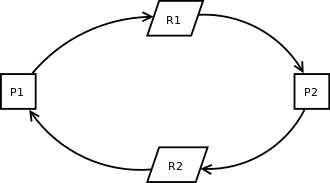
**OS第一次作业**

罗沛林 中美计算机 2016011979

1. **找出死锁、饥饿的定义并注明出自哪本参考书，然后通过自己的理解用自然语言通俗描述**。

解：**1.死锁定义：**死锁（英语：Deadlock），又译为死结，计算机科学名词。当两个以上的运算单元，双方都在等待对方停止运行，以获取系统资源，但是没有一方提前退出时，就称为死锁。在多任务操作系统中，操作系统为了协调不同进程，能否获取系统资源时，为了让系统运作，必须要解决这个问题。

这里指的是[进程](https://zh.wikipedia.org/wiki/%E8%BF%9B%E7%A8%8B" \o "进程)死锁，是个计算机技术名词。它是[操作系统](https://zh.wikipedia.org/wiki/%E6%93%8D%E4%BD%9C%E7%B3%BB%E7%BB%9F" \o "操作系统)或软件运行的一种状态：在多任务系统下，当一个或多个进程等待系统资源，而资源又被进程本身或其他进程占用时，就形成了死锁。有个变种叫[活锁](https://zh.wikipedia.org/w/index.php?title=%E6%B4%BB%E9%94%81&action=edit&redlink=1" \o "活锁（页面不存在）)。（来自维基百科）



P1、P2两个process都需要资源才能继续运行。P1拥有资源R2、还需要额外资源R1才能运行；P2拥有资源R1、还需要额外资源R2才能运行，两边都在互相等待而没有任何一个可执行。

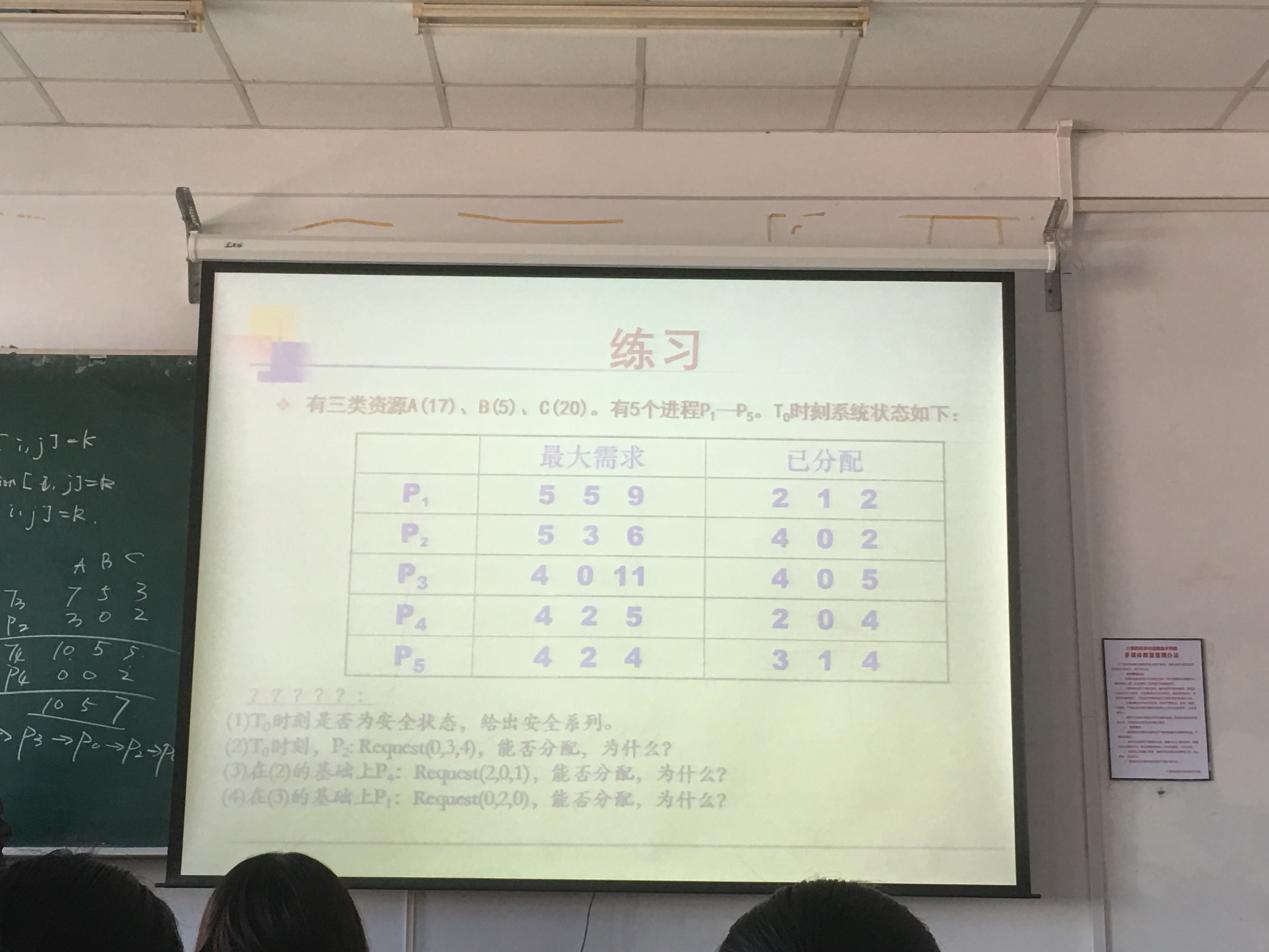
**2.饥饿定义：**进程饥饿，即为Starvation，指当等待时间给进程推进和响应带来明显影响称为进程饥饿。当饥饿到一定程度的进程在等待到即使完成也无实际意义的时候称为饥饿死亡。

亦有定义为：由于别的并发的激活的过程持久占有所需资源,使某个异步过程载客预测的时间内不能被激活。（来自百度百科）

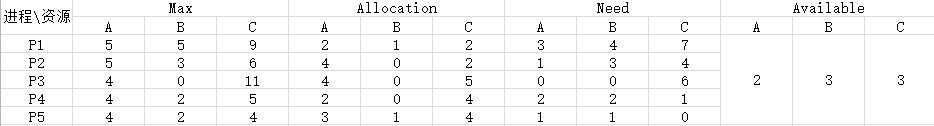
**3.死锁的通俗解释：**两个人吃牛排，刀和叉，我拿到了刀，要叉，你拿到了叉要刀，结果两个人都吃不到牛排。因为我不放下刀你拿不到叉，你不放下叉拿不到刀。

**4.饥饿的通俗解释：**当有多个进程需要打印文件时，如果系统分配打印机的策略是最短文件优先，那么长文件的打印任务将由于短文件的源源不断到来而被无限期推迟，导致最终的饥饿甚至饿死。

1. **解答此题。**



解：（1）T0时刻的状态：

T0时刻是安全状态，因为可以找到一个安全的序列。（P4，P5，P1，P2，P3）。

**分析：**T0时刻，可使用的A,B,C类资源各位2,3,3,若先将资源分配给P4以满足P4的需求，则进程P4完成后，将分配的资源释放出来，可用的资源变成4,3,7；

然后再把可用资源分配给P2以满足P2的需求，P2完成后，将分配给P2的资源释放出来，可用资源变成8,3,9；

接着再把可用资源分配给P3以满足P3的需求，P3完成后，将分配给P3的资源释放，可用资源变成12,3,14；

再接着把可用资源分配给P5以满足P5的需求，P5完成后，将分配给P5的资源释放，可用资源变成15,4,8；

最后将资源分配给P1以满足P1的需求，P1完成后资源释放。

最后将P1-P5全部完成，没有产生死锁状态，故T0时刻为安全状态，其安全序列为P4->P2->P3->P5->P1。

**（2）不能分配。**因为所剩余的资源数量不够。

T0时刻，P2所需A,B,C类资源分别为1,3,4，若P2提出0,3,4的请求，请求的资源小于他所需要的资源，第一个步骤可以满足；但是T0时刻可使用的资源为2,3,3.即P2请求的资源数大于可以分配的资源数，不能分配。因为所剩余的资源数量不够，所以尚无足够资源，P2须等待。

**（3）可以分配。**P4请求资源：P4发出请求向量Request(2,0,1)，系统按银行家算法进行检查：

①Request(2,0,1)≤Need(2,2,1)；

②Request(2,0,1)≤Available(2,3,3)；

③系统先假定可为P4分配资源，并修改Available，Allocation和Need向量，由此形成的资源变化情况如下表括号中所示：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 资源  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 |  |

④再利用安全性算法检查此时系统是否安全，如下表所示：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 资源  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，可以立即将P4所申请的资源分配给它。

当分配完成后，系统剩余的资源数量为（0,3,2），这时仍可找到一个安全的序列队（P4,P5,P1,P2,P3）。T0时刻，P4所需A,B,C类资源分别为2,2,1，若P2提出2,0,1的请求，请求的资源小于他所需要的资源，第一个步骤可以满足；T0时刻可使用的资源为2,3,3，

可使用资源数大于请求数，故系统可以试探着把资源分配给P4。

**（4）不能分配。**P1请求资源：在（3）的基础上，P4已顺利获取所申请的资源，由此形成的资源变化情况如下表所示：

|  |  |  |  |
| --- | --- | --- | --- |
| Max | Allocation | Need | Available |
| A B C | A B C | A B C | A B C |
| 5 5 9 | 2 1 2 | 3 4 7 | 0 3 2 |
| 5 3 6 | 4 0 2 | 1 3 4 |  |
| 4 0 11 | 4 0 5 | 0 0 6 |  |
| 4 2 5 | 4 0 5 | 0 2 0 |  |
| 4 2 4 | 3 1 4 | 1 1 0 |  |

在（2）的基础上，考虑P2发出的请求向量Request(0,3,4)，系统按银行家算法进行检查：

①Request(0,3,4)≤Need(1,3,4)；

②Request(0,3,4)＞Available(0,3,2)，让P2继续等待；

继而考虑P1发出请求的向量Request(0,2,0)，系统按银行家算法进行检查：

①Request(0,2,0)≤Need(3,4,7)；

②Request(0,2,0)≤Available(0,3,2)；

③系统先假定可为P1分配资源，并修改Available，Allocation和Need向量，由此形成的资源变化情况如下表括号中所示：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 资源  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 |  |

④此时可用资源Available(0,1,2)已不能满足任何进程的需要，故系统进入不安全状态，此时系统不分配资源。