操作系统第一次作业

1. **死锁与饥饿的定义**

**1、死锁：**如果一组进程中的每一个进程都在等待仅由该组进程中的其它进程才能引发的事件，那么该组进程是死锁的（Deadlock）。

（来源：计算机操作系统第四版）

**2、饥饿：**与[死锁](https://baike.baidu.com/item/%E6%AD%BB%E9%94%81)和非常相似。是指一个可运行的进程尽管能继续执行，但被调度器无限期地忽视，而不能被调度执行的情况。

（来源：百度百科）

**通俗的解释死锁与饥饿：**

**1、死锁：**多方抢占资源，每方只获得了一点但都不够，而且都不愿放弃自己手中的资源，造成每方的工作都无法完成。

举例：两个人过独木桥，两个人都走到了桥中间，一个人在等另一个人过去了再过桥，另一个在等这一个过去了在过桥。结果两个人都堵在中间动弹不得。

**2、饥饿：**一个进程在无限期的等待其他进程永远也不会释放给它的资源。

举例：三个人在传球，其中两个人传来传去，第三个人始终得不到球。

1. **有三类资源A(17)、B(5)、C(20)。有五个进程p1->p5。T0时刻系统状态如下:**

|  |  |  |
| --- | --- | --- |
|  | 最大需求 | 已分配 |
| P1 | **5 5 9** | **2 1 2** |
| P2 | **5 3 6** | **4 0 2** |
| P3 | **4 0 11** | **4 0 5** |
| P4 | **4 2 5** | **2 0 4** |
| P5 | **4 2 4** | 1. **1 4** |

**（1）T0时刻是否为安全状态，给出安全系列。**

**（2）T0时刻，P2:Request(0,3,4),能否分配，为什么？**

**（3）在（2）的基础上P4:Request(2,0,1),能否分配，为什么？**

**（4）在（3）的基础上P1:Request(0,2,0),能否分配，为什么？**

解：

**（1）**T0时刻的安全性：利用安全性算法对T0时刻的资源分配情况进行分析（如下图）可知，在T0时刻存在着一个安全序列{P3,P2,P4,P5,P1},故系统是安全的。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 资源  情况  进程 | Work | Need | Allocation | Work+Allocation | Finish |
| A B C | A B C | A B C | A B C |
| P3 | 2 3 3 | 0 6 6 | 4 0 5 | 6 3 8 | true |
| P2 | 6 3 8 | 1 3 4 | 4 0 2 | 10 3 10 | true |
| P4 | 10 3 10 | 2 2 1 | 2 0 4 | 12 3 14 | true |
| P5 | 12 3 14 | 1 1 0 | 3 1 4 | 15 4 18 | true |
| P1 | 15 4 18 | 3 4 7 | 2 1 2 | 17 5 20 | true |

**（2）**T0时刻，P2发出请求向量Request2(0,3,4)，系统按银行家算法进行检查：

Request2(0,3,4)<=Need2(1,3,4);

Request2(0,3,4)>Available(2,3,3),让P2等待。

故不可分配。

**（3）**在（2）的基础上P4发出请求向量Request4(2,0,1),系统按银行家算法检查：

Request4(2,0,1)<=Need4(2,2,1);

Request4(2,0,1)<=Available(2,3,3);

系统先假定可为P4分配资源，并修改Available，Allocation4和Need4向量，由此形成的资源变化情况如图：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 资源  情况  进程 | Max | Allocation | Need | Available |
| A B C | A B C | A B C | A B C |
| P1 | 5 5 9 | 2 1 2 | 3 4 7 | 2 3 3  (0 3 2) |
| P2 | 5 3 6 | 4 0 2 | 1 3 4 |
| P3 | 4 0 11 | 4 0 5 | 0 0 6 |
| P4 | 4 2 5 | 2 0 4  (4 0 5) | 2 2 1  (0 2 0) |
| P5 | 4 2 4 | 3 1 4 | 1 1 0 |

再利用安全性算法检查此时系统是否安全，如图。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 资源  情况  进程 | Work | Need | Allocation | Work+Allocation | Finish |
| A B C | A B C | A B C | A B C |
| P4 | 0 3 2 | 0 2 0 | 4 0 5 | 4 3 7 | true |
| P2 | 4 3 7 | 1 3 4 | 4 0 2 | 8 3 9 | true |
| P3 | 8 3 9 | 0 0 6 | 4 0 5 | 12 3 14 | true |
| P5 | 12 3 14 | 1 1 0 | 3 1 4 | 15 4 18 | true |
| P1 | 15 4 18 | 3 4 7 | 2 1 2 | 17 5 20 | true |

由所进行的安全性检查得知，可以找到一个安全序列{P4,P2,P3,P5,P1}。因此，系统是安全的，可以理解将P4所申请的资源分配给它。（即本次P4申请后，下一次仍由P4申请（0,2,0）即可保证系统安全）

**（4）**在（3）的基础上P1发出请求向量Request1(0,2,0),系统按银行家算法检查：

Request1(0,2,0)<=Need1(3,4,7);

Request1(0,2,0)<=Available(0,3,2);

系统先假定可为P1分配资源，并修改Available，Allocation1和Need1向量，由此形成的资源变化情况如图：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 资源  情况  进程 | Max | Allocation | Need | Available |
| A B C | A B C | A B C | A B C |
| P1 | 5 5 9 | 2 1 2  (2 3 2) | 3 4 7  (3 2 7) | 0 3 2  (0 1 2) |
| P2 | 5 3 6 | 4 0 2 | 1 3 4 |
| P3 | 4 0 11 | 4 0 5 | 0 0 6 |
| P4 | 4 2 5 | 4 0 5 | 0 2 0 |
| P5 | 4 2 4 | 3 1 4 | 1 1 0 |

进入安全性检查：可用资源Available（0,1,2）已不能满足任何进程的需求，故系统进入不安全状态，此时系统不分配资源。