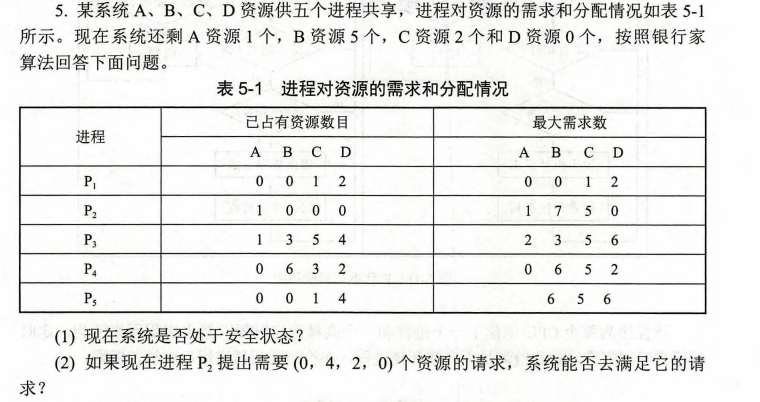
## 操作系统作业5

1. 简述产生死锁的4 个必要条件。举一个例子，虽然这4 个条件都满足，但不产生死锁。

* 互斥 -指进程的共享资源必须保持使用的互斥性
* 占有且等待 -一个进程占有了某些资源之后又要申请新的资源而得不到满足时，处于等待资源的状态，且不释放已经占用的资源
* 不可剥夺 -任何进程不能抢夺另一个进程所占用的资源
* 环路条件 -存在一组进程，其中每个进程分别等待另一个进程所占用的资源，形成环路等待条件
* 在满足上述四个条件的情况下，如果某一类资源有多个资源，就不一定会出现死锁

1. 简述处理死锁的3 种方法的基本思想，简要分析它们各自的利弊。

* 预防死锁
  + 通过破坏四个必要条件之一，可以使系统不具备产生死锁的条件
  + 适用于执行单一突发活动的进程，不需要抢占
  + 效率低，进程初始化时间延长，不便于灵活申请新资源
* 避免死锁
  + 在为申请者分配资源前先测试系统状态，若把资源分配给申请者会产生死锁则拒绝分配，否则接受申请，为它分配资源
  + 不需要抢占，不需要破坏四个条件
  + 必须事先声明每个进程的资源最大需求量；进程之间必须是无关的；系统中可供分配的资源数目必须是固定的；进程在占有资源时，不能退出
* 检测并恢复
  + 任凭死锁的出现。当系统中出现死锁时，就将系统重新启动
  + 不延长进程初始化时间；允许对死锁进行现场处理
  + 通过抢占解除死锁，可能造成损失



该状态下进程所需要的资源数为

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| P1 | 0 | 0 | 0 | 0 |
| P2 | 0 | 7 | 5 | 0 |
| P3 | 1 | 0 | 0 | 2 |
| P4 | 0 | 0 | 2 | 0 |
| P5 | 0 | 6 | 4 | 2 |

可分配资源数为

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Work | 1 | 5 | 2 | 0 |

P1已经完成，释放资源

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Work | 1 | 5 | 3 | 2 |

无法满足P2，可以满足P3，释放P3

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Work | 2 | 8 | 8 | 6 |

可以满足P2，释放P2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Work | 3 | 8 | 8 | 6 |

可以满足P4，释放P4

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Work | 3 | 14 | 11 | 8 |

可以满足P5，释放P5

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Work | 3 | 14 | 12 | 12 |

安全序列为P1、P3、P2、P4、P5，处于安全状态

* 申请条件既没超过需要资源量，也没超过可分配资源量，所以先临时分配

此时，进程所需要的资源数为

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| P1 | 0 | 0 | 0 | 0 |
| P2 | 0 | 3 | 3 | 0 |
| P3 | 1 | 0 | 0 | 2 |
| P4 | 0 | 0 | 2 | 0 |
| P5 | 0 | 6 | 4 | 2 |

可分配资源数为

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Work | 1 | 1 | 0 | 0 |

释放P1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Work | 1 | 1 | 1 | 2 |

释放P3

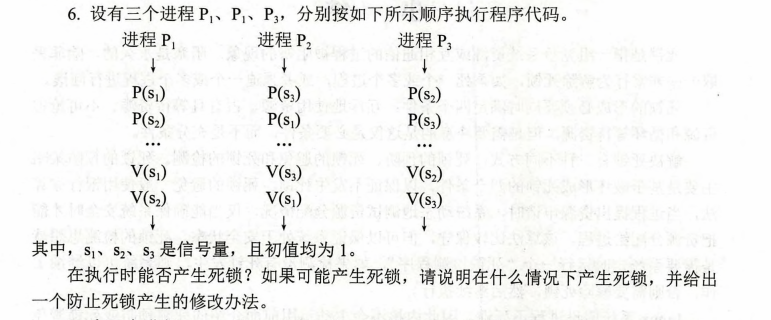
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Work | 2 | 4 | 6 | 6 |

释放P2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Work | 3 | 8 | 8 | 6 |

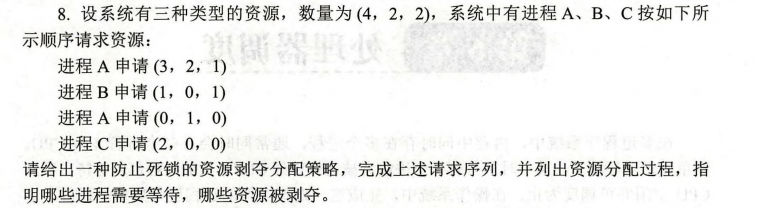
余下情况不变，可以满足请求





* P1执行P(s1)后中断，P2执行P(s3)后中断，P3执行P(s2)，三个资源分别被占用，无法进一步分配资源，导致死锁。
* 和哲学家问题类似，我们可以限制这个程序代码一次只允许两个进程同时执行，这样一定有一个进程能完全执行，释放资源后再供其他进程申请。
* 或者我们可以让每个进程必须按编号小的顺序获取资源，破坏循环等待条件





如果遇到死锁，基于最小代价原则剥夺资源

进程A申请，批准 (1，0，1)

进程B申请，批准 (0，0，0)

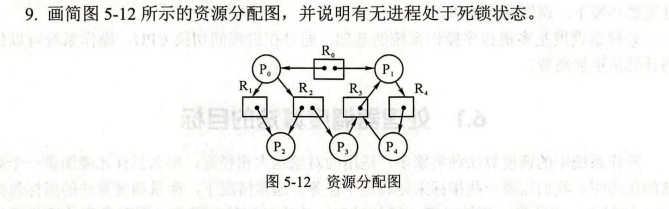
进程A申请，死锁，但因为两个是相同进程，可能需要按照一定顺序执行，不从A中剥夺资源，A等待

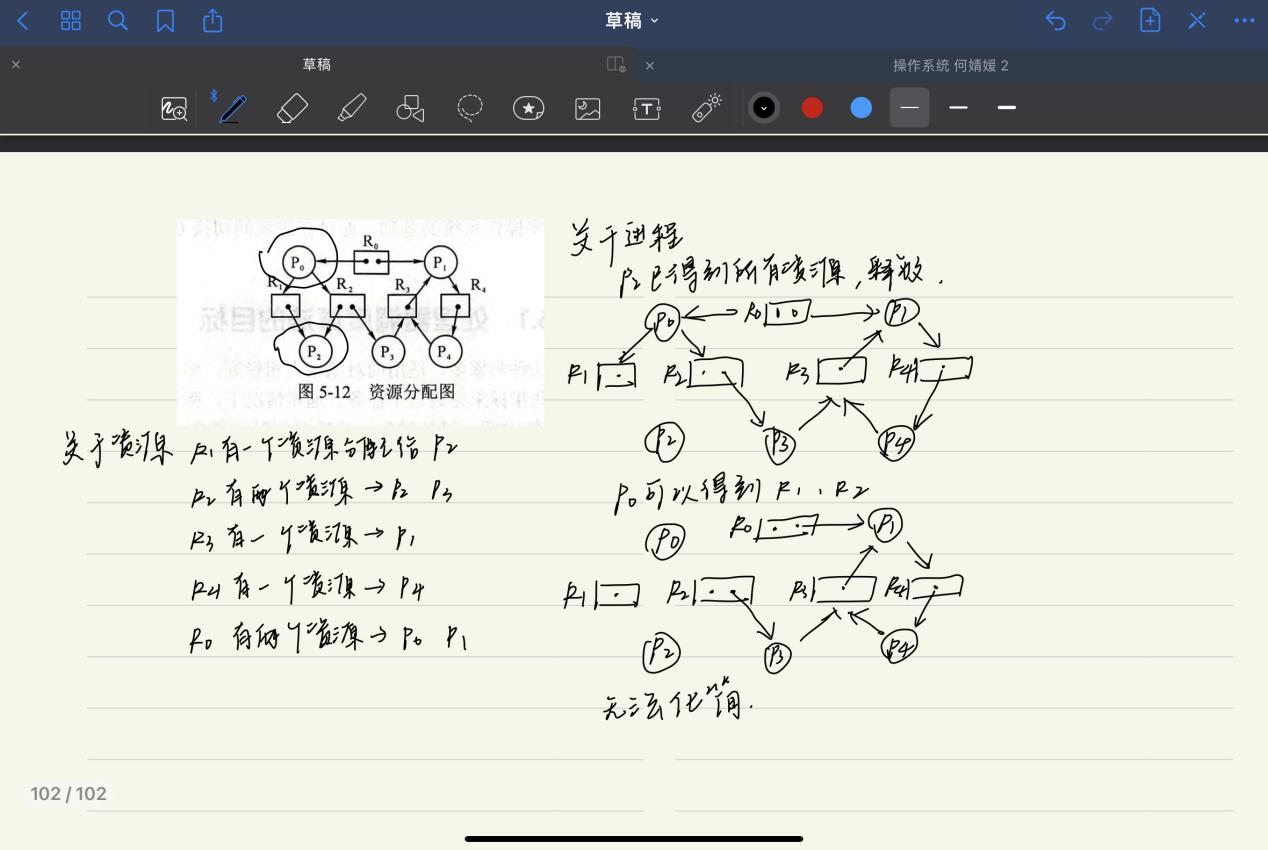
进程C申请，死锁，剥夺进程A的两个资源提供给进程C，A等待

等待进程BC或进程C完成后释放资源

A进程继续







P1P3P4处于死锁状态，就关注P1R3P4R4，在单资源的条件下有环，所以一定会死锁，连带造成了P3的死锁