

XI`AN TECHNOLOGICAL UNIVERSITY

课程设计报告

课程名称 计算机操作系统

专 业： 计算机科学与技术

班 级： 16060104

姓 名： 杨旭

学 号： 16060104117

指导教师： 姜虹

成 绩：

2019 年 1 月 4 日

**摘要**

避免死锁是处理死锁问题的一个有效策略，它是在资源的动态分配过程中，防止系统进入不安全状态，以避免发生死锁。这种方法施加的限制条件弱，可以获得较好的系统性能。因而，避免死锁算法的研究显得尤为重要

本文应用了最具代表性的避免死锁的算法——Dijkstra的银行家算法，通过在visual studio 2012编程环境编写C程序，并成功验证了该算法的有效性，证明了该算法完全可以满足避免死锁的要求



1. **绪论**

1．1前言  
Dijkstra (1965)提出了一种能够避免死锁的调度算法，称为银行家算法。  
它的模型基于一个小城镇的银行家，他向一群客户分别承诺了一定的贷款额度，每个客户都有一个贷款额度，银行家知道不可能所有客户同时都需要最大贷款额，所以他只保留一定单位的资金来为客户服务，而不是满足所有客户贷款需求的最大单位。  
这里将客户比作进程，贷款比作设备，银行家比作系统。  
客户们各自做自己的生意，在某些时刻需要贷款。在某一时刻，客户已获得的贷款和可用的最大数额贷款称为与资源分配相关的系统状态。一个状态被称为是安全的，其条件是存在一个状态序列能够使所有的客户均得到其所需的贷款。如果忽然所有的客户都申请，希望得到最大贷款额，而银行家无法满足其中任何一个的要求，则发生死锁。不安全状态并不一定导致死锁，因为客户未必需要其最大贷款额度，但银行家不敢抱这种侥幸心理。  
银行家算法就是对每一个请求进行检查，检查如果满足它是否会导致不安全状态。若是，则不满足该请求；否则便满足。  
检查状态是否安全的方法是看他是否有足够的资源满足一个距最大需求最近的客户。如果可以，则这笔投资认为是能够收回的，然后接着检查下一个距最大需求最近的客户，如此反复下去。  
如果所有投资最终都被收回，则该状态是安全的，最初的请求可以批准。  
1．2研究意义  
在多道程序系统中，多个进程的并发执行来改善系统的资源利用率，提高系统的吞吐量，但可能发生一种危险——死锁。所谓死锁(Deadlock)，是指多个进程在运行过程中因争夺资源而造成的一种僵局（DeadlyEmbrace），当进程处于这种状态时，若无外力作用，他们都无法在向前推进。  
要预防死锁，有摒弃“请求和保持”条件，摒弃“不剥夺”条件，摒弃“环路等待”条件等方法。  
但是，在预防死锁的几种方法之中，都施加了较强的限制条件；而在避免死锁的方法中，所施加的限制条件较弱，有可能获得令人满意的系统性能。在该方法中把系统状态分为安全状态和不安全状态，便可避免死锁的发生。  
而最具代表性的避免死锁的算法，便是Dijkstra的银行家算法。  
利用银行家算法，我们可以来检测CPU为进程分配资源的情况，决定CPU是否响应某进程的请求并为其分配资源，从而很好避免了死锁的产生。

1. **需求分析**
   1. 问题陈述

在多道程序系统中，虽然能够借助于多个进程的并发执行，来改善系统资源的利用率，提高系统的吞吐量，但是依然有风险存在，那就是一锁死。 所谓锁死是指，多个进程在运行中因争夺资源而造成的一种僵局，当进程的这种僵持状态时，若无外力作用，它们将无法再向前推进。组程序中， 每个进程都无限等待被该组进程中的另进程所占有的资源，因而水远无法得到资源，这种现象就叫做进程死锁。

银行家算法是一种最具有代表性的避免死锁的算法。

要解释银行家算法，必须先解释操作系统的安全状态和不安全状态。所谓安全状态，是指系统能按照某种进程顺序{P1,P2. Pn}(称{P1,P2.., Pn }序列为安全序列)，来为每个进程Pi分配其所需资源，直至满足每个进程对资源的最大需求，使每个进程都可以顺利完成。安全状态一定 没有死锁发生。

如果系统无法找到这样-一个安全序列，则称系统处于不安全状态。所谓安全序列，如果对每一个进程Pi(1<i<n)， 它以后尚需要的资源量不超过系统当前可利用的资源量与所有的进程Pj6<n)所占有的资源量之和，则称此进程序列{P1,2..，Pn}是安全的，称作安全序列。

* 1. 死锁的定义

如果一组进程中的每个进程都在等待仅有该组进程中的其他进程才能引发的事件，那么该组进程是死锁的（Deadlock）

* 1. 死锁的必要条件

1. 互斥条件。进程对所分配到的资源进行排他性使用，即在一段时间内，某资源只能被一个进程使用。如果此时还有其他进程请求该资源，则请求进程只能等待，直至占有该资源的进程用毕释放。
2. 请求和保持条件。进程已经保持了至少一个资源，但又提出了新的资源请求，而该资源已被其他进程占有，此时请求进程被阻塞，但对自己以获得的资源保持不放。
3. 不可抢占资源。进程以获得的资源在未被使用完之前不能被抢占，只能在进程使用完时由自己释放。
4. 循环等待条件。在发生死锁时，必然存在一个进程—资源循环链，即进程集合{P0,P1,P2,…,Pn}中的P0正在等待P1占用的资源，P1正在等待P2占用的资源，……,Pn正在等待已被P0占用的资源。
   1. 死锁的处理方法
5. 预防死锁。这是一种较简单和直观的事先预防方法。该方法是通过设置某些限制条件，去破坏产生死锁四个必要条件中的一个或几个来预防产生死锁。预防死锁是种较易实现的方法，已被广泛使用。
6. 避免死锁。同样是属于事先预防策略，但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，而是在资源的动态分配过程中，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。
7. 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。
8. 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。常用的方法是撤消一些进程，回收它们的资源，将它们分配给已处于阻寨状态的进程，使其能继续运行。  
   上述的四种方法，从1到4对死锁的防范程度逐渐减弱，但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（并发程度提高）  
   * 1. **避免死锁**

避免死锁同样属于实现与事先预防的策略但并不是实现采取某种限制措施，破坏产生死锁的必要条件，而是在资源动态分配过程中，防止系统进入不安全状态，以避免发生死锁。这种方法所施加的限制条件弱，可以获得较好的系统性能，目前常用此方法来避免死锁。

在死锁避免方法中，把系统状态分为安全状态和不安全状态。当系统处于安全状态时，可避免死锁。反之，当系统处于不安全状态时，则可能进入到死锁状态。

最具代表的避免死锁的算法是Dijkstra的银行家算法。

1. **概要设计**
   1. 设计思路

首先使用安全性算法检测当前系统中存在的进程是否安全，即此刻是否存在避免死锁的进程执行序列。若当前状态不安全需重新初始化系统中的进程。在确保安全后，用户可以提出进程请求，系统判断用户请求是否能够予以满足，若请求合法，则进行资源试分配，对试分配后的状态使用安全性算法进行检测，在确保安全后，对用户的请求进行资源分配，否则将本次的试探分配作废，恢复原来的资源分配状态，让用户请求的进程等待

* 1. 执行步骤

设进程Pi提出请求Request i [j]，则银行家算法按如下规则进行判断。

(1)如果Request i [j]<= Need[i,j]，则转（2)；否则，出错。

(2)如果Request i [j]<= Available[j]，则转（3)；否则，等待。

(3)系统试探分配资源，修改相关数据：

Available[j]-=Request i [j];

Allocation[i,j]+=Request[i,j];

Need[i,j]-=Request[i,j];

(4)系统执行安全性检查，如安全，则分配成立；否则试探险性分配作废，系统恢复原状，进程等待。

* 1. 安全性算法设计

1)设置两个工作向量Work=Available;Finish

(2)从进程集合中找到一个满足下述条件的进程，

Finish[i]==false;

Need[i.j]<=Work[j];

如找到，执行（3)；否则，执行（4)

(3)设进程获得资源，可顺利执行，直至完成，从而释放资源。

Work[j]=Work[j]+Allocation[i.j];

Finish[i]=true;

GOTO 2

(4)如所有的进程Finish= true，则表示安全；否则系统不安全。

* 1. 数据结构

1)可利用资源向量Available

是个含有m个元素的数组，其中的每一个元素代表一类可利用的资源数目。如果Available[j]=K，则表示系统中现有Rj类资源K个。

2）最大需求矩阵Max

这是一个n×m的矩阵，它定义了系统中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Rj类资源的最大数目为K。

3）分配矩阵Allocation

这也是一个n×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Rj类资源的 数目为K。

4）需求矩阵Need。

这也是一个n×m的矩阵，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Rj类资源K个，方能完成其任务。

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

1. **详细设计**
   1. 主要函数
2. void ProbeAlloc(int process, RESOURCE \*res) 资源试分配函数

在进程请求资源数合法的情况下，系统试探着把资源分配给请求进程，用此函数来修改资源数据结构的值

1. void RollBack(int process, RESOURCE \*res) 恢复资源分配状态函数

试分配后系统处于不安全状态，则将本次的试探分配作废，用此函数来恢复原来的资源分配状态

1. bool SafeCheck()安全性检测函数

执行安全性算法检查系统是否处于安全状态

1. bool request(int process, RESOURCE \*res) 进程资源请求函数

发出资源请求，并在确保请求合法的情况下调用void ProbeAlloc(int process, RESOURCE \*res) 系统试探着把资源分配给请求进程，用此函数来修改资源数据结构的值，并调用安全性检测函数bool SafeCheck()检查试探分配后系统是否处于安全状态，若不安全还需调用恢复资源分配状态函数void RollBack(int process, RESOURCE \*res)将本次的试探分配作废，恢复原来的资源分配状态

* 1. 数据判断

1. 对请求的进程名进行判断，当请求进程名0>i&&i>=PROCESS\_NUMBE时，提示该进程不存在，请重新输入。
2. 对进程请求资源数进行判断，如果Request i [j]>= Need[i,j],提示进程请求资源数大于所需资源数。如果Request i [j]>= Available[j]，提示系统尚无足够资源满足进程要求，需等待。