第5章 死　　锁

死锁是多道程序并发执行带来的又一严重问题，它是操作系统乃至并发程序设计中最难处理的问题之一。死锁产生的根本原因有两个：一是系统中的资源数目不能满足多个并发进程的全部资源需求，各进程竞争资源，如系统对资源分配不合理就会产生死锁，简记为：资源竞争；二是并发执行进程间的推进顺序不合理也可产生死锁，简记为：推进顺序不当。“资源竞争、推进顺序不当”，这10个字是本章的一条主线，本章所有内容都围绕它们展开。

死锁是所有操作系统都面临的潜在问题，死锁必须通过外力才能解决。死锁处理不好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给予死锁现象高度重视。学完本章我们会发现，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的方案。

本章主要讲解死锁的概念、死锁处理策略、死锁预防、死锁避免、死锁检测和解除。

【本章学习目标】

- 死锁的基本概念。
- 死锁产生的4个必要条件。
- 处理死锁的方法。
- 死锁预防与死锁避免的区别。
- 系统安全状态及其判别方法。
- 银行家算法及其应用。
- 死锁检测、死锁解除的概念和方法。

5.1 死锁的基本概念和产生原因

5.1.1 死锁的基本概念

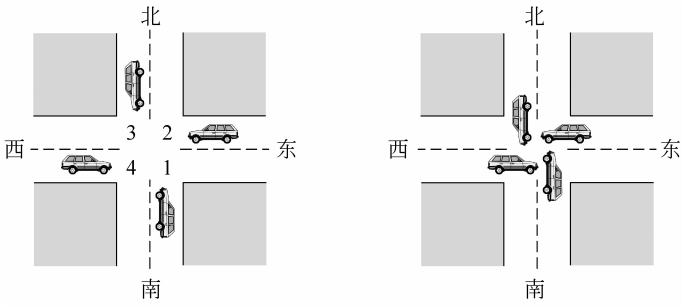
死锁是荷兰学者Dijkstra于1965年研究银行家算法时首先提出，此后Havender、Lynch等人分别于1968年、1971年相继对该问题取得共识并对其加以发展。并发执行的进程间存在着复杂同步关系，如果处理不好有时会出现后果严重的“死锁”现象。死锁是多道程序设计技术带来的又一个非常难处理的问题。

所谓死锁，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，在无外力干预下，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，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给予充分重视。

死锁问题不仅普遍存在于计算机系统中，日常生活中也广泛存在。先举几个典型例子帮助大家理解死锁。

先举一个现实生活中的常见的交通死锁问题。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 5.1(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，就会发生图 5.1(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。



(a) 可能死锁　　(b) 死锁

图 5.1　交通阻塞导致死锁示意图

在该例中，可把每个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 5.1(a)中标注的 1、2、3 和 4 所示。每个汽车进程在过路口时都要依次申请其中的两个资源。向北行驶的汽车依次申请 1 和 2 号资源；向西行驶的汽车依次申请 2 和 3 号资源；向南行驶的汽车依次申请 3 和 4 号资源；向东行驶的汽车依次申请 4 和 1 号资源。

出现死锁局面时，向北行驶的汽车占据 1 号资源申请 2 号资源；向西行驶的汽车占据 2 号资源申请 3 号资源；向南行驶的汽车占据 3 号资源申请 4 号资源；向东行驶的汽车占据 4 号资源申请 1 号资源。各个汽车进程彼此相互等待对方释放资源，同时不释放自己所拥有的资源，形成了一个相互等待链，导致死锁发生。

此时，若不采取外力措施干预，如交通警察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，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，某系统中有 P_1、P_2 两个进程并发执行，P_1 和 P_2 在执行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。

设进程 P_1、P_2 的执行过程如图 5.2 所示。

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，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 向前推进时想获得 CD-ROM 驱动器，但 CD-ROM 驱动器已被 P_2 占用，只能阻塞，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

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 5.2　两个并发执行进程的活动图

用，P_2 也只好阻塞，等待 P_1 释放该资源。两个进程此时陷入了相互等待，形成死锁。

第 3 章我们介绍的哲学家就餐问题中也存在着死锁风险。如果所有的哲学家恰巧同时拿起自己左边的筷子，此后各个哲学家都因不能获得右边筷子而阻塞。由于谁也不释放自己已占有的筷子，哲学家进程间形成死锁。

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 2 个进程，单个进程不能形成死锁。

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，在没有外力干预下，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。

5.1.2　产生死锁的原因

计算机系统产生死锁的根本原因有两个：一是系统中的资源数不能满足多个并发进程的全部资源需求，并发进程间竞争资源，当资源分配不合理时就会产生死锁，简记为资源竞争；二是并发执行进程间的推进顺序不合理也会导致死锁发生，简记为推进顺序不当。

下面逐一介绍这两个根本原因。

1. 资源竞争

死锁产生的根本原因是资源竞争其分配不当。因为多道程序并发执行，造成多个进程在执行中所需的资源数远远大于系统能提供的资源数。例如：图 5.2 所示例中的进程 P_1、P_2 之所以死锁，就是因为系统中的 CD-ROM 驱动器和打印机不够用。如果系统中配置了多台 CD-ROM 驱动器和打印机，进程 P_1 和 P_2 根本不会出现死锁。但是在一个系统中，仅仅为了防止死锁而配置多台相同的 CD-ROM 驱动器和打印机，从成本角度看是不现实的，各个进程对资源的需求量是动态变化的。例如某时系统中虽然有多个并发执行的进程，但它们对打印机提出的请求可能只有一个甚至没有，那么系统配置多台打印机就造成了极大的资源浪费。

计算机系统中有很多种资源，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 CPU、内存、磁盘等。

进程的可剥夺资源被剥夺后,系统保存该进程的断点信息。在合适的时候,该进程可从断点处恢复执行,继续使用该资源。

CPU 是典型的可剥夺资源。例如,当某进程获得 CPU 后,如果该进程的时间片用完,即使该进程没有发生阻塞还能继续使用 CPU 执行指令,CPU 也被系统剥夺分给其他进程。这也是我们前面介绍的进程并发执行的特点。如果 CPU 不允许被剥夺,并发执行的各进程只能串行,第一个进程执行完后才能执行第二个进程,依此类推。显然,这是单任务执行,不是并发执行。

内存也是可剥夺资源。假设有两个并发进程 A 和 B,它们大小均为 2KB。系统为这两个进程只分配了 2KB 的用户内存空间,即同一时刻只允许一个进程在内存中。当两个进程并发执行时,若进程 A 先被调度执行而进入内存,在没有执行完毕时,时间片用完,进程切换执行进程 B。进程 B 此时不能进入内存,因系统为进程 A 和进程 B 分配的用户空间已满。此时操作系统内存管理模块把进程 A 先对换到磁盘上,剥夺它所占有的内存分给进程 B 使用。进程 A 和进程 B 就这样并发执行,两者最终都能执行完毕。如果内存资源不允许被剥夺,那就只能先执行进程 A,A 执行完毕后,再执行进程 B,这仍然是串行执行。

注意:由于可剥夺性资源允许被其他进程剥夺,所以竞争可剥夺性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。这类资源众多,例如打印机、刻录机、CD-ROM 驱动器等。

不可剥夺资源的不可剥夺属性是由资源本身的特点决定的。例如打印机,一个进程获得打印机后,打印了部分内容。如果此时发生了进程切换,另一进程获得 CPU,它在执行过程中也要求使用打印机。当前打印机确实空闲,若分配给该进程使用,最终打印出的结果是混乱的,用户不能接受。

又例如,一个进程正通过刻录机刻录光盘,刻录期间其他进程对刻录机提出使用申请,系统如把刻录机分给另一进程,那么前一进程所刻光盘就变成废盘。刻录机在某进程使用完前,绝不允许和其他进程交替使用。

系统中所配置的非剥夺性资源数量往往不能满足多个并发进程的需要,从而产生资源竞争,如分配不当则会产生死锁。

死锁既可以发生在不可剥夺的硬件资源上,如图 5.2 所示例子中的打印机和 CD-ROM 驱动器;也可以发生在不可剥夺的软件资源上,如记录锁、信号量、消息、I/O 缓冲区中的信息等。例如某数据库系统中,一个进程对记录 R 加锁,另一个进程对记录 S 加锁,然后两个进程都试图将对方记录加锁,结果出现死锁。

第 3 章讲解的生产者-消费者问题中,如将生产者进程和消费者进程各自代码中的两个 P 操作颠倒,都先执行 P(mutex)操作,那么就存在发生死锁的风险。一个生产者进程获得了 CPU 的使用权,恰巧当前系统缓冲区已满。该生产者进程执行 P(empty)时,由于缓冲区已满,它在信号量 empty 上阻塞,等待消费者进程消费完产品后激活它。但该生产者进程

阻塞前已经对信号量 mutex 进行了 P 操作，P 当前值为 0。当消费者进程获得 CPU 使用权时，其也首先执行 P(mutex)，但由于 P 的当前值为 0，所以把 P 的值减 1 后也阻塞，等待生产者退出缓冲区时激活它。后续的生产者进程和消费者进程也都要先对信号量 mutex 进行 P 操作，所以也都阻塞。生产者进程和消费者进程彼此等待对方激活，但这永远不会发生，形成了死锁。所以，对信号量操作不当也会引起死锁。

消息也是一种能造成死锁的资源，如用户使用不当，就会出现死锁。例如某系统中有 3 个进程 P_1、P_2 和 P_3，它们之间需要传递消息 S_1、S_2、S_3，即：进程 P_1 发送消息 S_1 给进程 P_2，进程 P_2 发送消息 S_2 给进程 P_3，进程 P_3 发送消息 S_3 给进程 P_1。

如果每个进程都要发送消息和接收消息成功后才能向前推进，则可能有如下情况。

P_1：send(S_1)，receive(S_3)；

P_2：send(S_2)，receive(S_1)；

P_3：send(S_3)，receive(S_2)；

此时，P_1、P_2、P_3 均能顺利发送并得到所需的信息。

但是，如果各个进程都先接收消息后才发送消息，即：

P_1：receive(S_3)，send(S_1)；

P_2：receive(S_1)，send(S_2)；

P_3：receive(S_2)，send(S_3)；

在这种情况下，P_1、P_2、P_3 永远都不能接收到所需要的信息，出现死锁。

2. 推进顺序不当

并发执行的诸进程在运行中存在异步性，彼此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。

不可剥夺资源少未必一定产生死锁。死锁在一种很巧合的推进顺序中才会发生。在不能增加不可剥夺资源数量的前提下，我们要采取各种措施，尽量避免产生死锁的不合理推进顺序出现。

为了揭示进程推进顺序对死锁的影响，重新分析一下图 5.2 所示的例子。为了更具普遍性，把打印机和 CD-ROM 驱动器分别用不可剥夺资源 A 和不可剥夺资源 B 代替。图 5.3 给出了进程 P 和 Q 竞争资源 A 和 B 的进展情况，两个进程都独占使用 A、B 资源一段时间。

进程 P	进程 Q
⋮	⋮
申请资源 A	申请资源 B
⋮	⋮
申请资源 B	申请资源 A
⋮	⋮
释放资源 A	释放资源 B
⋮	⋮
释放资源 B	释放资源 A
⋮	⋮

图 5.3　两个并发执行进程活动图

为了更好地描述进程 P 和进程 Q 的推进顺序，图 5.4 给出了两者的推进顺序示意图。

横轴表示进程 P 的执行进展，纵轴表示进程 Q 的执行进展。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。在单 CPU 系统中，某一时刻只能有一个进程处于执行态，所以图中的路径线都是垂直或水平的。垂直线表示进程 Q 获得 CPU 执行，而进程 P 处于就绪态或阻塞态；水平线表示进程 P 获得 CPU 执行，进程 Q 处于就绪态或阻塞态。

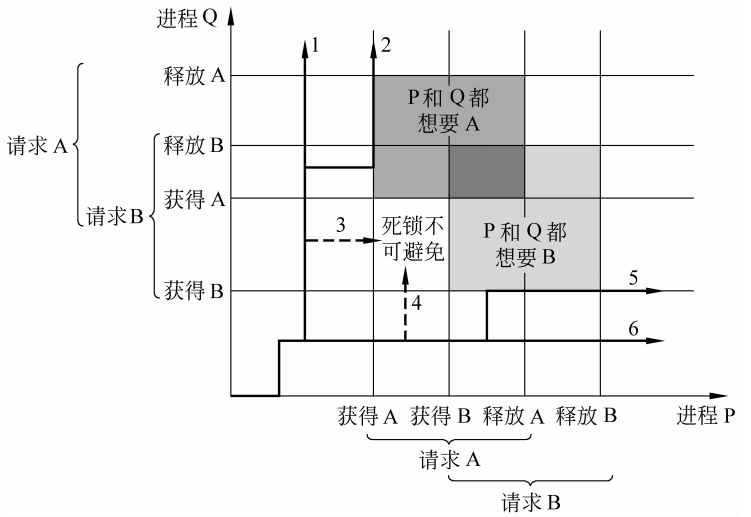


图 5.4　进程 P 和 Q 的推进顺序示意图

图 5.4 中给出了 6 种不同的执行路径，表示 6 中不同的进程间推进顺序。

(1) 进程 Q 申请并获得资源 B 和资源 A，执行结束后释放资源 B 和资源 A。然后，进程 P 被调度执行，它可以获得所需的资源 A 和资源 B。进程 Q 和进程 P 均顺利执行完毕。

(2) 进程 Q 申请并获得资源 B 和资源 A，此时进程发生切换，进程 P 被调度执行。进程 P 申请资源 A，而此时资源 A 已分配给进程 Q，进程 P 只能阻塞。然后，进程 Q 再次被调度执行，执行结束后释放资源 B 和资源 A。这时，进程 P 变为就绪态被调度程序调度，系统能满足 P 所需的 A、B 资源，执行完毕。

(3) 进程 Q 先执行，申请并获得资源 B。此时进程发生切换，进程 P 被调度执行，它申请并获得了资源 A。此时，无论进程 Q 还是进程 P 谁被调度选中执行，都会出现阻塞。进程 Q 因申请资源 A 不能获得而阻塞，进程 P 因申请资源 B 不能获得也阻塞。两个进程互相等待对方释放资源，陷入死锁。

(4) 进程 P 先执行，申请并获得资源 A。此时进程发生切换，进程 Q 被调度执行，它申请并获得资源 B。此时，与路径③类似，无论进程 Q 还是进程 P 谁被调度选中执行，都会出现阻塞，两进程陷入死锁。

(5) 进程 P 先被调度，申请并获得资源 A 和资源 B。此时进程发生切换，进程 Q 被调度执行；进程 Q 申请资源 B 而资源 B 已分配给进程 P，进程 Q 阻塞。进程 P 被再次调度执行，它使用完资源 A 和资源 B 后释放。这时，进程 Q 变为就绪态被调度程序调度，系统能满足 Q 所需的所有资源，执行完毕。

(6) 进程 P 申请并获得资源 A 和资源 B，执行结束后释放资源 A 和资源 B。然后，进程 Q 被调度执行，它可以获得所需的资源 B 和资源 A。进程 P 和进程 Q 均顺利执行完毕。

细心的读者会发现，进程 P 和进程 Q 只有在很巧合的情况下才会出现死锁。多数情况下，二者不形成死锁。上例的 6 种推进顺序中，只有(3)和(4)两种推进顺序发生死锁。

在不能增加系统中不可剥夺资源数量的前提下，应力争并发执行进程按照合理的顺序推进，这就可避免死锁的发生。但这在实现中存在较大困难，多个并发进程间的推进顺序数量巨大，多种多样，很难一一进行考察，排除所有产生死锁的推进顺序。

由以上分析，我们得出以下两个结论。

(1) 用户编写应用程序时或操作系统进行资源分配时应采取相应措施，避免导致死锁发生的进程间的推进顺序出现。

为避免死锁现象的产生，用户在编写应用程序时，除了考虑如何实现某种功能外，还要考虑避免死锁发生。如上例中，进程 P 不是同时申请两个资源，改成先申请资源 A，使用完后立即释放 A，再申请资源 B，使用完后释放 B。进程 Q 不用做任何修改。这样不管两个进程以何种顺序推进，都不会出现死锁。当然这无形当中增加了应用程序编写者的编程负担。

从另一个角度看，应用程序编写者编程时不考虑死锁问题，而是交由操作系统处理。操作系统在进行资源分配前进行分析，如不产生死锁则分配，产生死锁则不响应进程的资源请求。这样也能有效地避免死锁发生，但这增加了操作系统的开发难度和运行开销。

(2) 死锁是在特定的推进顺序中才会出现，具有一定的隐蔽性。

有的死锁隐藏的很深，只有在很巧合的情况下，导致死锁的推进顺序才会发生。发生死锁的可能性远低于不发生死锁的可能性。应用程序编写者或操作系统设计者往往需要花费大量精力去测试程序或系统，才能发现死锁并解决它。

实际工程中，软件工程师们通常综合权衡死锁发生的频度和可能造成的后果。如果死锁平均每 5 年发生一次，而由硬件故障、程序漏洞等所造成的系统瘫痪频度远远高于 5 年一次，那么就没有必要为避免死锁而付出巨额的系统代价。UNIX 和 Windows 等商用系统都采用这种做法。而计算机用户也宁愿容忍偶然性故障带来的损失，而不愿系统经常性地进行死锁处理而大量牺牲系统性能。

5.2 死锁的必要条件

死锁发生必须具备一定的条件。Coffman 首先提出死锁发生的 4 个必要条件，故又称为 Coffman 条件。

由于这 4 个条件是必要条件，一旦死锁出现，这 4 个条件都必然成立。因此，只要有一个必要条件被系统或用户破坏掉、不再出现，则死锁也一定不会发生。

死锁产生的 4 个必要条件如下。

(1) 互斥条件。在某段时间内，系统某资源只能由一个进程占用，如果此时还有其他进程对该资源发出请求，只能阻塞等待，直到占有该资源的进程用完释放，其他进程才能占有该资源。例如：打印机、CD-ROM、扫描仪等不可剥夺性资源。这个条件是由资源本身的不可剥夺属性决定的。

(2) 请求和保持条件。进程已经拥有了至少一个资源，之后又提出新的资源请求，但新请求资源此时已被其他进程占有，该进程因而阻塞。阻塞期间，进程对已获得的资源保持不放。参与死锁的进程中至少有两个进程占有资源，并申请其他资源。

(3) 不剥夺条件。进程已获得的资源在未使用完之前不能被系统或其他进程剥夺,只有在使用完毕后由进程自己释放。

由此条件我们可知,可剥夺资源均不会造成死锁。但是,并不是所有资源都是可剥夺的,有相当一部分资源是不可剥夺的。

(4) 环路等待条件。系统中存在一个进程等待序列$\{p_1, p_2, \cdots, p_n\}$,其中,$p_1$ 等待 p_2 所占有的某个资源,p_2 等待 p_3 所占有的某个资源,……,p_n 等待 p_1 所占有的某个资源,从而形成一条进程循环等待环。环路中的进程必处于死锁状态。

环路等待条件隐含着前 3 个条件,即只有前 3 个条件成立,第 4 个条件才会成立。

注意: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环,但存在循环等待环不一定产生死锁。这点希望读者引起注意。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 3 个进程并发执行,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 p_1 和 p_2 形成的循环等待环将被打破,死锁解除。

5.3 死锁的处理

5.3.1 死锁的处理方法

死锁普遍存在于并发执行进程间,处理死锁的方法有很多种。最简单的处理方法是鸵鸟算法(Ostrich Algorithm),对死锁现象视而不见,不予理睬。这虽然不是一个积极的方法,但是它却是目前通用操作系统中采用最多的方法。实际系统中,防止死锁的代价很高,往往比系统重启 100 次的代价还高。这也是 Windows、UNIX、Linux 等现代操作系统都没有采取死锁防止措施的原因。

如果涉及死锁处理的是高可靠性系统或实时控制系统,鸵鸟算法则不宜采用。这些系统的设计者们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,可把死锁的处理方法分成 4 类。

(1) 预防死锁

预防死锁是在系统运行之前就采取相应措施,确定不会发生死锁的资源分配算法,消除发生死锁的任何可能性。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,但因简单明了并且安全可靠,现仍被广泛采用。

清除死锁发生的必要条件可预防死锁。破坏 4 个必要条件中的一个或几个,死锁则无法产生。预防死锁的代价往往是降低系统资源的利用率和减少系统吞吐量。

(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。

死锁避免与死锁防止策略不同,它不对进程申请资源施加任何限制,而是对进程提出的每一次资源请求进行动态检查。依据检查结果决定是否分配资源,确保该资源请求批准后系统不会进入死锁状态或潜在死锁状态。对于会使系统进入死锁状态或潜在死锁状态的资

源请求，系统拒绝资源分配。

避免死锁虽好，但也存在两个缺点：一是对每个进程申请资源命令的分析计算较为复杂，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，而该数值恰是考察资源分配可行性的重要参数之一。

请读者注意死锁避免和死锁预防的区别。死锁预防的限制条件比较严格，实现起来较为简单，但往往降低系统效率；死锁避免的限制条件相对宽松，资源分配后需要通过算法来判断是否存在出现死锁的可能，实现起来较为复杂。

(3) 检测死锁

检测死锁即确定死锁现象，准确识别出陷入死锁的进程和资源。确定死锁发生较为复杂，尤其是系统中并发进程数较多时，很难做出精确的判断。有时检查死锁的进程自己也会陷入死锁，系统此时自然无法准确地检测死锁。

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，会给系统或用户造成一定的损失。

(4) 解除死锁

确定死锁发生后，系统可通过相关措施解除死锁。常用方法是抢占某个进程占有的资源、撤销或挂起一些进程，力争以较低的系统代价解除死锁。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。

5.3.2 资源分配图

资源分配图(Resource Allocation Graph)是描述死锁问题的常用工具，在很多问题的求解中都有应用。

资源分配图是一个有向图，该图由结对组成：G＝(V，E)。式中 V 是顶点集，E 是边集。

定点集 V 由 P 和 R 两部分构成，$P=\{P_1,P_2,\cdots,P_n\}$为系统中的进程集合，$R=\{r_1,r_2,\cdots,r_m\}$为系统中所有资源类型的集合。边集 E 也由两类有向边组成，从进程 P_i到资源 r_j的有向边记做 $P_i\rightarrow r_j$，代表进程 P_i提出申请一个单位的 r_j资源；从资源 r_j到进程 P_i的有向边记做 $r_j\rightarrow P_i$，代表一个单位的 r_j资源已经分配给了进程 P_i。形如 $P_i\rightarrow r_j$的边被称作申请边；形如 $r_j\rightarrow P_i$的边被称作分配边。

一个进程通常用一个圆圈表示，圈内为进程的名字。一个资源类通常用一个方框表示，方框内的圆点表示系统中本类资源的个数。

当进程 P_i申请资源 r_j中的一个资源实例时，在资源分配图中增加一条申请边。当该申请可满足时，该申请边立即被改为一条分配边。当进程释放该资源实例时，该分配边被去掉。

下边我们举例说明如何使用资源分配图。某资源分配图如图 5.5 所示。

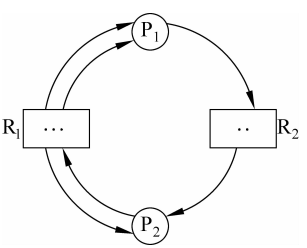


图 5.5 资源分配图示例

由图 5.5 可知，系统中有两个进程 P_1和 P_2，两个资源类 R_1和 R_2。通过对有向边的分析可知，进程 P_1获得了 2 个 R_1类资源实例，申请一个 R_2类资源实例。进程 P_2获得了一个 R_1类资源实例和一个 R_2类资源实例，申请一个 R_1类资源实例。

根据死锁的定义我们可知，如果资源分配图中没有环路，系统就不会陷入死锁状态；如存在环路，系统极有可能出现死锁。但环路并不是充分条件，还需进一步的考察环路中的各个进程是否还能继续向前推进，如果不能则可断定系统出现了死锁。

例如，图 5.5 中虽出现了环路，但两个进程 P_1 和 P_2 仍能正常结束，不会产生死锁。而图 5.6 所示的资源分配图中也出现了环路，通过进一步考察发现，进程 P_1 和 P_2 已不能再向前推进，出现死锁。

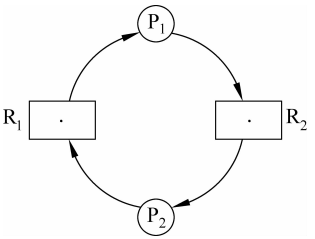


图 5.6　资源分配图示例

5.4　死锁的静态预防

死锁预防是保证系统不进入死锁状态的静态策略，它的基本思想是采取一定的措施破坏 4 个死锁必要条件中的一个或几个，从而预防死锁的发生。

5.4.1　破坏互斥条件

如果允许系统中的所有资源都能共享使用，即破坏死锁的“互斥”使用条件，系统将不会发生死锁。但是，临界资源本身的性质决定其不能共享使用，否则无法保证正确性。打印机就是一个典型代表，如进程能共享使用打印机，则打印出的结果没有任何意义。系统不但不能破坏“互斥”条件，还要采取各种办法保证独占资源的互斥使用。

在实际系统中，可采用时分或空分虚拟技术把必须互斥使用的、不可剥夺的独占设备改造成为可被多个进程共享的虚拟设备，预防死锁的发生。例如采用 SPOOLing 技术把打印机改造成虚拟打印机，任何一个想要打印的进程将打印内容先发送到可共享的“输入井”（磁盘的一个固定区域）中，发送完毕后，该进程即认为打印完毕，而真正的打印输出需等到打印机空闲时才进行。

5.4.2　破坏请求和保持条件

破坏这个条件的办法很简单，可采用预分配资源方法。进程运行前，系统一次性地分配其运行所需的所有资源，而不是随着进程的推进而陆续分配。由于进程运行前一次性地获得了所需的所有资源，该进程可顺利执行完毕，不会发生死锁。

这种方法虽简单、易行且安全，但是系统资源浪费严重，进程执行明显延迟。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。采用预分配策略后，系统开始就把 CD-ROM 驱动器和打印机分配给该进程。这样一来，进程读入数据时，打印机空闲；进程计算时，CD-ROM 驱动器和打印机都闲置；进程输出时，CD-ROM 驱动器闲置，极大地浪费了系统资源。此外，在进程执行前分配资源时，若进程所需资源中只有某个资源没满足，该进程就不能执行，一直要等到所需的资源都满足时才能执行。一般情况下，进程所缺的某些资源在进程开始时并不是必需的，进程执行到必须使用该资源时，该资源此时可能已空闲可用。预分配资源策略极大地延迟了进程运行，降低了系统的并发程度。

预分配资源方法在实现上还有一个很大的缺陷，即系统无法在进程执行前精确估计其