进程死锁及解决办法

操作系统 2009-09-24 16:48:58 阅读767 评论1 字号：大中小 订阅

一、要点提示

（1） 掌握死锁的概念和产生死锁的根本原因。

（2） 理解产生死锁的必要条件--以下四个条件同时具备：互斥条件、不可抢占条件、占有且申请条件、循环等待条件。

（3） 记住解决死锁的一般方法，掌握死锁的预防和死锁的避免二者的基本思想。

（4） 掌握死锁的预防策略中资源有序分配策略。

（5） 理解进程安全序列的概念，理解死锁与安全序列的关系。

（6） 了解银行家算法。

（7） 了解资源分配图。

（8） 了解死锁的检测及恢复的思想。

二、内容简介

在计算机系统中有很多一次只能由一个进程使用的资源，如打印机，磁带机，一个文件的I节点等。在多道程序设计环境中，若干进程往往要共享这类资源，而且一个进程所需要的资源不止一个。这样，就会出现若干进程竞争有限资源，又推进顺序不当，从而构成无限期循环等待的局面。这种状态就是死锁。系统发生死锁现象不仅浪费大量的系统资源，甚至导致整个系统崩溃，带来灾难性后果。所以，对于死锁问题在理论上和技术上都必须给予高度重视。

8.1 死锁的概念

死锁是进程死锁的简称，是由Dijkstra于1965年研究银行家算法时首先提出来的。它是计算机操作系统乃至并发程序设计中最难处理的问题之一。实际上，死锁问题不仅在计算机系统中存在，在我们日常生活中它也广泛存在。

1.什么是死锁

我们先看看这样一个生活中的例子：在一条河上有一座桥，桥面较窄，只能容纳一辆汽车通过，无法让两辆汽车并行。如果有两辆汽车A和B分别由桥的两端驶上该桥，则对于A车来说，它走过桥面左面的一段路（即占有了桥的一部分资源），要想过桥还须等待B车让出右边的桥面，此时A车不能前进；对于B车来说，它走过桥面右边的一段路（即占有了桥的一部分资源），要想过桥还须等待A车让出左边的桥面，此时B车也不能前进。两边的车都不倒车，结果造成互相等待对方让出桥面，但是谁也不让路，就会无休止地等下去。这种现象就是死锁。如果把汽车比做进程，桥面作为资源，那麽上述问题就描述为：进程A占有资源R1，等待进程B占有的资源Rr；进程B占有资源Rr，等待进程A占有的资源R1。而且资源R1和Rr只允许一个进程占用，即：不允许两个进程同时占用。结果，两个进程都不能继续执行，若不采取其它措施，这种循环等待状况会无限期持续下去，就发生了进程死锁。

在计算机系统中，涉及软件，硬件资源都可能发生死锁。例如：系统中只有一台CD-ROM驱动器和一台打印机，某一个进程占有了CD-ROM驱动器，又申请打印机；另一进程占有了打印机，还申请CD-ROM。结果，两个进程都被阻塞，永远也不能自行解除。

所谓死锁，是指多个进程循环等待它方占有的资源而无限期地僵持下去的局面。很显然，如果没有外力的作用，那麽死锁涉及到的各个进程都将永远处于封锁状态。从上面的例子可以看出，计算机系统产生死锁的根本原因就是资源有限且操作不当。即：一种原因是系统提供的资源太少了，远不能满足并发进程对资源的需求。这种竞争资源引起的死锁是我们要讨论的核心。例如：消息是一种临时性资源。某一时刻，进程A等待进程B发来的消息，进程B等待进程C发来的消息，而进程C又等待进程A发来的消息。消息未到，A，B，C三个进程均无法向前推进，也会发生进程通信上的死锁。另一种原因是由于进程推进顺序不合适引发的死锁。资源少也未必一定产生死锁。就如同两个人过独木桥，如果两个人都要先过，在独木桥上僵持不肯后退，必然会应竞争资源产生死锁；但是，如果两个人上桥前先看一看有无对方的人在桥上，当无对方的人在桥上时自己才上桥，那麽问题就解决了。所以，如果程序设计得不合理，造成进程推进的顺序不当，也会出现死锁。

2.产生死锁的必要条件

从以上分析可见，如果在计算机系统中同时具备下面四个必要条件时，那麽会发生死锁。换句话说，只要下面四个条件有一个不具备，系统就不会出现死锁。

〈1〉互斥条件。即某个资源在一段时间内只能由一个进程占有，不能同时被两个或两个以上的进程占有。这种独占资源如CD-ROM驱动器，打印机等等，必须在占有该资源的进程主动释放它之后，其它进程才能占有该资源。这是由资源本身的属性所决定的。如独木桥就是一种独占资源，两方的人不能同时过桥。

〈2〉不可抢占条件。进程所获得的资源在未使用完毕之前，资源申请者不能强行地从资源占有者手中夺取资源，而只能由该资源的占有者进程自行释放。如过独木桥的人不能强迫对方后退，也不能非法地将对方推下桥，必须是桥上的人自己过桥后空出桥面（即主动释放占有资源），对方的人才能过桥。

〈3〉占有且申请条件。进程至少已经占有一个资源，但又申请新的资源；由于该资源已被另外进程占有，此时该进程阻塞；但是，它在等待新资源之时，仍继续占用已占有的资源。还以过独木桥为例，甲乙两人在桥上相遇。甲走过一段桥面（即占有了一些资源），还需要走其余的桥面（申请新的资源），但那部分桥面被乙占有（乙走过一段桥面）。甲过不去，前进不能，又不后退；乙也处于同样的状况。

〈4〉循环等待条件。存在一个进程等待序列{P1，P2，...，Pn}，其中P1等待P2所占有的某一资源，P2等待P3所占有的某一源，......，而Pn等待P1所占有的的某一资源，形成一个进程循环等待环。就像前面的过独木桥问题，甲等待乙占有的桥面，而乙又等待甲占有的桥面，从而彼此循环等待。

上面我们提到的这四个条件在死锁时会同时发生。也就是说，只要有一个必要条件不满足，则死锁就可以排除。

8.2 死锁的预防

前面介绍了死锁发生时的四个必要条件，只要破坏这四个必要条件中的任意一个条件，死锁就不会发生。这就为我们解决死锁问题提供了可能。一般地，解决死锁的方法分为死锁的预防，避免，检测与恢复三种（注意：死锁的检测与恢复是一个方法）。我们将在下面分别加以介绍。

死锁的预防是保证系统不进入死锁状态的一种策略。它的基本思想是要求进程申请资源时遵循某种协议，从而打破产生死锁的四个必要条件中的一个或几个，保证系统不会进入死锁状态。

〈1〉打破互斥条件。即允许进程同时访问某些资源。但是，有的资源是不允许被同时访问的，像打印机等等，这是由资源本身的属性所决定的。所以，这种办法并无实用价值。

〈2〉打破不可抢占条件。即允许进程强行从占有者那里夺取某些资源。就是说，当一个进程已占有了某些资源，它又申请新的资源，但不能立即被满足时，它必须释放所占有的全部资源，以后再重新申请。它所释放的资源可以分配给其它进程。这就相当于该进程占有的资源被隐蔽地强占了。这种预防死锁的方法实现起来困难，会降低系统性能。

〈3〉打破占有且申请条件。可以实行资源预先分配策略。即进程在运行前一次性地向系统申请它所需要的全部资源。如果某个进程所需的全部资源得不到满足，则不分配任何资源，此进程暂不运行。只有当系统能够满足当前进程的全部资源需求时，才一次性地将所申请的资源全部分配给该进程。由于运行的进程已占有了它所需的全部资源，所以不会发生占有资源又申请资源的现象，因此不会发生死锁。但是，这种策略也有如下缺点：

（1）在许多情况下，一个进程在执行之前不可能知道它所需要的全部资源。这是由于进程在执行时是动态的，不可预测的；

（2）资源利用率低。无论所分资源何时用到，一个进程只有在占有所需的全部资源后才能执行。即使有些资源最后才被该进程用到一次，但该进程在生存期间却一直占有它们，造成长期占着不用的状况。这显然是一种极大的资源浪费；

（3）降低了进程的并发性。因为资源有限，又加上存在浪费，能分配到所需全部资源的进程个数就必然少了。

（4）打破循环等待条件，实行资源有序分配策略。采用这种策略，即把资源事先分类编号，按号分配，使进程在申请，占用资源时不会形成环路。所有进程对资源的请求必须严格按资源序号递增的顺序提出。进程占用了小号资源，才能申请大号资源，就不会产生环路，从而预防了死锁。这种策略与前面的策略相比，资源的利用率和系统吞吐量都有很大提高，但是也存在以下缺点：

（1）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是件困难事，并增加了系统开销；

（2）为了遵循按编号申请的次序，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。

8.3 死锁的避免

上面我们讲到的死锁预防是排除死锁的静态策略，它使产生死锁的四个必要条件不能同时具备，从而对进程申请资源的活动加以限制，以保证死锁不会发生。下面我们介绍排除死锁的动态策略--死锁的避免，它不限制进程有关申请资源的命令，而是对进程所发出的每一个申请资源命令加以动态地检查，并根据检查结果决定是否进行资源分配。就是说，在资源分配过程中若预测有发生死锁的可能性，则加以避免。这种方法的关键是确定资源分配的安全性。

1.安全序列

我们首先引入安全序列的定义：所谓系统是安全的，是指系统中的所有进程能够按照某一种次序分配资源，并且依次地运行完毕，这种进程序列{P1，P2，...，Pn}就是安全序列。如果存在这样一个安全序列，则系统是安全的；如果系统不存在这样一个安全序列，则系统是不安全的。

安全序列{P1，P2，...，Pn}是这样组成的：若对于每一个进程Pi，它需要的附加资源可以被系统中当前可用资源加上所有进程Pj当前占有资源之和所满足，则{P1，P2，...，Pn}为一个安全序列，这时系统处于安全状态，不会进入死锁状态。

虽然存在安全序列时一定不会有死锁发生，但是系统进入不安全状态（四个死锁的必要条件同时发生）也未必会产生死锁。当然，产生死锁后，系统一定处于不安全状态。

2.银行家算法

这是一个著名的避免死锁的算法，是由Dijstra首先提出来并加以解决的。

[背景知识]

一个银行家如何将一定数目的资金安全地借给若干个客户，使这些客户既能借到钱完成要干的事，同时银行家又能收回全部资金而不至于破产，这就是银行家问题。这个问题同操作系统中资源分配问题十分相似：银行家就像一个操作系统，客户就像运行的进程，银行家的资金就是系统的资源。

[问题的描述]

一个银行家拥有一定数量的资金，有若干个客户要贷款。每个客户须在一开始就声明他所需贷款的总额。若该客户贷款总额不超过银行家的资金总数，银行家可以接收客户的要求。客户贷款是以每次一个资金单位（如1万RMB等）的方式进行的，客户在借满所需的全部单位款额之前可能会等待，但银行家须保证这种等待是有限的，可完成的。

例如：有三个客户C1，C2，C3，向银行家借款，该银行家的资金总额为10个资金单位，其中C1客户要借9各资金单位，C2客户要借3个资金单位，C3客户要借8个资金单位，总计20个资金单位。某一时刻的状态如图所示。

C1 2(7)

C2 2(1)

C3 4(4)

余额2

C1 2(7)

C3 4(4)

余额4

C1 2(7)

余额8

余额10

(a)

(b)

(c)

(d)

银行家算法示意

对于a图的状态，按照安全序列的要求，我们选的第一个客户应满足该客户所需的贷款小于等于银行家当前所剩余的钱款，可以看出只有C2客户能被满足：C2客户需1个资金单位，小银行家手中的2个资金单位，于是银行家把1个资金单位借给C2客户，使之完成工作并归还所借的3个资金单位的钱，进入b图。同理，银行家把4个资金单位借给C3客户，使其完成工作，在c图中，只剩一个客户C1，它需7个资金单位，这时银行家有8个资金单位，所以C1也能顺利借到钱并完成工作。最后（见图d）银行家收回全部10个资金单位，保证不赔本。那麽客户序列{C1，C2，C3}就是个安全序列，按照这个序列贷款，银行家才是安全的。否则的话，若在图b状态时，银行家把手中的4个资金单位借给了C1，则出现不安全状态：这时C1，C3均不能完成工作，而银行家手中又没有钱了，系统陷入僵持局面，银行家也不能收回投资。

综上所述，银行家算法是从当前状态出发，逐个按安全序列检查各客户谁能完成其工作，然后假定其完成工作且归还全部贷款，再进而检查下一个能完成工作的客户，......。如果所有客户都能完成工作，则找到一个安全序列，银行家才是安全的。

从上面分析看出，银行家算法允许死锁必要条件中的互斥条件，占有且申请条件，不可抢占条件的存在，这样，它与预防死锁的几种方法相比较，限制条件少了，资源利用程度提高了。

这是该算法的优点。其缺点是：

〈1〉这个算法要求客户数保持固定不变，这在多道程序系统中是难以做到的。

〈2〉这个算法保证所有客户在有限的时间内得到满足，但实时客户要求快速响应，所以要考虑这个因素。

〈3〉由于要寻找一个安全序列，实际上增加了系统的开销。

8.4 死锁的检测与恢复

一般来说，由于操作系统有并发，共享以及随机性等特点，通过预防和避免的手段达到排除死锁的目的是很困难的。这需要较大的系统开销，而且不能充分利用资源。为此，一种简便的方法是系统为进程分配资源时，不采取任何限制性措施，但是提供了检测和解脱死锁的手段：能发现死锁并从死锁状态中恢复出来。因此，在实际的操作系统中往往采用死锁的检测与恢复方法来排除死锁。

死锁检测与恢复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的位置和原因，并能通过外力破坏死锁发生的必要条件，从而使得并发进程从死锁状态中恢复出来。

1.放大观看>>)

图中所示为一个小的死锁的例子。这时进程P1占有资源R1而申请资源R2，进程P2占有资源R2而申请资源R1，按循环等待条件，进程和资源形成了环路，所以系统是死锁状态。进程P1，P2是参与死锁的进程。

下面我们再来看一看死锁检测算法。算法使用的数据结构是如下这些：

占有矩阵A：n\*m阶，其中n表示并发进程的个数，m表示系统的各类资源的个数，这个矩阵记录了每一个进程当前占有各个资源类中资源的个数。

申请矩阵R：n\*m阶，其中n表示并发进程的个数，m表示系统的各类资源的个数，这个矩阵记录了每一个进程当前要完成工作需要申请的各个资源类中资源的个数。

空闲向量T：记录当前m个资源类中空闲资源的个数。

完成向量F：布尔型向量值为真（true）或假（false），记录当前n个并发进程能否进行完。为真即能进行完，为假则不能进行完。

临时向量W：开始时W：=T。

算法步骤：

（1）W：=T，

对于所有的i=1，2，...，n，

如果A[i]=0，则F[i]：=true；否则，F[i]：=false

（2）找满足下面条件的下标i：

F[i]：=false并且R[i]〈=W

如果不存在满足上面的条件i，则转到步骤（4）。

（3）W：=W+A[i]

F[i]：=true

转到步骤（2）

（4）如果存在i，F[i]：=false，则系统处于死锁状态，且Pi进程参与了死锁。什麽时候进行死锁的检测取决于死锁发生的频率。如果死锁发生的频率高，那麽死锁检测的频率也要相应提高，这样一方面可以提高系统资源的利用率，一方面可以避免更多的进程卷入死锁。如果进程申请资源不能满足就立刻进行检测，那麽每当死锁形成时即能被发现，这和死锁避免的算法相近，只是系统的开销较大。为了减小死锁检测带来的系统开销，一般采取每隔一段时间进行一次死锁检测，或者在CPU的利用率降低到某一数值时，进行死锁的检测。

2.死锁的恢复

一旦在死锁检测时发现了死锁，就要消除死锁，使系统从死锁状态中恢复过来。

（1）最简单，最常用的方法就是进行系统的重新启动，不过这种方法代价很大，它意味着在这之前所有的进程已经完成的计算工作都将付之东流，包括参与死锁的那些进程，以及未参与死锁的进程。

（2）撤消进程，剥夺资源。终止参与死锁的进程，收回它们占有的资源，从而解除死锁。这时又分两种情况：一次性撤消参与死锁的全部进程，剥夺全部资源；或者逐步撤消参与死锁的进程，逐步收回死锁进程占有的资源。一般来说，选择逐步撤消的进程时要按照一定的原则进行，目的是撤消那些代价最小的进程，比如按进程的优先级确定进程的代价；考虑进程运行时的代价和与此进程相关的外部作业的代价等因素。

此外，还有进程回退策略，即让参与死锁的进程回退到没有发生死锁前某一点处，并由此点处继续执行，以求再次执行时不再发生死锁。虽然这是个较理想的办法，但是操作起来系统开销极大，要有堆栈这样的机构记录进程的每一步变化，以便今后的回退，有时这是无法做到的。