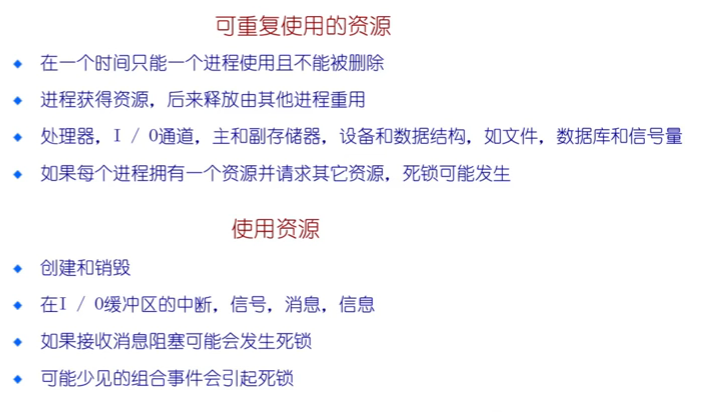
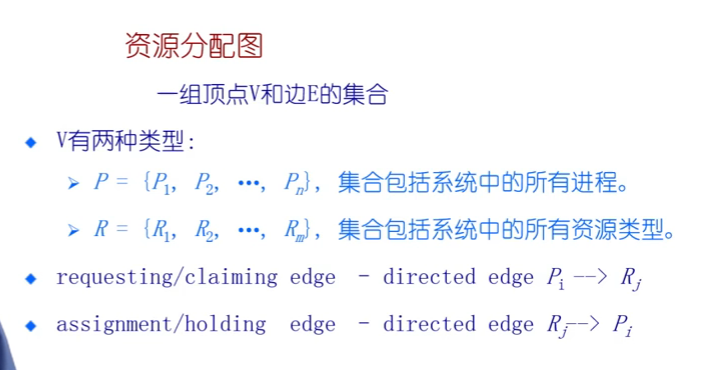
死锁：

一个进程需要一个资源，而需要的资源被另一个进程所占有，而另一个进程也需要该进程占用的一个资源。

出现死锁是由于进程的并发执行导致的。

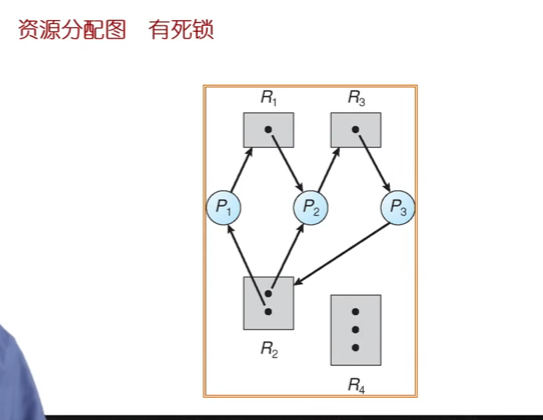
一个进程在使用一些资源时，资源便被占用了，资源的使用具有互斥性。



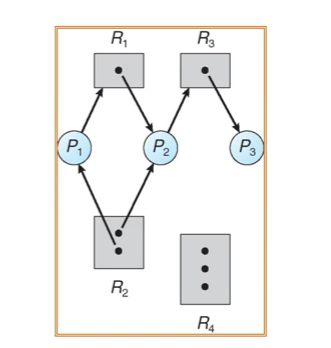


1. >R进程申请访问某个资源
2. >P资源被分配给进程

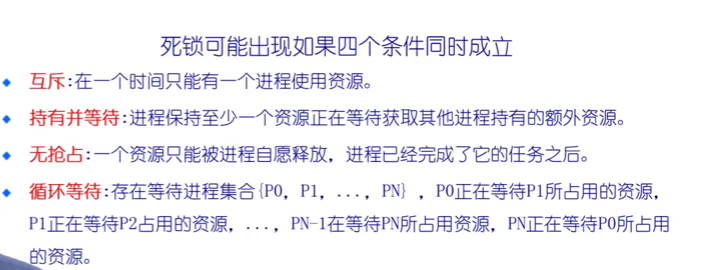
有死锁的图：



没死锁的图：



死锁特征：



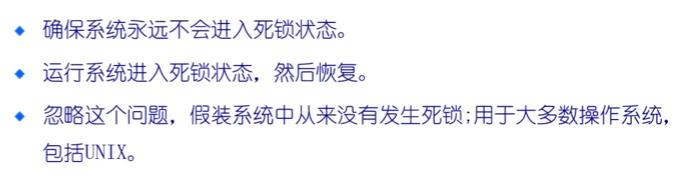
这是死锁的必要条件，但不是充分条件

死锁的处理办法：



其约束条件是从上至下，一次递减，预防最强

思路：



死锁预防：

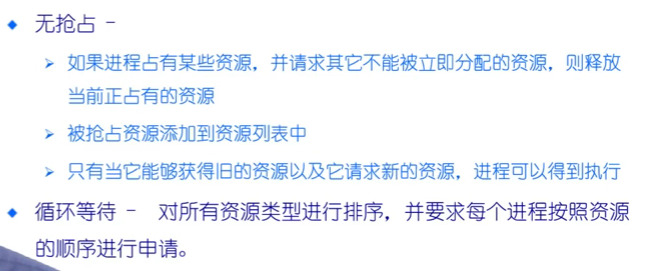
把死锁的四个条件其中一个打破便可以阻止死锁

改互斥不太好

要么拿我需要的全部资源要么就不拿，但是资源的利用率会很低

资源是互斥的，如果要抢占的话，就需要把那个进程杀死掉，比较强势

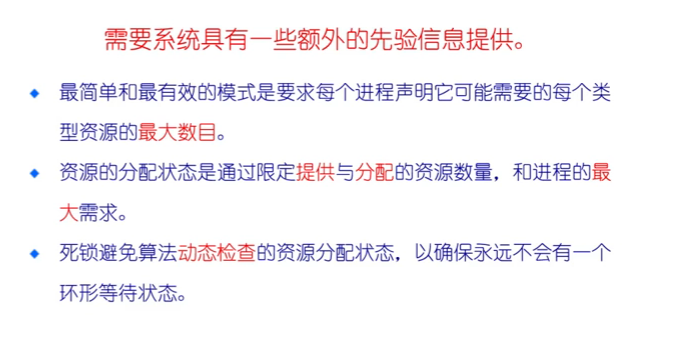
打破循环等待的环，破坏性较小



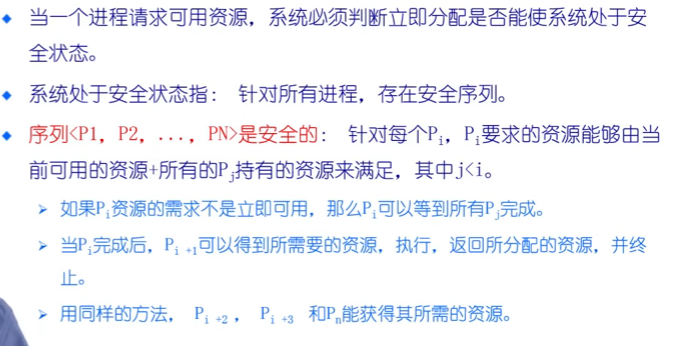
死锁避免：

进程在执行中会申请资源，操作系统对申请的资源进行判断会不会死锁，如果会死锁，则不允许给出该资源。

需要知道一些先验信息：



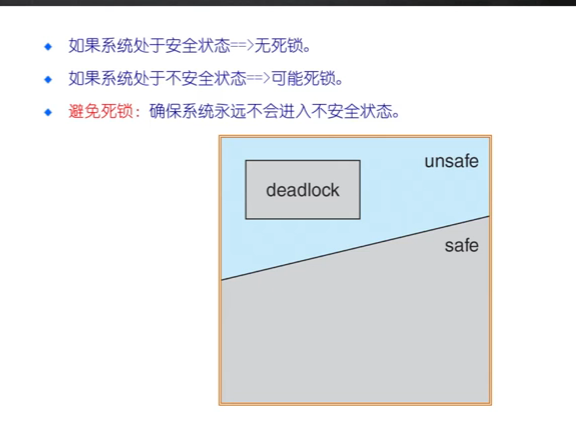
有了环形等待不一定死锁，但是这里条件比较苛刻，环形等待是不安全状态，算法认为你可能会死锁。（约束比较大）



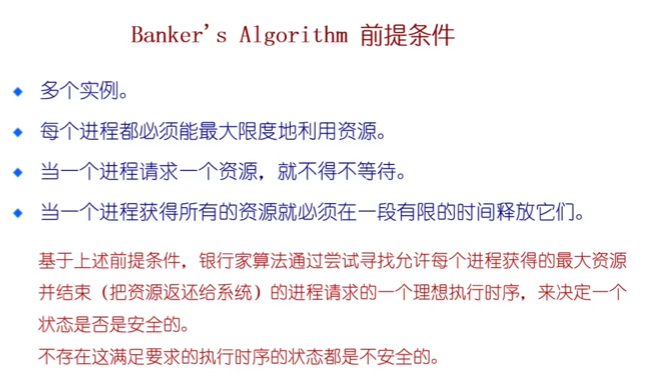
什么是安全状态：

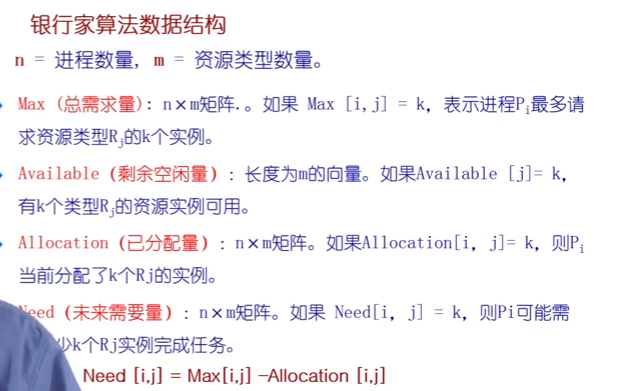
针对所有进程，存在一个时间序列，按照顺序执行进程，每个进程都会得到满足，有限时间内该进程会结束。

判断进程得到满足：Pi运行时，前面的P0-P(i-1)进程都已经结束了，前面进程拥有的资源加当前可用资源加一起均可为Pi所用，此时Pi判断是否可以执行。



银行家算法：（死锁避免算法）

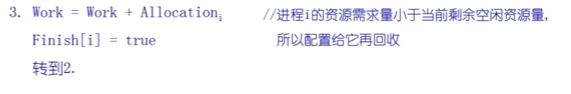




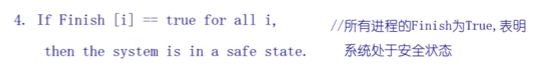


要求的是Need的每一项都小于Work

这样的进程可以正常结束。



进程使用的资源执行完毕，进行回收到work



一直循环直到所有的进程都是true，那这就是一个安全序列

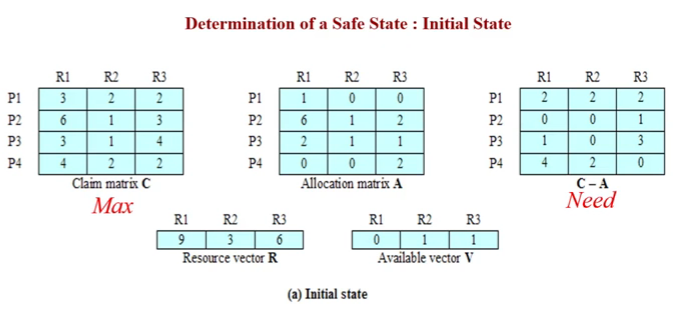
按照该序列执行便全都能结束

如果在2找不到满足的进程就是不安全状态

安全状态检测算法：

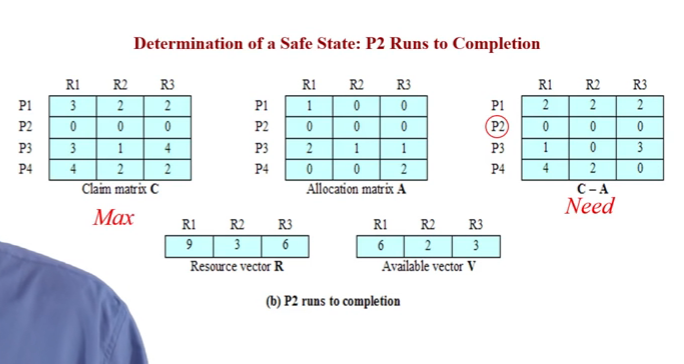


例子：



Allocation是已经分配的，max是总的最大要分配的，need是还需要的，available是还可以分配的。

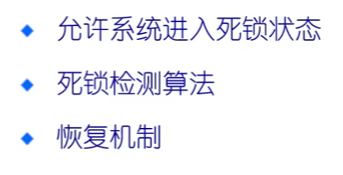
第一个P2可以满足，p2满足后其资源返回给available。

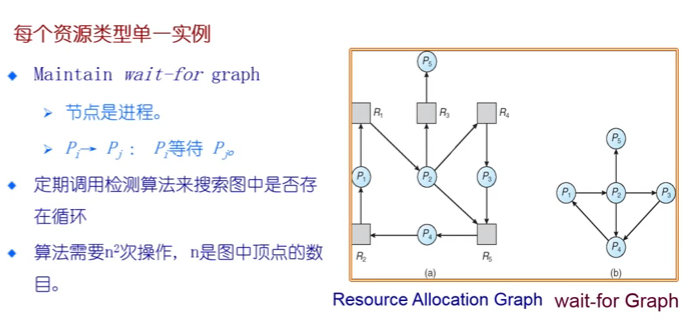


再接着找下一个。

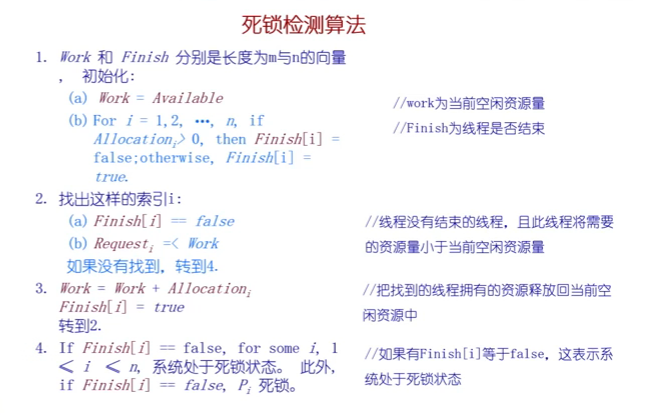
最后能找到安全的一个序列为p2-p1-p3-p4

死锁检测：





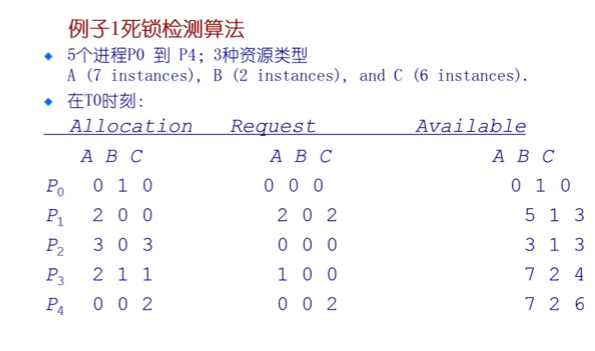
把资源分配图变成等待图，等待图中有环，就可能死锁（我觉得这个单一是指一类资源只有一个可用资源用来分配）

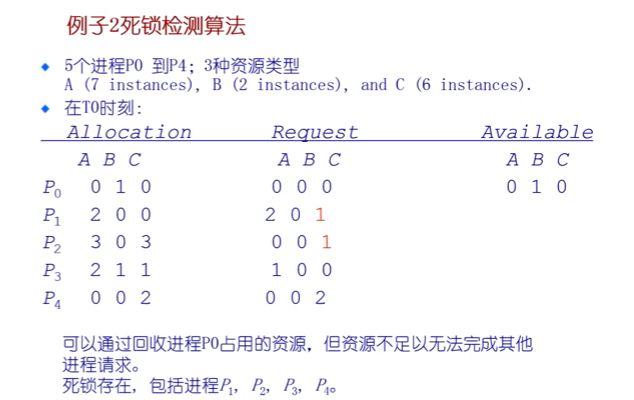


银行家是need，这里是request。

银行家需要提前知道每一个进程需要的最大资源的个数是多少。

死锁检测算法：

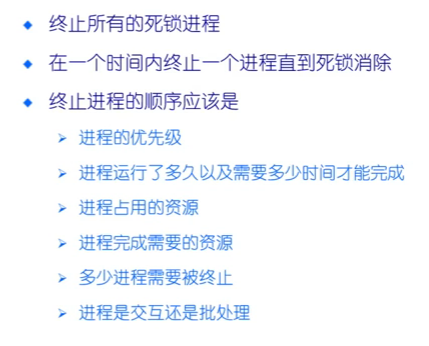




死锁恢复：

一些手段：

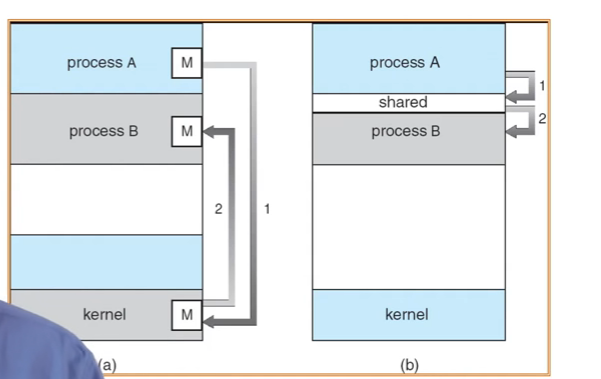
杀死进程：



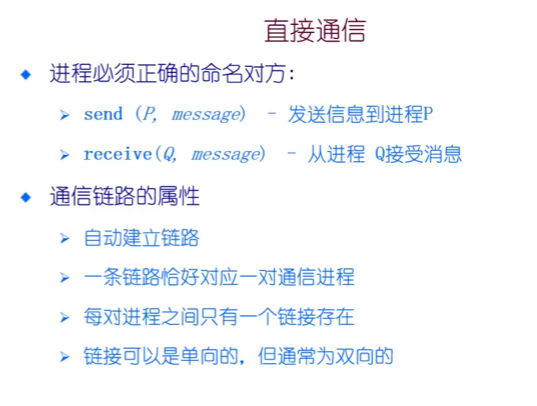
也可以是资源抢占，把某一个进程的资源抢占

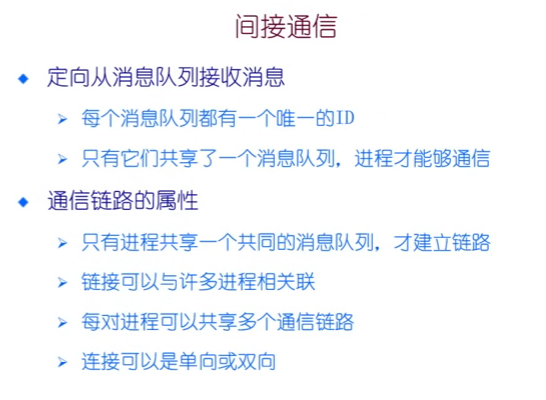
**进程间通信（IPC）：**

进程间要相对保持独立，一个进程不能随便访问其他进程的地址空间



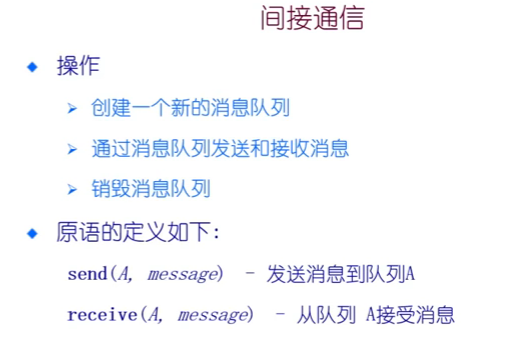
间接通信 直接通信





间接通信会把信息先发送到一个区域，这个区域一般是操作系统提前指定好的区域

接收方也是从该区域接收数据。这个结点是操作系统的一个共享的资源。





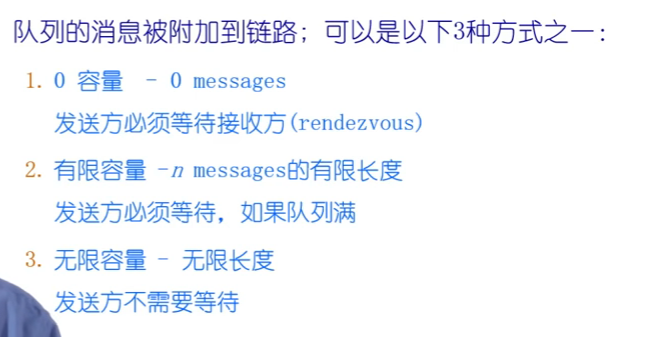
阻塞认为是同步的

而非阻塞是异步的

信号、管道、消息队列和共享内存：

如果我们把消息缓存起来，可以极大的提高效率

缓存的容量：

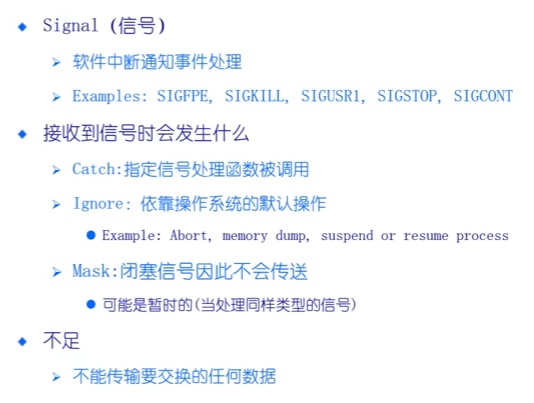


容量为0时，直接发送时发送方必须等到接收方收到之后才能返回，不然消息数据可能丢失

（其类比于阻塞发送方式）

有限容量，如果缓冲区满了，则发送方必须等待

信号：（不是信号量）



不适合用来传递数据，只是用来通知

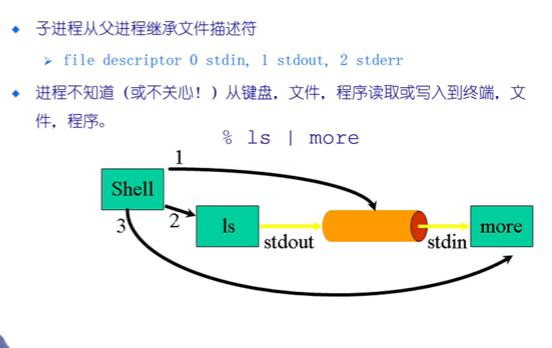
类似于异步打断

一般来说信号处理完之后会回到被打断的函数

为了实现调用程序，操作系统会修改应用程序的堆栈，将堆栈信息的下一跳指令为调用函数的入口地址，将返回修改为原来中断的位置。

管道：

用来数据交换



“|”就是管道，ls的输出到|管道，而|也是more的输入

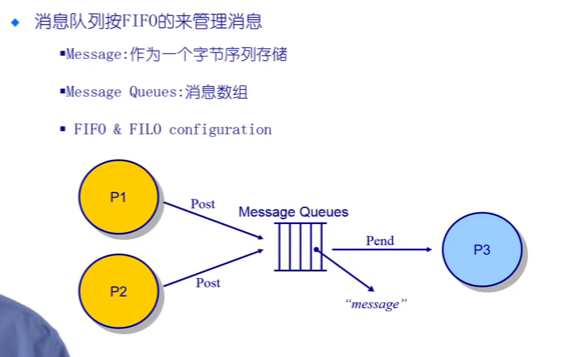
管道的buffer是有限的，消息多的话会阻塞、等待被唤醒

两个可以使用管道是因为他们有一个共同的父进程，共享一些资源

消息队列：

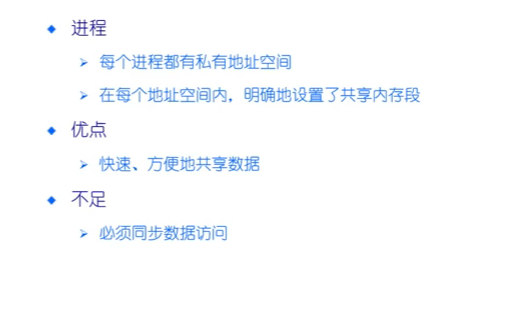
（管道是父进程为子进程建立好的一个通道，如果两个进程没有父子关系，那么管道就无法使用）

消息队列可以实现多个不相干的进程通过消息队列来传递数据



共享内存：

管道、消息队列都是间接通信的方式，而共享内存是直接通信的方式



两个进程有一个特殊的内存空间是都能够访问到的

在通信的时候不需要操作系统介入实现send和recieve，效率很高

但是需要一些同步互斥的机制

如何实现：

把同一块物理地址，映射到不同进程的相同或不同的地址空间中。访问虚拟地址时会访问到同一个物理地址。