直到运行结束，再释放其占有的全部资源，在图 3-6(a)中，相当于消除了图中 P_1 的所有请求边和分配边，使之成为孤立结点。在图 3-6(b)中，接着可使进程资源分配图中另一个进程获得前面进程释放的资源继续执行，直到完成又释放出它所占用的所有资源，相当于又消除了图中 P_2 若干请求边和分配边。如此下去，经过一系列简化后，若能消除图中所有边，使所有进程成为孤立结点，形成如图 3-6(c)所示的情况，则该图可完全简化；否则称该图是不可完全简化的。系统为死锁状态的充分条件是：当且仅当该状态的进程资源分配图是不可完全简化的。该充分条件称为死锁定理。

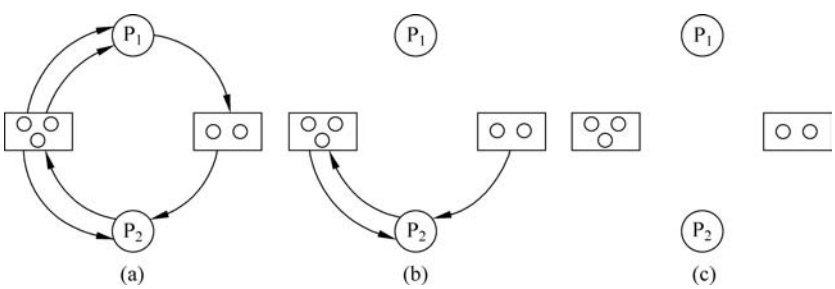

图 3-6 资源分配图的简化

3.6.3 死锁检测算法

当系统中每类资源的实例是多个时,可采用下面介绍的死锁检测算法进行检测。该算法由 Shoshani 和 Coffman 提出,采用了与银行家算法类似的数据结构。

算法中采用的数据结构如下。

(1) 当前可分配的空闲资源向量 Available(1：m)。m 是系统中的资源类型数。Available[i]表示系统中现有的 i 类资源数量。

(2) 资源分配矩阵 Allocation(1：n,1：m)。Allocation[i,j]表示进程 i 已占有的 j 类资源的数量。

(3) 需求矩阵 Request(1：n,1：m)。Request[i,j]表示进程 i 还需申请 j 类资源的数量。

死锁检测算法如下。

① 令 Work 和 Finish 分别表示长度为 m 和 n 的向量,初始化 Work=Available；对于所有 i=1,…,n,如果 Allocation[i]≠0,则 Finish[i]=false,否则 Finish[i]=true。

② 寻找一个下标 i,它满足条件：Finish[i]=false 且 Request[i]<=Work,如果找不到这样的 i,则转向步骤④。

③ Work=Work+Allocation[i]；Finish[i]=true；转向步骤②。

④ 如果存在 i,1<=i<=n,Finish[i]=false,则系统处于死锁状态。若 Finish[i]=false,则进程处于死锁环中。

在上面的算法中,如果一个进程所申请的资源能够满足,就假定该进程能得到所需的资源向前推进,直至结束,释放所占有的全部资源。接着查找是否有另外的进程也满足这种条件。如果某进程在以后还要不断申请资源,那么它还可能会被检测出死锁。

【例 3】 设系统中有 3 个资源类{r1,r2,r3}和 5 个并发进程{P_1,P_2,P_3,P_4,P_5},其中 r1 有 7 个,r2 有 3 个,r3 有 6 个。在 T_0 时刻各进程分配资源和申请情况表如表 3-8 所示。

表 3-8 T_0 时刻各进程分配资源和申请情况表

资源情况 / 进程	Allocation			Request			Available		
	r1	r2	r3	r1	r2	r3	r1	r2	r3
P_1	0	1	0	0	0	0	0	1	0
P_2	2	0	0	2	0	2			
P_3	3	0	3	0	0	0			
P_4	2	1	1	1	0	0			
P_5	0	0	2	0	0	2			

根据上面的死锁检测算法可以得到一个进程的安全序列<P_1,P_3,P_2,P_4,P_5>对于所有的 Finish[i]=true,所以,此时系统 T_0 时刻不处于死锁状态。假定,进程 P_2 现在申请一个单位为 r3 的资源,则系统资源分配情况表如表 3-9 所示。

表 3-9 满足进程 P_2 申请后的系统资源分配情况

资源情况 / 进程	Allocation			Request		
	r1	r2	r3	r1	r2	r3
P_1	0	1	0	0	0	0
P_2	2	0	0	2	0	2
P_3	3	0	3	0	0	1
P_4	2	1	1	1	0	0
P_5	0	0	2	0	0	2

此时，系统处于死锁状态，参与死锁的进程集合为$\{P_2, P_3, P_4, P_5\}$。

系统何时进行死锁检测呢？这取决于死锁进程出现的频率和当死锁出现时所影响进程的数量等因素。若死锁经常出现，检测死锁算法应该经常被调用。一种常用方法是当进程申请资源不能满足，就进行检测。如果死锁检测过于频繁，系统开销就增大；如果检测时间间隔过长，卷入死锁的进程数量又会增多，使得系统的资源和 CPU 的利用率大大下降，一个折中的办法就是定期检测，如每小时检测一次或当 CPU 的利用率低于 40%时检测。

3.7 死锁的解除

当死锁检测程序检测到死锁存在时，应设法将其解除，让系统从死锁状态中恢复过来，常用的解除死锁的办法有以下几种。

(1) 立即结束所有进程的执行，并重新启动操作系统。这种方法简单，但以前所做的工作全部作废，损失很大。

(2) 撤销涉及死锁的所有进程，解除死锁后继续运行。这种方法能彻底破坏死锁的循环等待条件，但将付出很大代价。例如有些进程可能已经计算了很长时间，由于被撤销而使产生的部分结果也被消除，重新执行时还要再次进行计算。

(3) 逐个撤销涉及死锁的进程，回收其资源，直至死锁解除。但是先撤销哪个死锁进程呢？可选择符合下面条件(之一)的进程先撤销：消耗的 CPU 时间最少者、产生的输出最少者、预计剩余执行时间最长者、占有资源数最少者或优先级最低者。

死锁解除后，应在适当的时候重新执行被撤销的进程，当重新启动进程时，应从哪一点开始执行呢？一种最简单的办法是让进程从头开始执行，但这样就要花费较高的代价。有的系统在进程执行过程中设置校验点，当重新启动时，让进程回退到发生死锁之前的那个校验点开始执行。设置校验点的办法对于执行时间长的进程来说是有必要的，但系统要花费较大的代价来记录进程的执行情况以及相应的恢复工作。

(4) 抢夺资源。从涉及死锁的一个或几个进程中抢夺资源，把夺得的资源再分配给涉及死锁的其他进程直到死锁解除。

采用抢夺资源的方法解决死锁问题时应考虑以下三个问题。

(1) 抢夺哪些进程的哪些资源。总是希望能以最小的代价结束死锁，因而必须关注涉及死锁的进程所占有的资源数，以及它们已经执行的时间等因素。

(2) 被抢夺者的恢复。如果一个进程的资源被抢夺了，它就无法继续执行，因而应该让它返回到某个安全状态并记录有关的信息，以便重新启动该进程执行。

(3) 进程的“饿死”。如果经常从同一个进程中抢夺资源,那么该进程总是处于资源不足的状态而不能完成所担负的任务,该进程就被“饿死”。所以,一般总是从执行时间短的进程中抢夺资源,以免“饿死”现象的发生。

3.8 死锁的综合处理策略

从表 3-1 各种处理死锁的基本方法的比较中可见,所有解决死锁的方法都各有其优缺点。与其将操作系统机制设计为只采用其中策略,还不如在不同情况下使用不同的策略更有效。于是提出一种综合的死锁策略:把资源分成几组不同的资源类,为预防在资源类之间由于循环等待产生死锁,可使用前面的线性排序策略。在一个资源类中,使用该类资源最适合的算法。作为该技术的一个例子,可以考虑下列资源类。

- 可交换空间:在进程交换中所使用的辅存储器(即“辅存”)中的存储块。
- 进程资源:可分配的设备,如磁带设备和文件。
- 主存:可以按页或按段分配给进程。
- 内部资源:例如 I/O 通道。

以上列出的次序表示了资源分配的次序。考虑到一个进程在其生命周期中的步骤顺序,这个序是最合理的。在每一类资源中,可以采用以下策略。

(1) 对于可交换空间,通过要求一次性分配所有请求的资源来预防死锁,就像占有且等待预防办法一样。如果知道最大存储需求(一般通常情况下都知道),则这个策略是合理的。死锁避免也是可能的。

(2) 对于进程资源,死锁避免的方法通常是有效的,这是因为进程可以事先声明它们将需要的这类资源。采用资源排序的预防策略也是可能的。

(3) 对于主存,基于抢占的预防是最适合的策略。当一个进程被抢占后,它仅仅被换到辅存,释放空间以解决死锁。

(4) 对于内部资源,可以使用基于资源按序排列的预防策略。

3.9 线 程 死 锁

在支持多线程的操作系统中,除了会发生进程之间的死锁外,还会发生线程之间的死锁。由于不同的线程可以属于同一个进程,也可以属于不同的进程。因此,与进程死锁比较,线程死锁分为属于同一进程的线程死锁和属于不同进程的线程死锁。

1) 同一进程的线程死锁

线程的同步工具有互斥锁。由于同一进程的线程共享该进程资源,为了实现线程对进程内变量的同步访问,可以采用互斥锁。假如,L_1 和 L_2 为两个互斥锁,进程内的一个线程先获得 L_1,然后申请获得 L_2,同一进程内的另一个线程先获得 L_2,再申请获得 L_1。这样一来,同一进程内的两个线程陷入死锁。

2) 不同进程的线程死锁

如果在进程 P_1 中主存在一组线程$\{P_{11},P_{12},\cdots,P_{1m}\}$,在进程 P_2 中主存在一组线程$\{P_{21},P_{22},\cdots,P_{2m}\}$。在同一时间段内,进程 P_1 内的线程获得资源 R_1,进程 P_2 内的线程获得

资源 R_2。如果进程 P_1 内的某个线程 P_{1i} 请求资源 R_2，由于不能满足而进入阻塞状态；进程 P_2 内某个线程 P_{2j} 请求资源 R_1，由于不能满足而进入阻塞状态。线程 P_{1i} 和线程 P_{2j} 相互等待对方释放资源，这时将出现不同进程线程间的死锁。

当将进程看作为单线程进程时，死锁进程的解决方法同样适用于同一进程的线程死锁和不同进程的线程死锁。

3.10 本章小结

死锁是多个并发进程因竞争资源及进程执行顺序非法而造成的一种状态。系统产生死锁的四个必要条件是互斥条件、占有且等待条件、不剥夺条件和循环等待条件。解决死锁的方法一般有预防、避免、检测和解除等四种。

(1) 预防是采用某种策略，限制并发进程对资源的请求，从而使得死锁的必要条件在系统执行的任何时间都不满足。例如，可以采用静态分配策略、抢占资源和层次分配策略来预防死锁。

(2) 避免则是指系统在分配资源时，根据资源的使用情况提前做出预测，从而避免死锁的发生。例如，可以采用银行家算法来避免死锁。

(3) 检测是指系统设有专门的机构，当死锁发生时，该机构能够检测到死锁发生，并精确地确定与死锁有关的进程和资源，通常可以用进程资源分配图来检测死锁。

(4) 解除是与检测相配套的一种措施，用于将进程从死锁状态下解脱出来。可以采取重启系统、撤销所有涉及死锁进程、逐个撤销涉及死锁的进程、抢夺资源等方法来解除死锁。

所有解决死锁的方法都各有其优缺点。与其将操作系统机制设计为只采用其中策略，还不如在不同情况下使用不同的策略更为有效。

本章最后介绍了线程死锁。与进程死锁比较，线程死锁分为属于同一进程的线程死锁和属于不同进程的线程死锁。当将进程看作为单线程进程时，死锁进程的解决方法同样适用于同一进程的线程死锁和不同进程的线程死锁。


视频讲解

习 题 3

(1) 何谓死锁？产生死锁的原因是什么？

(2) 产生死锁的四个必要条件是什么？

(3) 处理死锁的方法有哪几种？

(4) 死锁的预防的基本思想是什么？

(5) 如何破坏请求和保持条件？

(6) 如何破坏不剥夺条件？

(7) 如何破坏循环等待条件？

(8) 死锁的避免的基本思想是什么？

(9) 简述银行家算法的工作过程。

(10) 什么是进程的安全序列？何谓系统的安全状态？

(11) 在生产者-消费者问题中，如果对调生产者(或消费者)进程的两个 P 操作和两个

V操作的次序,会发生什么情况?试说明之。

(12) 一台计算机有8台磁带机,它们由N个进程竞争使用,每个进程可能需要3台磁带机。请问N为多少时,系统没有死锁的危险?并说明原因。

(13) 假定系统有4个同类资源和3个进程,进程每次只申请或释放1个资源。每个进程最大资源需求量为2。这个系统为什么不会发生死锁?

(14) 若系统有m个同类资源,被n个进程共享,分别在$m>n$和$m\leqslant n$时,每个进程最多可以请求多少个这类资源,从而使系统一定不会发生死锁?

(15) 设系统中有3种类型的资源(A,B,C)和5个进程(P1,P2,P3,P4,P5),A资源的数量为17,B资源的数量为5,C资源的数量为20。在T_0时刻系统状态表如表3-10所示。

表3-10 T_0时刻系统状态表

资源情况 / 进程	最大资源需求量			已分配资源量			剩余资源数		
	A	B	C	A	B	C	A	B	C
P1	5	5	9	2	1	2			
P2	5	3	6	4	0	2			
P3	4	0	11	4	0	5	2	3	3
P4	4	2	5	2	0	4			
P5	4	2	4	3	1	4			

系统采用银行家算法尝试死锁避免策略。

① T_0时刻是否为安全状态?若是,请给出安全序列。

② 在T_0时刻,若进程P2请求资源(0,3,4),是否能实施资源分配?为什么?

③ 在②的基础上,若进程P4请求资源(2,0,1),是否能实施资源分配?为什么?

④ 在③的基础上,若进程P1请求资源(0,2,0),是否能实施资源分配?为什么?

(16) 在银行家算法中,若出现如表3-11所示的资源分配情况:现在系统还剩资源A类2个,B类1个,C类2个,D类0个。请回答下面问题。

表3-11 系统资源分配情况表

进　程	已分配资源数	最大资源需求数
P0	0 0 1 2	0 0 2 2
P1	2 0 0 0	2 7 5 0
P2	0 0 3 4	6 6 5 6
P3	2 3 5 4	4 3 5 6
P4	0 3 3 2	0 6 5 2

① 现在系统是否处于安全状态?若是,请给出安全序列。

② 若现在进程P2请求资源(0,1,0,0),是否能实施资源分配?为什么?

(17) 有三个进程P1、P2和P3并发工作。进程P1需用资源R1和R3;进程P2需用资源R1和R2;进程P3需用资源R2和R3。

① 若对资源分配不加限制,会发生什么情况?为什么?

② 为保证进程正常的工作,应采用怎样的资源分配策略?为什么?

(18) 假设系统有5类独占资源:r1、r2、r3、r4、r5。各类资源分别有2、2、2、1、1个单位

的资源。系统有5个进程：P1、P2、P3、P4、P5，其中P1已占有2个单位的r1，且申请1个单位的r2和1个单位的r4；P2已占有1个单位的r2，且申请1个单位的r1；P3已占有1个单位的r2，且申请1个单位的r2和1个单位的r3；P4已占有1个单位的r4和1个单位的r5，且申请1个单位的r3；P5已占有1个单位的r3，且申请1个单位的r5。

① 试画出该时刻的资源分配图。

② 什么是死锁定理？如何判断①给出的资源分配图中有无死锁？请给出判断过程和结果。

(19) 假设一个多线程应用程序仅使用读写锁来同步。如果使用多个读写锁，根据死锁的四个必要条件，应用程序是否会发生死锁？试说明理由。

(20) 死锁的避免、预防和检测的区别是什么？