文章编号：2095-2163(2022)07-0206-05 中图分类号：TP399 文献标志码：A

基于银行家算法的资源调度研究与实现

刘 璇
(贵阳信息科技学院 信息工程系，贵阳 550025)

摘 要：由于计算机系统资源具有某些特性,将会导致多进程并发时产生资源的竞争,银行家算法是避免死锁的一种有效方法,能提前预测系统是否处于安全状态。银行家算法利用系统可用资源向量、最大需求矩阵、已分配资源矩阵、还需矩阵 4 种数据结构来进行资源分配。本文采用 C 语言编程,设计并实现了银行家算法。通过仿真实验证明,该算法在一定程度上能有效的避免死锁产生。

关键词：多进程；银行家算法；安全序列；死锁

Research and implementation of resource scheduling based on banker algorithm

LIU Xuan
(Department of Information Engineering, Guiyang Institute of information technology, Guiyang 550025, China)

【Abstract】 Some resources of computer system have some characteristics, such as exclusivity, which will lead to resource competition when multiple processes are concurrent. Banker algorithm is an effective method to avoid deadlock and can predict whether the system is in a safe state in advance. Banker algorithm uses some data structures, such as system available resource vector, maximum demand matrix, to allocate resources. The banker algorithm is designed and implemented in C language. The simulation results show that the algorithm can effectively avoid deadlock to a certain extent.

【Key words】 multi process; banker algorithm; safety sequence; deadlock

0 引 言

在现代操作系统中,利用多道进程的并发执行来提高资源的利用率和吞吐量,为用户程序提供了良好的运行环境,但同时也带来一些新问题。由于计算机系统资源是有限的,并且部分资源还具有独占和排他的特性[1]。计算机系统中打印机的数量、计算机的内存容量、输入输出设备等的数量均是有限的,进程在申请这些资源时只能互斥的使用。如多个进程同时申请同一台打印机时,进程之间会发生互相抢夺,由于打印机数量有限,只能有一个进程得到满足,如果分配不合理,将会造成死锁[1-3]。

1 死锁

死锁(Deadlock)是指计算机系统中存在多个进程,在执行过程中,相互竞争资源或在进程之间进行某些数据传递、申请等而造成的一种互相等待的现象。若无外界因素的干扰,多个进程都将无法继续推进,所有进程都处于互相等待状态,此时系统产生死锁[3]。

由于资源的使用是互斥的,假设当某个进程提出申请资源时,此资源正在被别的进程所占用,在不可剥夺条件下,若无外力协助,该进程得不到资源的满足。假设每个进程都在等待被其它进程占用的资源,而其它进程也缺少资源,进程在执行完毕前不可能主动放弃资源,这就陷入一种死循环状态,所有进程都在互相等待一种不可能发生的情况。计算机系统中,如果系统的资源分配策略不当,更常见的可能是程序员写的程序有错误等,则会导致进程因竞争资源不当而产生死锁的现象。如有进程 P_1、P_2、P_3 和资源 A、B、C,3 个进程所占有的资源和申请的资源如图 1 所示。

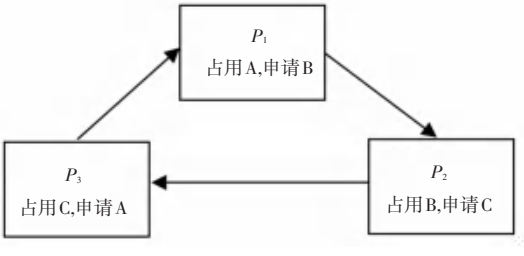

图 1 进程资源申请图

Fig. 1 Process resource application diagram

图中 P_1 进程占用 A 资源同时申请 B 资源；P_2 进程占用 P_1 进程申请的 B 资源,同时申请 C 资源；P_3 进程占用 P_2 进程申请的 C 资源,同时申请 P_1 进

作者简介：刘 璇(1994-),女,硕士,助教,主要研究方向:物联网工程、智能信息处理。

收稿日期：2022-01-16

程占用的 A 资源。由此可见,3 个进程之间申请的资源和占有的资源形成一个封闭的环状,在资源不可剥夺情况下,3 个进程都在等待别的进程释放自己所需的资源,而别的进程没有执行完毕,不可能释放资源,此时进程陷入永久等待,任何一个进程都得不到满足,都不会执行完毕,都不会释放资源,这就是一种典型的陷入死锁状态。

计算机系统产生死锁的最主要原因是系统资源有限,进程对资源的竞争产生了死锁,所以并发进程对分配资源的顺序就会有一定要求,若能合理的推进顺序,则能在很大程度上降低系统产生死锁的概率[4]。

2　银行家算法

1965 年,Dijkstra 根据银行系统现金贷款发放思想,提出了避免系统陷入死锁的算法,将其命名为银行家算法(Bankers algorithm)[5]。该算法首先检测进程申请的资源数量系统是否能满足;通过与系统现存可用资源数进行对比分析,若申请资源数小于等于可用资源数,说明当前资源能够满足该进程,可进行资源分配;若大于,则说明当前资源不能进行资源分配,否则系统产生死锁。若在进程的执行过程中还需继续申请资源,则要判断本次申请资源数和已占有资源数之和是否大于进程的最大资源数,若大于,拒绝分配;若小于,还需继续检测系统当前可用资源能否满足进程的此次申请;若能满足进程申请,给予分配,否则拒绝分配[6-9]。

2.1　银行家算法中的数据结构

假定系统中有 n 个并发进程 $P_1,P_2,\cdots,P_n$,m 类资源 $R_1,R_2,\cdots,R_m$。在银行家算法中需定义 1 个向量和 4 个矩阵。

(1)系统可用资源向量 ***Available***[m]:表示系统中各类可用资源数量。初始值是系统中 m 类资源全部可用数量,该值随资源的分配与回收动态变化。如:***Available***[***i***] = k,表示系统中现有可用 i 类资源的数量为 k 个。

(2)最大需求矩阵 Max[n][m]:表示系统中的 n 个进程,每个进程对 m 类资源的最大需求量。如:Max[i][j] = k,表示进程 P_i 需要 R_j 类资源的数量为 k 个。

(3)分配矩阵 ***Allocation***[n][m]:表示系统中各进程已占用的各类资源数。如:***Allocation***[i][j] = k,表示进程 P_i 已获得 R_j 类资源的数量为 k 个。

(4)需求矩阵 ***Need***[n][m]:表示每个进程所需的各类资源数。如:***Need***[i][j] = k,表示进程 P_i 还需要分配 R_j 类资源 k 个即可执行完毕。

上述 3 个矩阵之间存在如下关系:

Need[i][j] = Max[i][j] − ***Allocation***[i][j]

(5)请求矩阵 ***Request***[n][m]:表示每个进程当前申请分配各类资源数。如:***Request***[i][j] = k,表示进程 P_i 需要 k 个 R_j 类的资源。

2.2　银行家算法描述

设 ***Request*** 是进程 P 的请求向量,如果 ***Request***[j] = K,表示进程 P_i 需要 K 个 R 类型的资源。当进程 P_i 发出资源请求(***Request***1,2,…,0)后(表示 m 类资源分别申请 1,2,…,0 个),系统按以下步骤进行检测,判断是否进行资源分配。

(1) 当 ***Request***[i][j] > ***Need***[i][j] 时,不能进行资源分配。因为进程 P_i 所申请的资源数已超过其最大需求量,进程 P_i 出错。

(2) 当 ***Request***[i][j] > ***Available***[j] 时,不能分配资源,进程 P_i 进入等待状态。

(3) 除以上两种情况外,系统可给予分配,但是需要修改相应的数据结构为:

Available[j] = ***Available***[j] − ***Request***[i][j]

Alocatio[i][j] = ***Allionta***[i][j] + ***Request*** [i][j]

Need[i][j] = ***Need***[i][j]] − ***Request***[i][j]

(4)检测系统安全性。调用安全性算法检查系统安全状态,如果检测结果为安全状态,则给进程 P_i 分配所申请资源;否则进程 P_i 不能分配资源,并修改进程为等待资源状态,恢复下列数据结构后返回。

Available[j] = ***Available***[j] + ***Request***[i][j]

Allocation[i][j] = ***Allocation***[i][j] − ***Request***[i][j]

Need[i][j] = ***Need***[i][j] + ***Request***[i][j]

2.3　安全性算法

安全性算法是银行家算法的子算法(如 2.2 节中步骤 4)。为了保证安全性检查,在不影响 ***Available***、Max、***Allocation*** 和 ***Need*** 的状态下,需新建两个向量(临时变量)***Work***、***Finish*** 的数据结构,用以检验系统的安全状态。

其中,工作向量 ***Work***[m] 表示系统可分配给进程使用的各类资源数(有 m 类资源),其初始值与 ***Available*** 相等;完成向量 ***Finish***[n] 表示系统是否有足够的资源分配给所有待分配资源的进程。若 ***Finish***[i] > ***Available***[i],则表示系统资源量不满足;反之可以满足。

安全性检测算法实现步骤如下:

Step 1　对工作向量和完成向量进行初始化:

$$\boldsymbol{Work}[j] = \boldsymbol{Available}[j] \quad j = 1,2,\cdots,m$$

$$\boldsymbol{Finish}[i] = false \quad i = 1,2,\cdots,n$$

工作向量 ***Work*** 初始值与 ***Available*** 相等，***Finish*** 中的所有位全为 *false*。当有足够资源分配给进程 P_i 时，再令 $\boldsymbol{Finish}[i] = true$。

Step 2 从进程集合中寻找满足条件：$\boldsymbol{Finish}[i] = falseNee、d[i][j] \leqslant \boldsymbol{Work}[j]$ 的一个进程。若满足条件，则表明系统当前资源能满足进程 P_i 的所有资源申请，转去执行步骤 3；否则表示系统不能满足当前所有进程的资源申请，则执行步骤 4。

Step 3 当进程 P_i 分配到资源后，将继续执行直至完成整个进程。之后，释放所占用的全部资源，转回步骤 2 继续调度下一个可满足资源申请的进程。此时工作向量和完成向量调整如下：

$$\boldsymbol{Work}[j] = \boldsymbol{Work}[j] + \boldsymbol{Allocation}[i][j]$$

Step 4 对于任意 $i(i = 1,2,\cdots,m)$，都使得 $\boldsymbol{Finish}[i]$ 的值为真，则系统处于安全状态；否则，系统处于不安全状态。

执行安全性算法的本质，就是找到一个符合当前系统资源的安全序列。如果该序列存在，则系统所有进程都可顺利往前推进。系统按照该序列调度进程，系统就不会产生死锁现象，使操作系统处于安全状态。

3 算法建模

算法建模分为两个模块：银行家算法模块和安全性算法模块。用户通过输入数据，分别对可利用资源矩阵 ***Available***、最大需求矩阵 Max、分配矩阵 ***Allocation***、需求矩阵 ***Need*** 赋初值。采用 C 语言编程实现[10-12]。

3.1 银行家算法模块

银行家算法模块主要是通过对进程所申请资源的 ***Request***、***Need***、***Available*** 向量之间关系进行判断，判断系统当前资源能否满足该进程，能否对该进程进行资源分配。实现过程如下：

(1)如果满足 $\boldsymbol{Request} \leqslant \boldsymbol{Need}$ 条件，表示进程申请的资源数小于等于该进程运行所需的所有资源数，则转向步骤(2) 继续判断；否则，系统出错。

(2)如果 $\boldsymbol{Request} \leqslant \boldsymbol{Available}$ 条件成立，表示进程所申请的资源数小于或等于当前系统可用资源数，满足该进程提出的申请，则分配资源给进程；否则，进程处于阻塞态。

(3)系统执行安全性算法。

3.2 安全性算法模块

安全性算法模块主要是对系统的安全性进行检测，通过遍历所有进程 P_i，对完成向量 ***Finish*** 的值进行判断，从而判断系统是否处于安全队列。实现过程如下：

(1)设置两个向量

①工作向量：$\boldsymbol{Work} = \boldsymbol{Available}$，表示系统可用的各类资源数，其初值与 ***Available*** 相等；

②完成向量 ***Finish***：表示系统是否有足够资源满足进程，*true* 表示有，*false* 表示没有。

(2)若 $\boldsymbol{Finish}[i] = false \&\& \boldsymbol{Need} \leqslant \boldsymbol{Work}$，则执行(3)；否则执行(4)($i$ 为资源类别)

(3)给进程 P_i 分配所申请资源，进程执行完成时回收所有资源。$\boldsymbol{Work} = \boldsymbol{Work} + \boldsymbol{Allocation}$；$\boldsymbol{Finish}[i] = true$；转(2)。

(4)若所有进程都使得 $\boldsymbol{Finish}[i] = true$，则表示系统处于安全状态；反之系统处于不安全状态。

4 设计与实现

设计中寻找安全序列的部分使用循环结构完成，通过分别处理循环的终止条件确定系统是否安全。若满足条件 $\boldsymbol{Finish}[i]$ 的值为 *false*，并且 $\boldsymbol{Need}[i,j]$ 小于或等于 $\boldsymbol{Work}[j]$ 时，说明系统处于安全状态，此时将 $\boldsymbol{Finish}[i]$ 向量置为 *true*。循环中设置一个变量来累加计数，当该变量与进程数量相等时，说明已将 $\boldsymbol{Finish}[i]$ 向量全部置为 *true*，则终止循环。

对于不安全系统，不可将向量 $\boldsymbol{Finish}[i]$ 都置为 *true*，必定存在 *false*。由于需要不断循环查找并尝试分配，在寻求一个安全序列时，若在一轮的寻找中没有一个可以安全执行的进程，则说明往后也不存在安全的进程，即可跳出循环，结束寻找。银行家算法流程如图 2 所示。

资源分配过程关键代码如下：

```
/*开始进行预分配*/
    num  = 0;
if(num == resourceNum)
  {
num = 0;
for(int j = 0; j < resourceNum; j++)
{
/*分配资源*/
      Available[j] = Available[j] - request[j];
      Allocation[curProcess][j]  =  Allocation
[curProcess][j] + request[j];
```

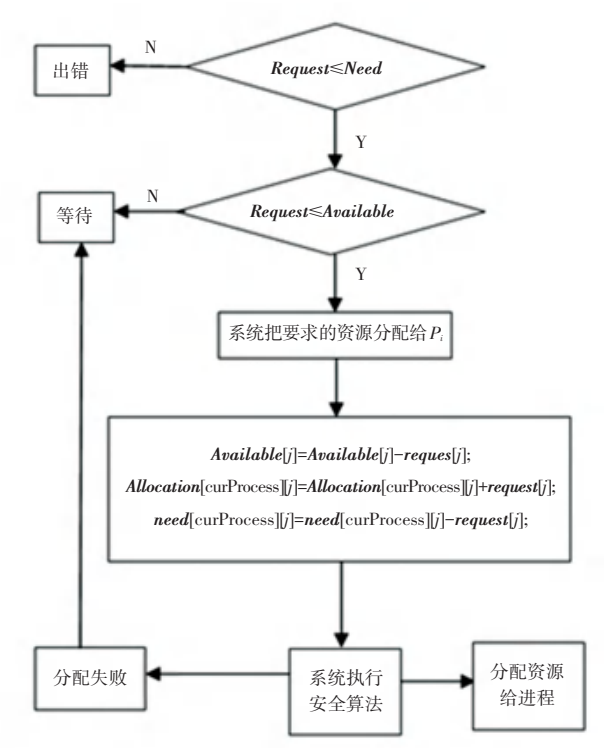

图 2　银行家算法流程图

Fig. 2　Banker algorithm flow chart

```
need[curProcess][j] = need[curProcess][j]
- request[j];
/*记录分配以后,是否该进程需要值为 0*/
if(need[curProcess][j] == 0)
num ++;
}
/*如果分配以后出现该进程对所有资源的需求为 0,即刻释放该进程占用资源(视为完成)*/
if(num == resourceNum)
{
/*释放已完成资源*/
for(int i = 0;i < resourceNum;i ++ )
Available[i] = Available[i] + Allocation
[curProcess][i];
printf("\n 本次分配进程 P%d 完成,该进程占用资源全部释放完毕! \n",curProcess);
}
else
{
/*资源分配可以不用一次性满足进程需求*/
printf("\n 本次分配进程 P%d 未完成! \n",
curProcess);
}
/*预分配完成以后,判断该系统是否安全*/
if(! isSafe())
{
for(int j = 0; j < resourceNum; j ++)
{
Available[j] = Available[j] + request[j];
Allocation[curProcess][j] = Allocation
[curProcess][j] - request[j];
need[curProcess][j] = need[curProcess][j]
+ request[j];
}
printf("资源不足,分配失败! \n");
}
```

程序运行结果截图如图 3 所示。

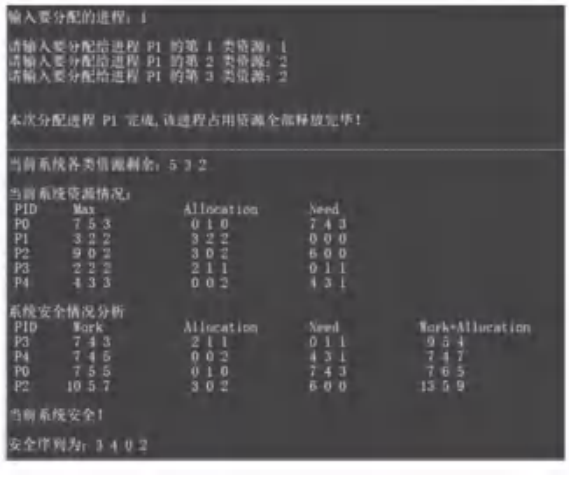

图 3　程序运行结果

Fig. 3　Program running result diagram

5　结束语

经仿真实验得出,银行家算法虽然无法从本质上解决死锁问题,但是该算法可以提高多进程并发算法的资源利用率,能预测当前系统是否处于安全状态,在避免死锁的方面有较突出的表现。由于死锁产生的根本原因是资源的数量不足,并且银行家算法在计算过程中要不断的检测每个进程占有资源、还需资源以及系统可用资源的情况,整个过程将耗费大量时间。为此,相关问题还有待进一步探索研究。

参考文献

[1] 张成姝. 操作系统教程[M]. 清华大学出版社,2019:121-124.
[2] 梁家荣, 徐霜, 伍华健. 基于扩展安全级的 Torus 网络容错路由算法研究[J]. 计算机工程与应用, 2009, 45(29): 99-101.

(下转封三)