doi：

矩阵在计算机线程死锁的检测与防止中的应用

XXX

西安电子科技大学 计算机学院，陕西 西安 710071；

摘要：在多线程操作系统中，死锁是一个必须解决的问题，本文对操作系统中经常出现的死锁问题进行了讨论，阐述了死锁出现的原因、必要条件，提出了使用线程和资源的邻接矩阵对操作系统死锁状态进行判断，谈论了一个避免死锁的经典算法——银行家算法，并进行了实际验证模拟。最后提出了死锁状态的恢复办法。

关键词：多线程 死锁 邻接矩阵 银行家算法

中图分类号：TP3-05 文献标识码：A 文章编号：

Application of Matrix in Detection and Prevention of Deadlock in Computer Thread

School of Computer, Xidian University, Xi'an 710071, China;

Abstract: In the multi-threaded operating system, deadlock is a problem that must be solved. In this paper, the deadlock problem often found in the operating system is discussed. The reasons and the necessary conditions of deadlock are expounded, and the use of threads and resources The adjacency matrix is used to judge the deadlock state of the operating system, and a classic algorithm to avoid deadlock - banker algorithm is discussed and the actual verification is carried out. Finally, the deadlock state of the recovery approach.

Key Words: multi - thread deadlock adjacency matrix banker algorithm

1 死锁

1.1 死锁的定义

在多道程序系统中，虽可借助于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中因争夺资源而造成的一种僵局(Deadly_Embrace),当进程处于这种僵持状态时，若无外力作用，它们都将无法再向前推进。一组进程中，每个进程都无限等待被该组进程中另一进程所占有的资源，因而永远无法得到的资源，这种现象称为进程死锁，这一组进程就称为死锁进程。

1.2 死锁产生的原因

产生死锁的原因主要是：

（1） 因为系统资源不足。

（2） 进程运行推进的顺序不合适。

（3） 资源分配不当等。

如果系统资源充足，进程的资源请求都能够得到满足，死锁出现的可能性就很低，否则就会因争夺有限的资源而陷入死锁。其次，进程运行推进顺序与速度不同，也可能产生死锁。

1.3 产生死锁的四个必要条件

（1） 互斥条件：一个资源每次只能被一个进程使用。

（2） 请求与保持条件：一个进程因请求资源而阻塞时，对已获得的资源保持不放。

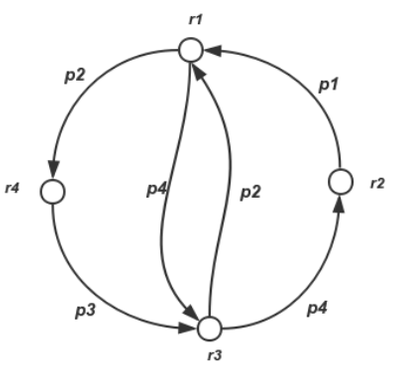
（3） 不剥夺条件:进程已获得的资源，在末使用完之前，不能强行剥夺。

（4） 循环等待条件:若干进程之间形成一种头尾相接的循环等待资源关系。

2 死锁的检测

2.1 资源请求的有向图表示：

将 t 时刻所需资源 Rn={r1,r2,r3…} 看做节点，t 时刻运行的程序集合 P = {p1,p2,p3…} 看做有向边，从程序占有的资源，指向请求的资源，绘出有向图，例如：



2.2 邻接矩阵检测计算机死锁

将 t 时刻资源和程序的有向图用邻接矩阵表示：

$$A=\begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{pmatrix}$$

其中:

$$a_{ij}=0 \quad <ri,rj>\notin P$$
$$a_{ij}=1 \quad <ri,rj>\in P$$

求出该邻接矩阵的可达性矩阵：

$$P=A\vee A^2\vee A^3\vee\cdots\vee A^n$$

若 P 内元素均为 1，即：

$$R(P)=n$$

时，程序陷入死锁，各资源相互请求，无法推进。

3 死锁的避免

不需事先采取各种限制措施去破坏产生死锁的必要条件，而是在资源的动态分配过程中，用某种方法去防止系统进入不安全状态，从而避免发生死锁。 银行家算法的基本思想是分配资源之前,判断系统是否是安全的；若是，才分配。它是最具有代表性的避免死锁的算法。

3.1 银行家算法

银行家算法是最著名的死锁避免算法。它提出的思想是：把操作系统看做是银行家，操作系统管理的资源相当于银行家管理的资金，进程向操作系统请求分配资源相当于用户向银行家贷款。操作系统按照银行家制定的规则为进程分配资源，当进程首次申请资源时，要测试该进程对资源的最大需求量，如果系统现存的资源可以满足它的最大需求量则按当前的申请量分配资源，否则就推迟分配。当进程在执行中继续申请资源时，先测试该进程已占用的资源数与本次申请的资源数之和是否超过了该进程对资源的最大需求量。若超过则拒绝分配资源，若没有超过则再测试系统现存的资源能否满足该进程尚需的最大资源量，若能满足则按当前的申请量分配资源，否则也要推迟分配。

3.2 算法的数据结构描述

可利用资源矢量 Available：含有 m 个元素的歎组，其中的每一个元素代表一类可用的资源数目。Available[j]=K，则表示系统中现有 Rj 类资源 K 个。

最大需求矩阵 Max：为 n*m 矩阵，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。Max[i, j]=K，则表示进程 i 需要 Rj 类资源的最大数目为 K。

分配矩阵 Allocation：为 n*m 矩阵，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。All0Cati0n[i, j]= K，则表示进程 i 当前已分得 Rj 类资源的数目为 K。

需求矩阵 Need：为 n*m 矩阵，表示每个进程尚需的各类资源数。Need[i, j]=K，则表示进程 i 还需要 Rj 类资源的数目为 K。

上述三个矩阵间存在下述关系：
Need[i, j] = Max[i, j] - Allocation[i, j]

3.3 算法描述

设 Requesti 是进程 Pi 的请求矢量，如果 Requesti[j]K，表示进程 Pi 需要 Rj 类资源 K 个。当 Pi 发出资源请求后，系统按下述步骤进行检查：

①如果 Requesti[j] <= Need[i, j]，便转向步骤②；否则认为出错，因为它所需要的资源数已超过它所宣布的最大值。

②如果 Requesti[j] <= Available[j]，便转向步骤③;否则，表示尚无足够资源，Pi 须等待。

③系统试探着把资源分配给进程 Pi，并修改下面数据结构中的数值：

[cpp] view plain copy

Available[j] = Available[j] - Requesti[j];

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

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

④系统执行安全性算法，检查此次资源分配后，系统是否处于安全状态。若安全，才正式将资源分配给进程 Pi，以完成本次分配；否则，将本次的试探分配作废，恢复原来的资源分配状态，让进程 Pi 等待。

3.4 程序实验

需要建立相应的数组

```
int allocation[M][M];
int request[M][M];
int available[M];
int line[M];   //管理不占用资源的进程
int no[M];   //记录造成死锁的进程 i
nt work[M];
```

4 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。

附录：

程序源码：

```
#include "stdio.h"
#define M 50
int allocation[M][M];
int request[M][M];
int available[M];
int line[M];
int no[M];
int n, m, i, j, f, a = 0;

int main() {
    void check();
    void remove();
```

```
    void show();
    printf("输入进程总数:");
    scanf("%d", &n);
    printf("输入资源种类数量:");
    scanf("%d", &m);
    printf("输入进程已占用的资源Allocation:\n");
    for (i = 0; i < n; i++)
        for (j = 0; j < m; j++)
            scanf("%d", &allocation[i][j]);
    printf("输入进程的请求矩阵request:\n");
    for (i = 0; i < n; i++)
        for (j = 0; j < m; j++)
            scanf("%d", &request[i][j]);
    printf("输入系统可利用资源available:\n");
    for (j = 0; j < m; j++)
        scanf("%d", &available[j]);
    show();
    check();
    f = 1;
    for (i = 0; i < n; i++) {
        if (line[i] == 0) {
            f = 0;
            no[a++] = i;//记录死锁序号
        }
    }
    if (f == 0) {
        printf("该系统将发生死锁！\n");
        printf("造成死锁的进程为：");
        for (i = 0; i < n; i++)
            printf("%2d", no[i]);
        printf("\n");
        remove();
        show();
    } else {
        printf("不会发生死锁！\n");
    }

}

void check()//死锁检测
{
    int k, ;
    int x;
    int work[M];
    for (i = 0; i < n; i++)
        line[i] = 0;
    for (i = 0; i < n; i++)  //(2)
    {
        x = 0;
        for (j = 0; j < m; j++) {
            if (allocation[i][j] == 0)
                x++;
            if (x == m)
                line[i] = 1;
        }
    }
    for (j = 0; j < m; j++)//(3)
        work[j] = available[j];
    k = n;
    do {
        for (i = 0; i < n; i++) {
            if (line[i] == 0) {
                f = 1; //空置条件是否满足
                for (j = 0; j < m; j++)
                    if (request[i][j] > work[j])
                        f = 0;
                if (f == 1)    //找到满足条件的进程
                {
                    line[i] = 1;
                    for (j = 0; j < m; j++)
                        work[j] = work[j] + allocation[i][j];
//释放资源
                    available[j] = work[j];
                }
            }
        }
        k--;
    } while (k > 0);
}

void remove() //死锁解除
{
    for (i = 0; i < n; i++) {
        if (line[i] == 0) {
            for (j = 0; j < m; j++) {
                available[j] += allocation[i][j];
                allocation[i][j] = 0;
```

```
                request[i][j] = 0;
            }
        }
    }
    printf("死锁解除！\n");
}

void show() {
    printf("进程");
    printf("    ");
    printf("allocation");
    printf("    ");
    printf("request");
    printf("    ");
    printf("available");
    printf("\n");
    for (i = 0; i < n; i++) {
        printf("%2d", i);
        printf("          ");
        for (j = 0; j < m; j++)
            printf("%2d", allocation[i][j]);
        printf("         ");
        for (j = 0; j < m; j++)
            printf("%2d", request[i][j]);
        printf("       ");
        for (j = 0; j < m; j++) {
            if (i > 0)
                break;
            printf("%2d", available[j]);
        }
        printf("\n");
    }
}
```

参考文献：

[1] 吴企渊. 计算机操作系统［M］. 清华大学，2006.

[2] 刘三阳等. 线性代数［M］.-2 版. 北京：高等教育出版社，2009.7：12.

[3] 方世昌. 离散数学［M］.-3 版. 西安：西安电子科技大学出版社，2009.1：267

[4] 甄志龙 于远诚 OS 中死锁问题的状态模型探讨 [J]通化师范学院学报 2005.