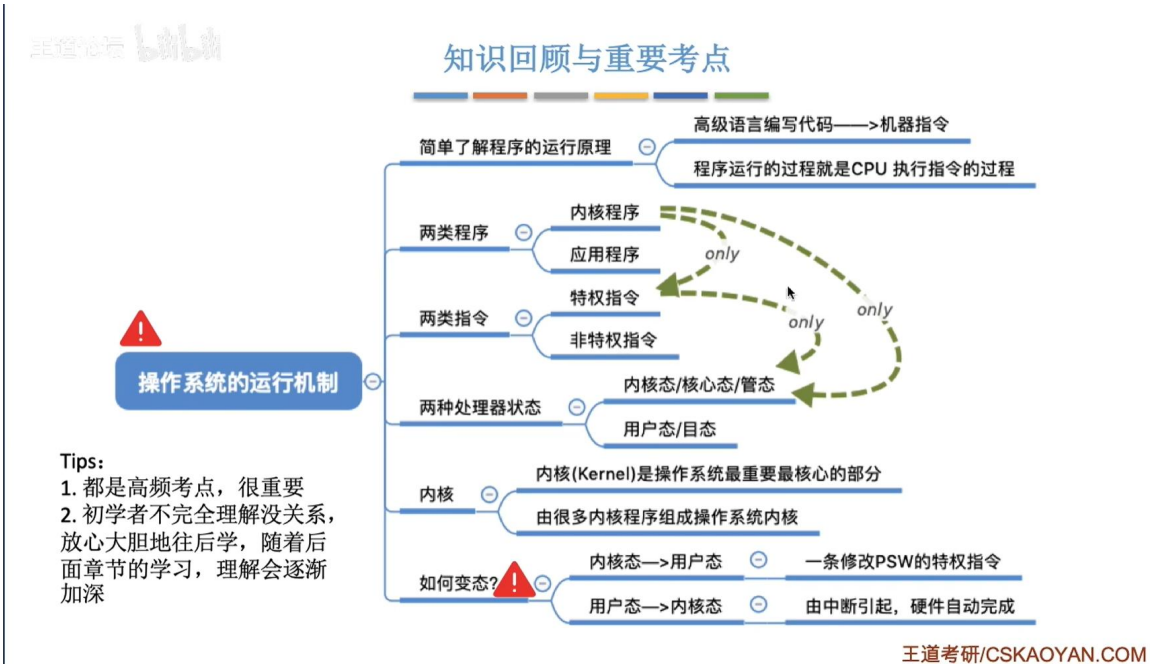
1.3.1操作系统的运行机制

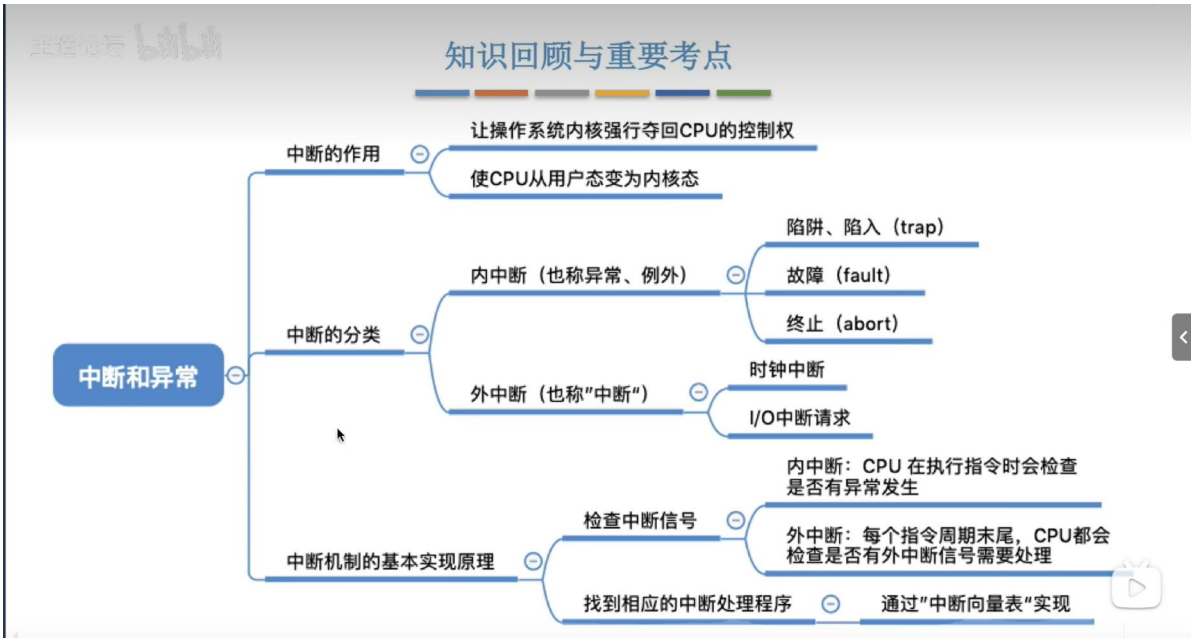


内核态（核心态，管态）和用户态（目态）:cpu的某种状态，内核态可以用特权指令。用户态只能用非特权指令。

psw：程序状态字program status word，存储0/1，表示用户态/内核态

内核态用户态的切换：内核-用户：执行一条特权指令，使psw更改，cpu变成用户态。 用户-内核：引发中断，然后硬件自动完成变态。

1.3.2中断和异常

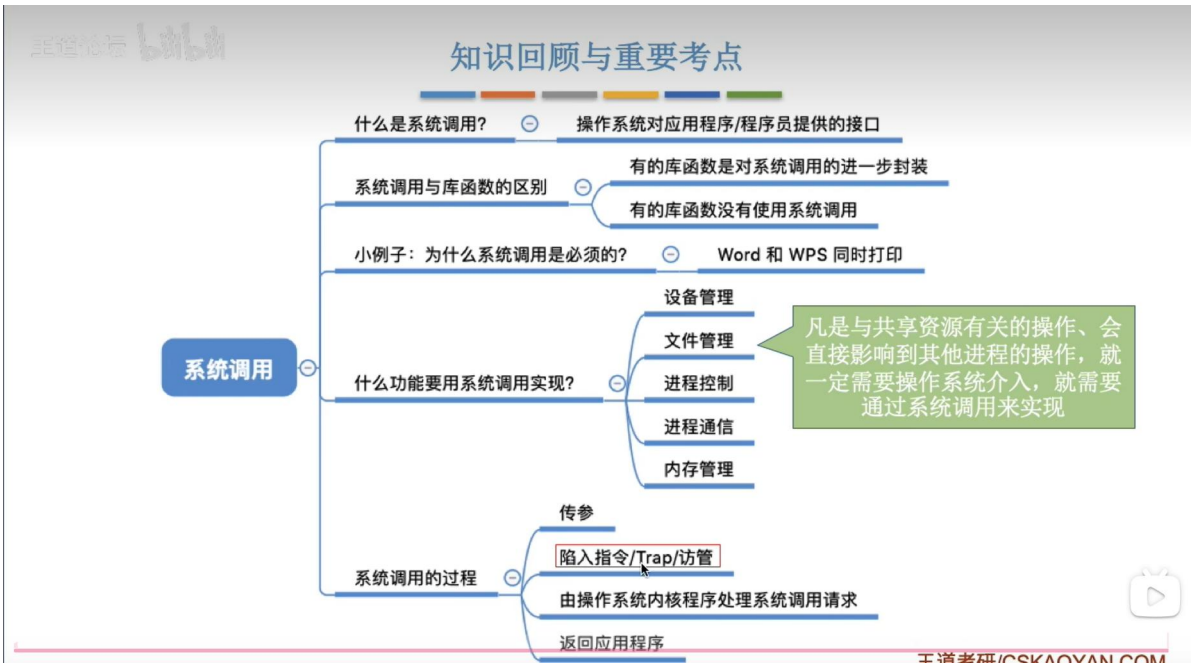


trap：应用程序可以调用trap指令，引发一个内部中断信号。trap是非特权指令。

supervisor call：是一种特殊的指令，用于在用户程序和操作系统内核之间进行交互。当用户程序需要执行一些特权操作，比如访问硬件设备、进行文件I/O、创建新进程等，它可以通过发起SVC来请求操作系统内核的支持

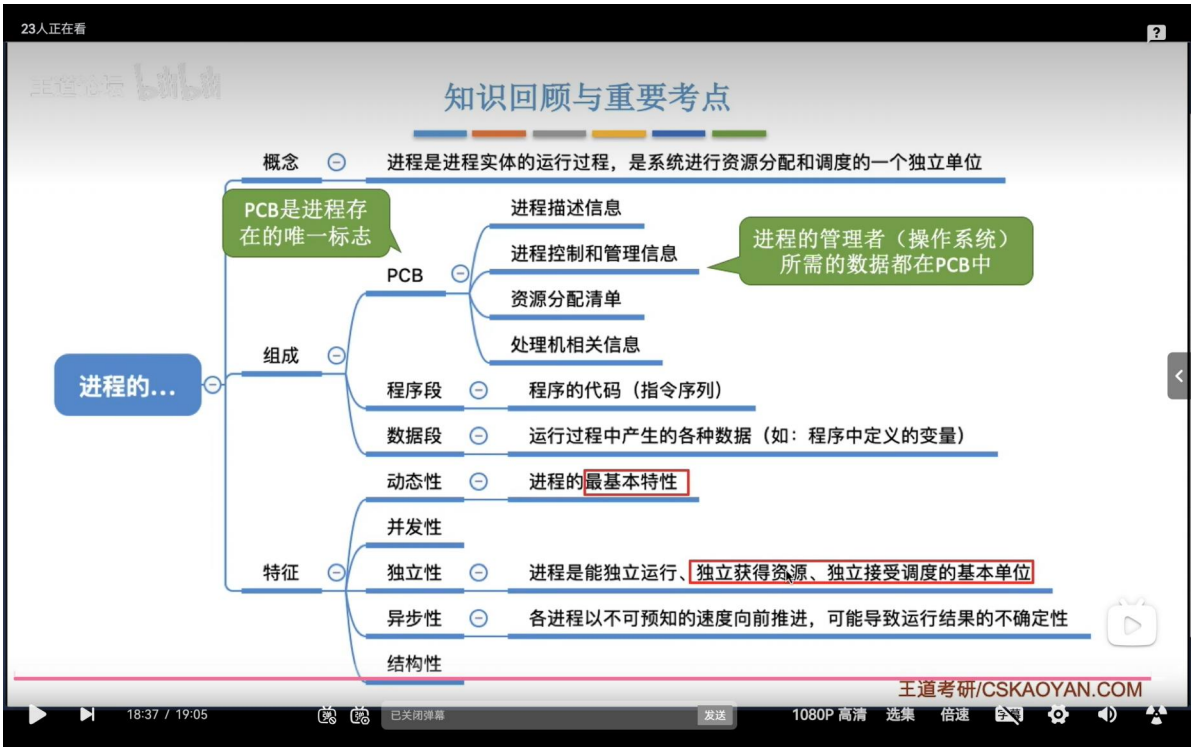
仅有中断可以使操作系统夺回cpu使用权。

中断处理的过程：interrupt handler:
硬件部分：当进程处理器发现中断信号时，先把即将执行的指令的pc和psw存入control stack中，然后处理器加载新的pc值（中断处理程序）。
软件部分：存储进程的剩余部分，进程中断，处理中断，恢复软件存的进程，恢复硬件存的pc和psw。

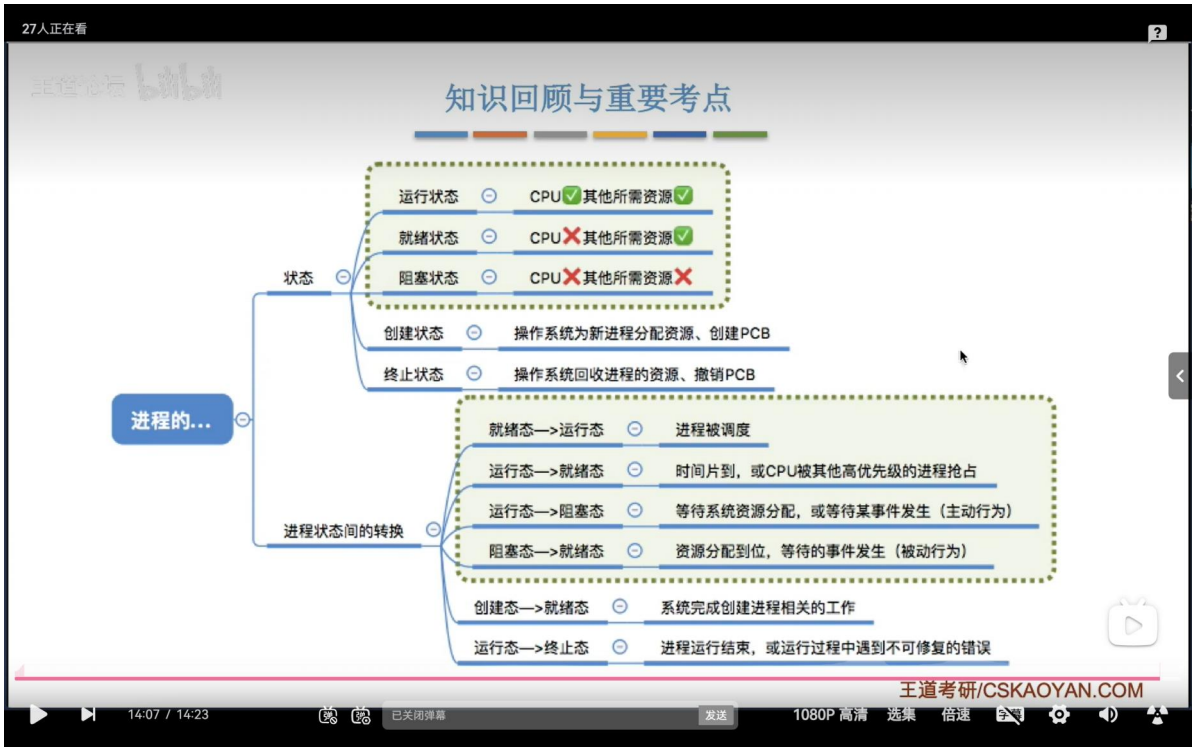


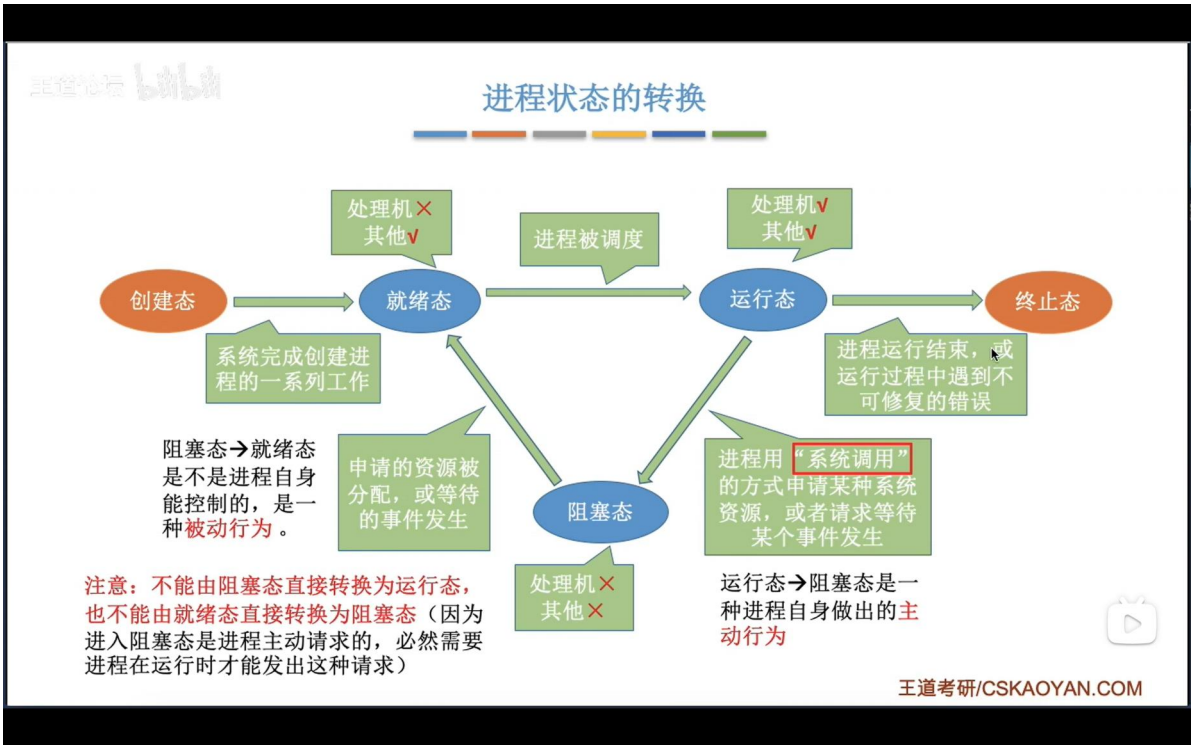
2 进程

进程的概念


23人正在看
知识回顾与重要考点
进程的...
概念
进程是进程实体的运行过程，是系统进行资源分配和调度的一个独立单位
组成
PCB是进程存在的唯一标志
PCB
进程描述信息
进程控制和管理信息
资源分配清单
处理机相关信息
进程的管理者（操作系统）所需的数据都在PCB中
程序段
程序的代码（指令序列）
数据段
运行过程中产生的各种数据（如：程序中定义的变量）
特征
动态性
进程的最基本特性
并发性
独立性
进程是能独立运行、独立获得资源、独立接受调度的基本单位
异步性
各进程以不可预知的速度向前推进，可能导致运行结果的不确定性
结构性
王道考研/CSKAOYAN.COM
18:37 / 19:05
已关闭弹幕
发送
1080P 高清
选集
倍速

进程的状态和转换





进程控制

原语：原语就是实现一个不会被打断的函数。不会被打断是通过两个特权指令：关中断和开中断。先关在开。

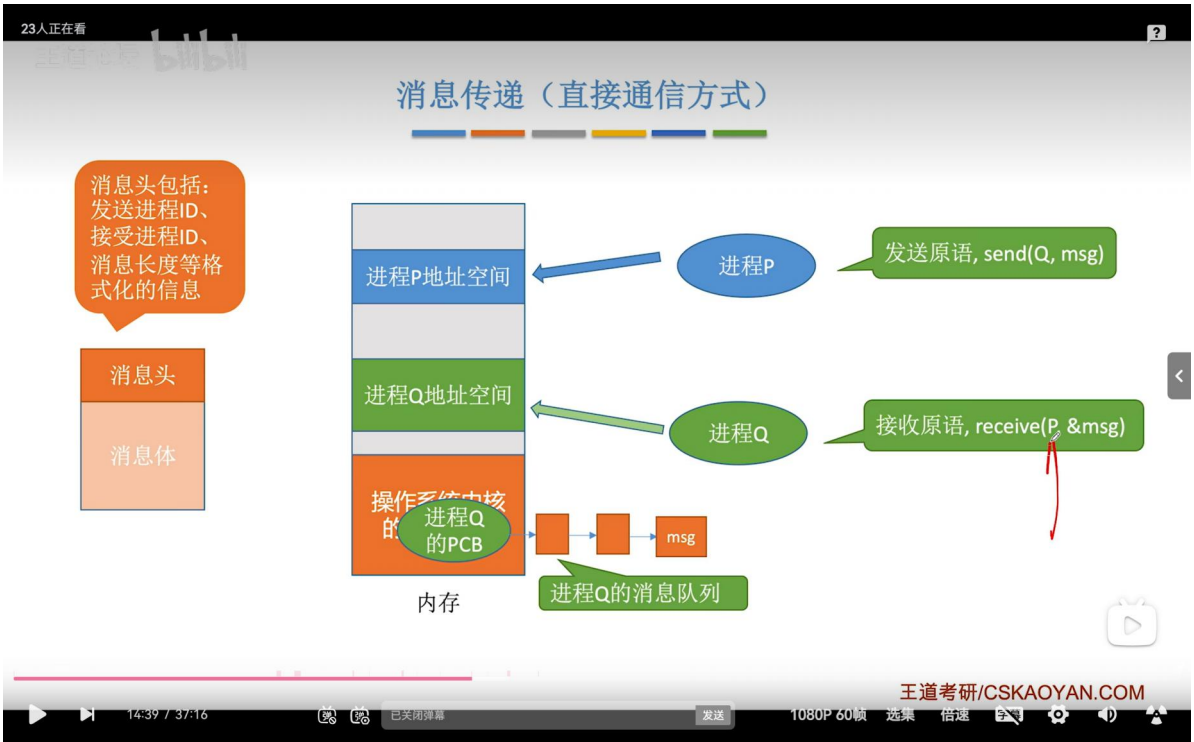
进程通信

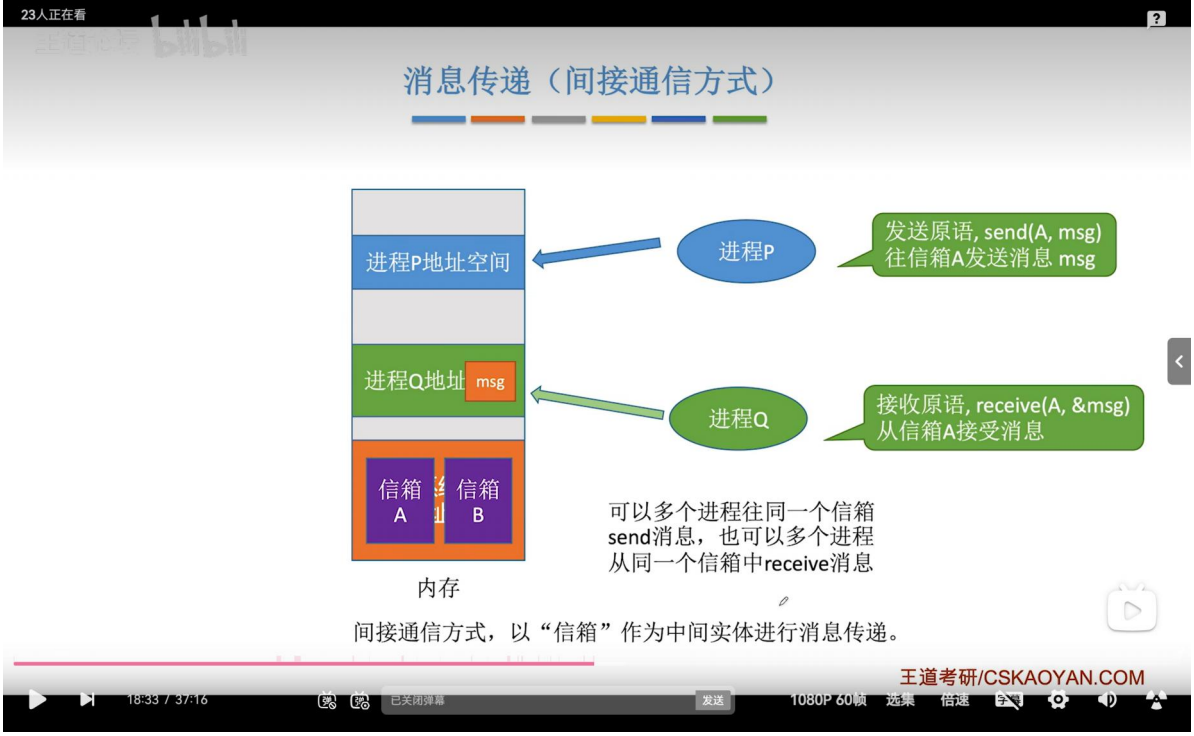
三种方式

1.共享空间，共享空间需要是互斥的访问

2.消息传递：直接传递和间接传递

通过两个原语send & receive实现。





3.管道通信

特点：1.单向 2.FIFO，消息传递和共享空间不是FIFO的 3.单独一片内存区域 4.内存写满时，写进程将阻塞，管道空时，读进程阻塞。

3线程

线程的概念

一个进程可能有多个线程。线程是执行的最小单位。

线程也有对应的状态，就绪态-执行态。（阻塞态不按顺序）。

线程的特点：

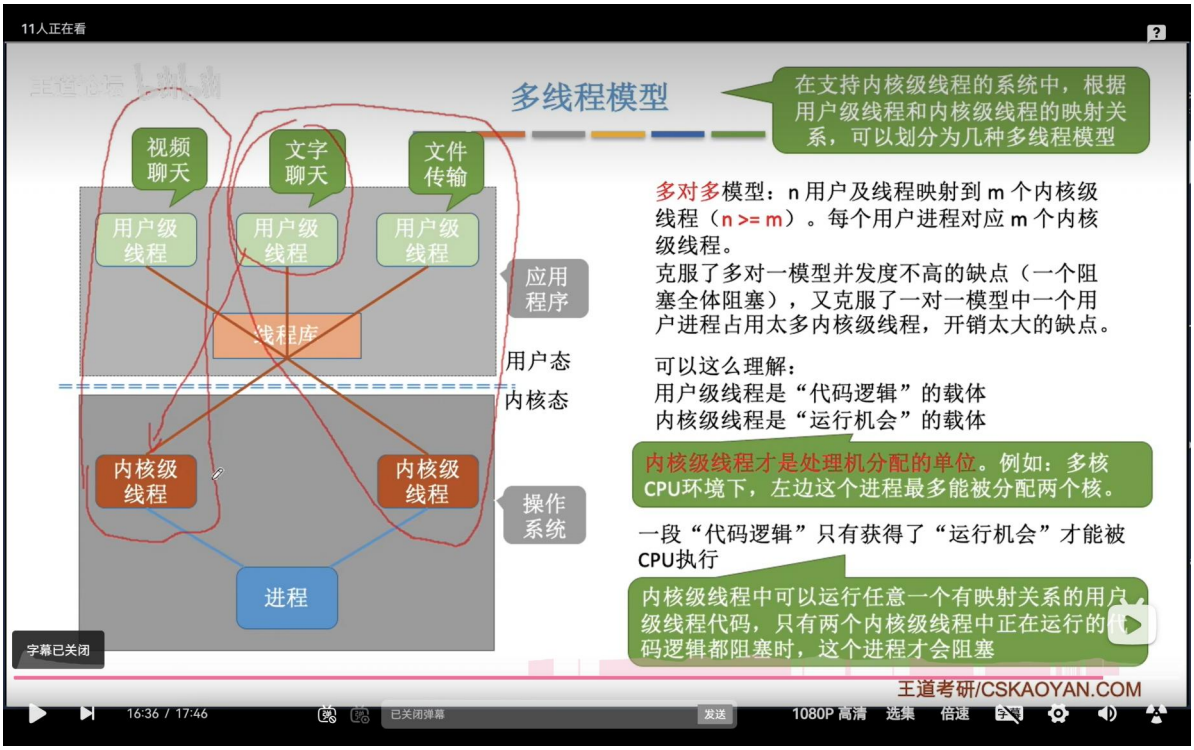
1.共享进程资源

2.线程切换快，不用切换进程
3.TCB,线程控制块，记录线程基本信息

引入线程：进一步提升os的并发度，原本进程并发，现在线程并发。
线程的实现方式：ULT user-level-thread用户级线程，类似代码+while循环的方式，让用户设计线程，被称为ULT。并且cpu执行这个ULT时切换到用户态。

KLT
：内核级线程，kernel-level thread，一个内核级线程对应一个ULT，也就是说，线程切换需要cpu变态。
KLT的好处是：os调度的基本单位是thread，如果是多核cpu可以并行。
KLT的坏处是：线程切换麻烦。

多线程模型

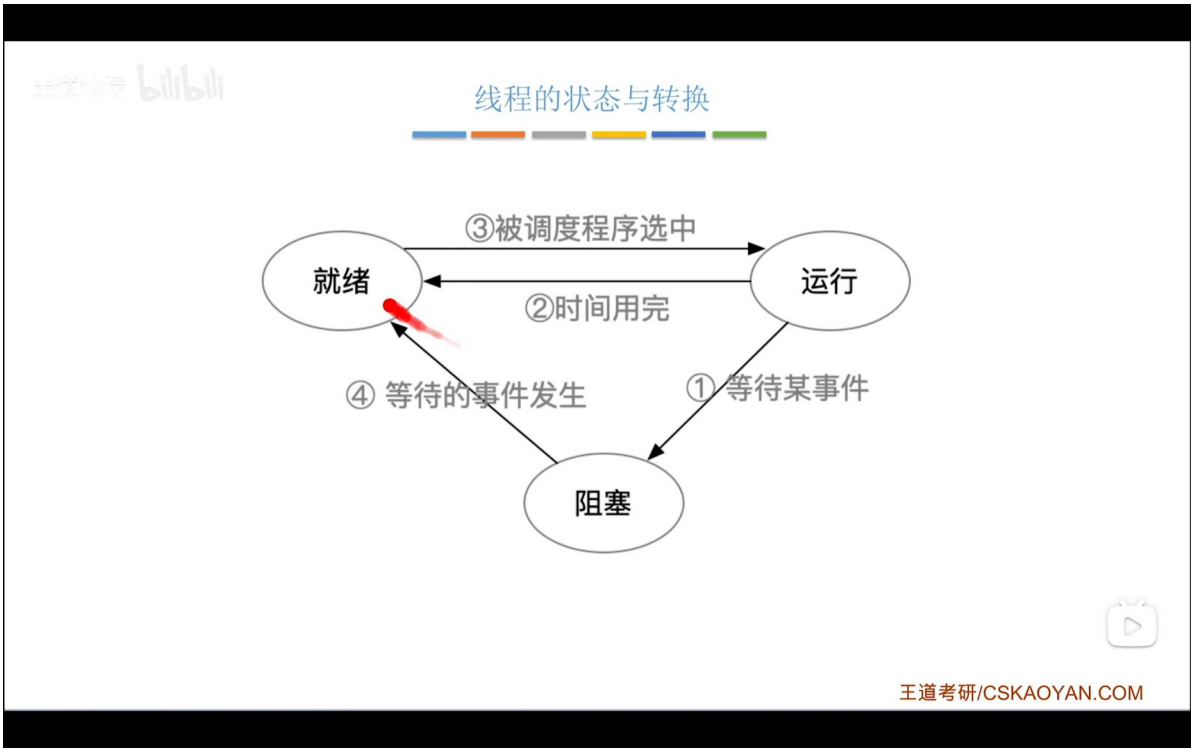


多对多模型：一个线程库给多个核用，一个线程库可以分开运行。

线程表

TCB：thread control table 线程控制块
Thread table:存储一个proc中的多个线程的TCB

线程的状态转换



调度和分派

Scheduler-Dispatcher

调度是：选择调度策略，即哪个进程先运行；以及选择运行的进程。

分派是：运行的具体过程，包括进程切换时上下文的保存，cpu的状态转换，地址切换到用户程序并运行。

处理机调度

1.高级调度：在外存的作业后备队列中，选取job。job是一个进程的集合，job是一个任务，调度job就是决定执行哪个job。

2.中级调度：选取挂起队列中的进程。

3.低级调度：调度进程的PCB，从**就绪队列**中选取一个进程。

我认为调度可以改名为选取。

挂起和阻塞

1.挂起：挂起是指把一个进程移出内存，并把PCB放入挂起队列中，需要运行进程时取回进程数据。

2.阻塞：阻塞的进程依旧在内存中，阻塞进程的PCB放置在阻塞队列中。一旦阻塞的事件完成，进程被移入就绪队列中。

调度的方式

1.剥夺调度方式（抢占式）

2.非剥夺式：等待proc主动让出cpu，

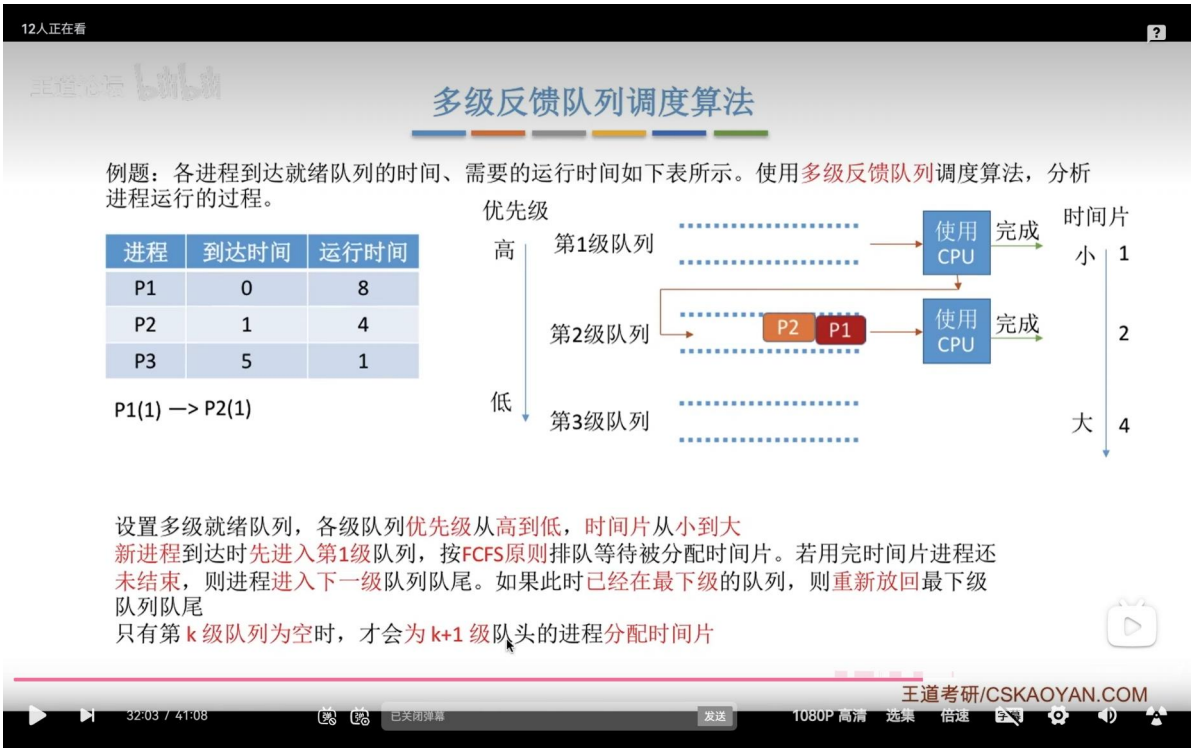
调度算法的评价指标

1.CPU利用率：CPU运行时间/周转时间

2.周转时间：turnaround，job完成时间-job开始时间，

平均周转时间 = $周转时间/job数量$

带权周转时间=周转时间/实际运行时间
响应时间=首次开始运行时间-arrival time
等待时间=周转时间-运行时间
调度算法
1.FCFS-serve
2.SJF-short job first，短-运行时间短
3.HRRN=high response ratio Next高响应比优先，响应比：等待时间+剩余运行时间/剩余运行时间，HRRN是**非抢占式**
4.优先级
5.时间片轮转(roundrobin)
6.多级反馈(multilevel feedback)，抢占式。时间片+优先级 + n个队列 + FCFS



越靠后的队列时间片越大。越靠前的队列越早被运行。
按先后顺序插入到队列中
7.多级队列调度，n个优先级队列，越靠前的队列优先级越高。但是队列是按进程类型分类的，每个队列可以再采取各自的调度算法。
8.SRTF：short remaining time first，最短运行时间优先，抢占式的SJF。
9.SPN:shortest process next，同SJF。

临界区问题

critical section problem.当进程使用共享资源时，称进程处于临界区(CS)。CS状态下是不允许调度的，也就是说，CS状态下的进程不允许切换状态。并且CS状态下的进程必须：**互斥+无饥饿**

同步和互斥

1. 同步：进程之间有某种合作，这种合作体现在进程执行的顺序，原本是独立平行执行，现在是按某种顺序执行。
2. 互斥
3. 软件方法：单标志法，双标志先检查，双标志后检查，peterson。turn,flag,flag,flag+turn,turn做谦让。

4. 硬件方法：关中断+cs访问+开中断；TestAndSet(和锁差不多)；Swap(exchange)；

信号量（必考）Semaphores

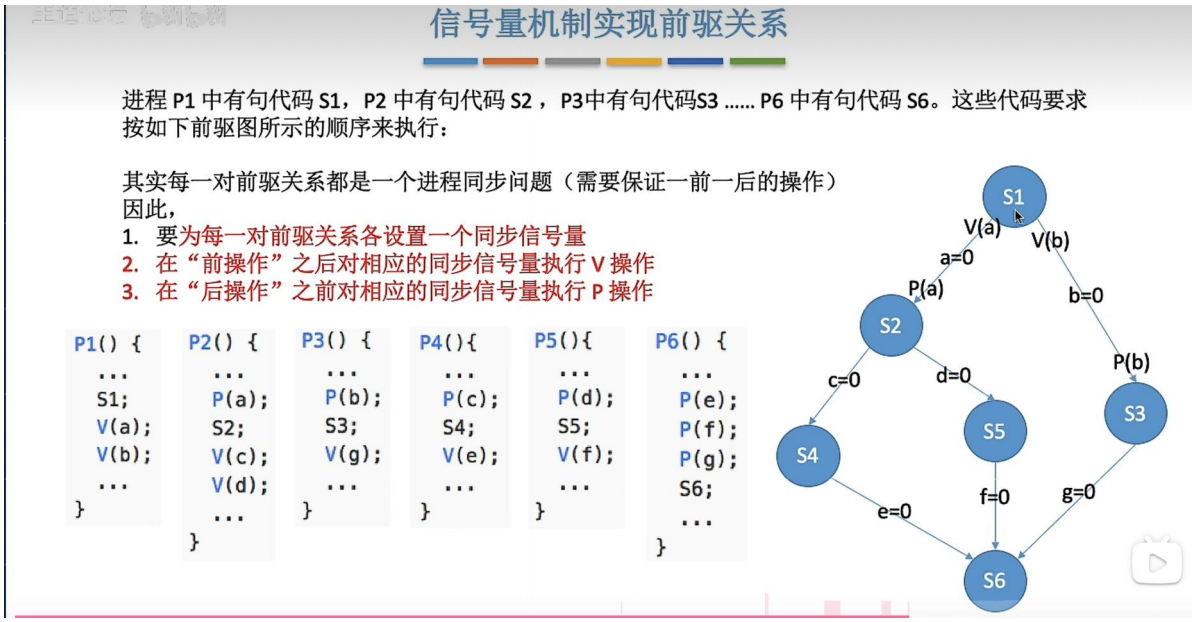
1. 信号量：

```
struct semaphore{
    int value;
    struct queue q;
}
void wait(s:semaphore){
    s.count--;
    if s.count<0
    {queue.append(proc);}
    block proc;
}
void signal(s){
    s.count++;
    if (s.count<0){//如果队列中还有在等待的话
        queue.popout(proc);
        ready proc;
    }
}
```

s.count:当count<=0时，count的绝对值表示queue中有多少proc在等待。当count>0时，count表示还有多少资源可以使用。

2. 信号量可以实现同步、互斥和前驱

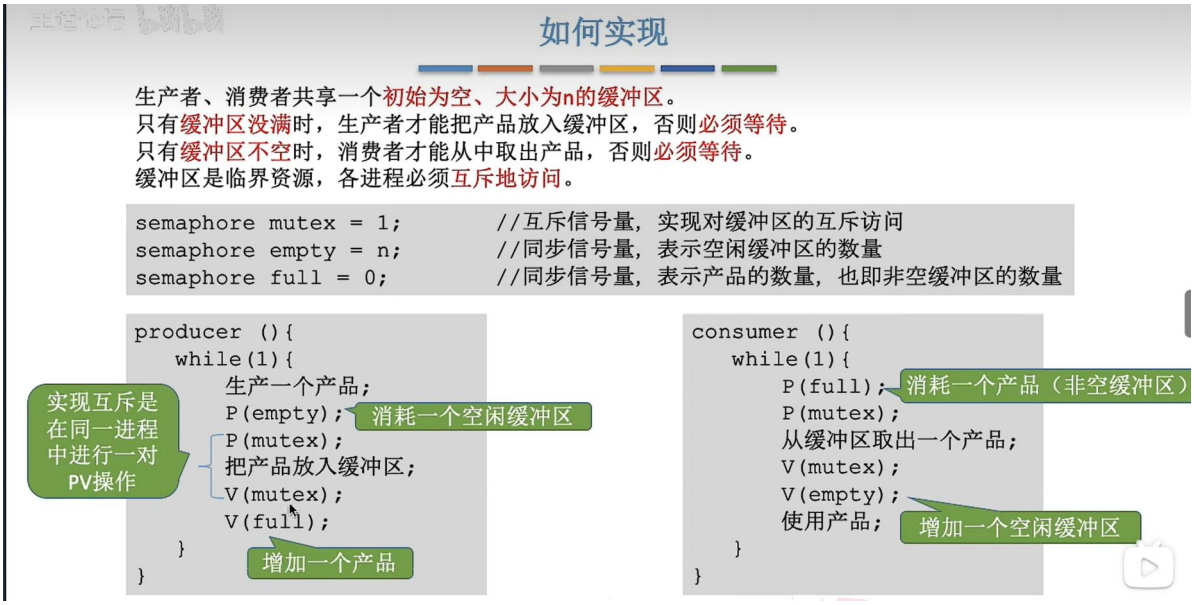
```
互斥：初始化s为1
    wait();cs;signal();
同步：proc2必须在proc1执行完signal之后才能执行，这里假设s初始化为0
    Proc1:
        代码1；
        代码2；
        signal(s);
        代码3
    Proc2：
        wait(s);
        代码4；
前驱：
```



图中V等价为signal，P等价为wait

生产者消费者问题

问题简介：生产者生产某样东西，输入到buffer中，然后消费者获取。
条件：1.生产者生产必须互斥，否则可能会生产到同一个buffer中
2.buffer有限，当buffer满时，不允许生产。当buffer空时，不允许消费。
3.消费也需要互斥。
实现方法：1.信号量Mutex 2.信号量Full 3.信号量empty

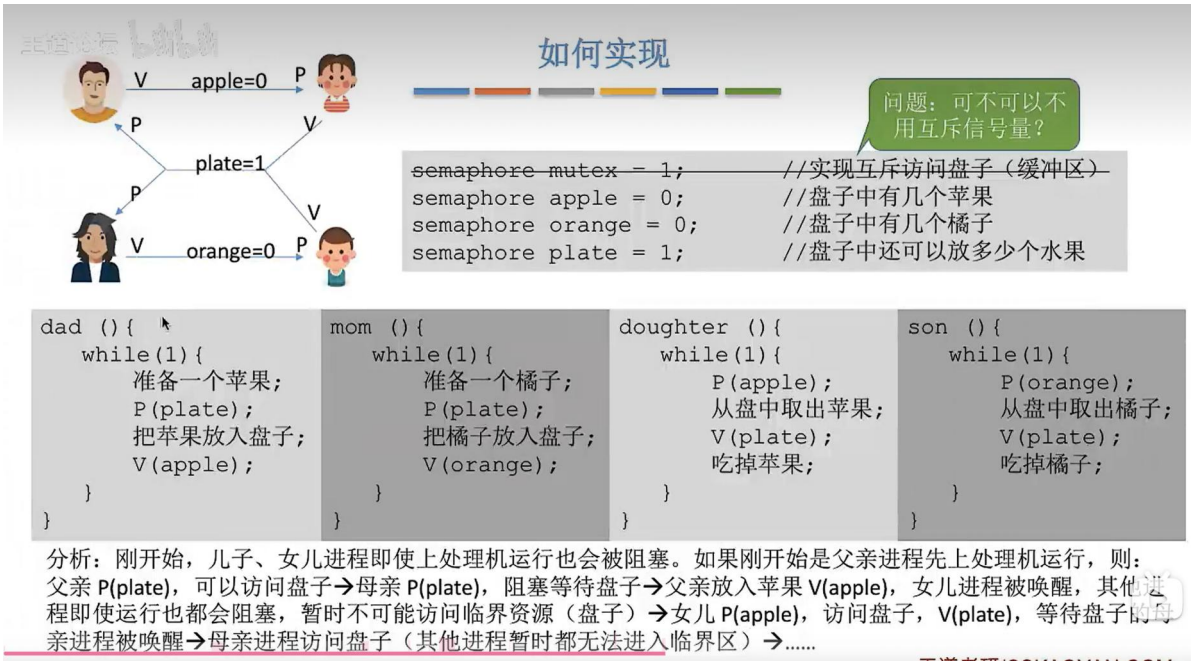


注意持有信号量的顺序。full和empty都不能放在mutex中，否则如果empty=0时，占用了mutex，直接死锁了。

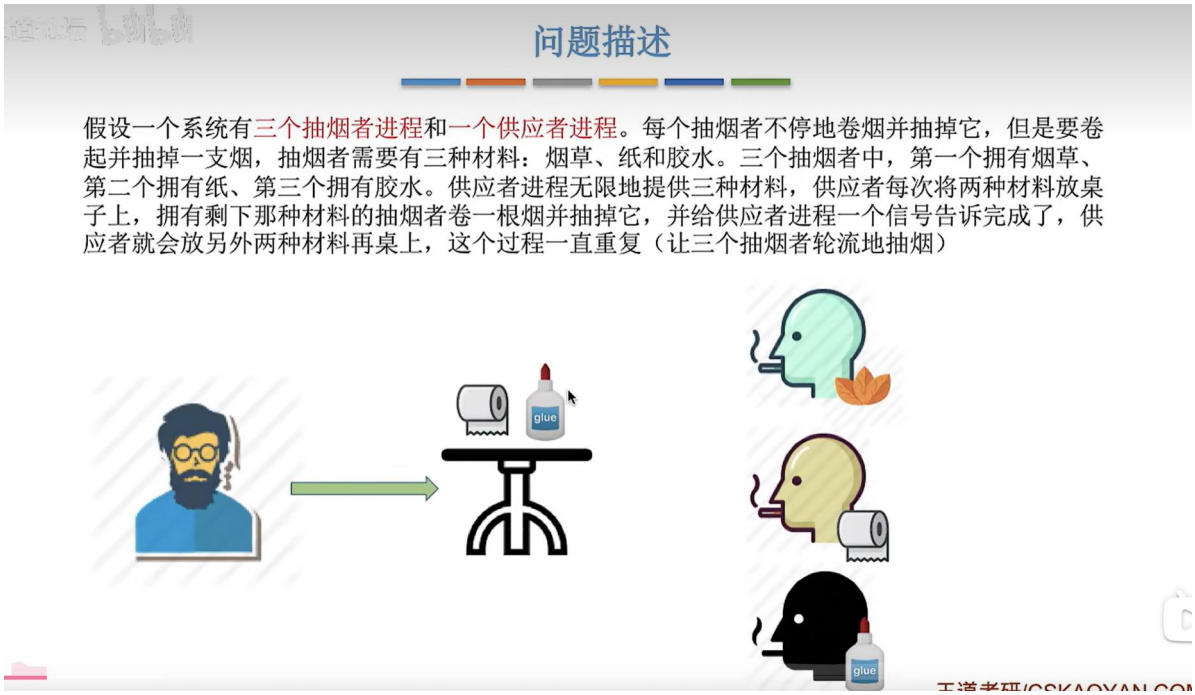
如果将empty和full同时放入mutex，那也一样。

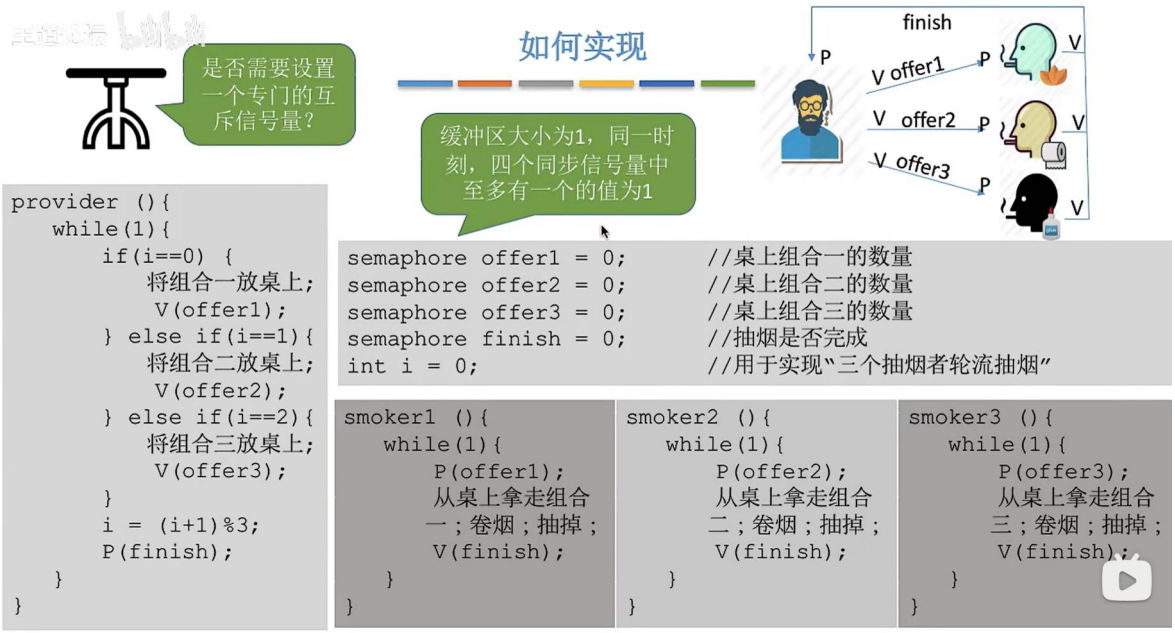
```
Initialization: S.count=1; //mutual excl.
                N.count=0; //full spaces
                E.count=k; //empty spaces

    Producer:                Consumer:
    While (1)                While (1)
    { produce v;             { wait(N);
      wait(E);                 wait(S);
      wait(S);                 w:=take();
      append(v);               signal(S);
      signal(S);               signal(E);
      signal(N);               consume(w);
    }                        }

    append(v):               take():
    b[in]:=v;                w:=b[out];
    in++;                    out++;
         mod k;                   mod k;
                             return w;
```

多消费者多生产者：父母和儿女问题



吸烟者问题





小总结：同步互斥问题说白了就是先V后P问题，通过不断地VP顺序使进程按自己想要的顺序运行。

读写问题

问题描述：1.读者可以同时读文本 2. 写者要互斥的写文本 3.如果写者要写文本，读者不能读文本。 4.写者如果要写，就必须尽快安排，不能让其饿死。

实现：

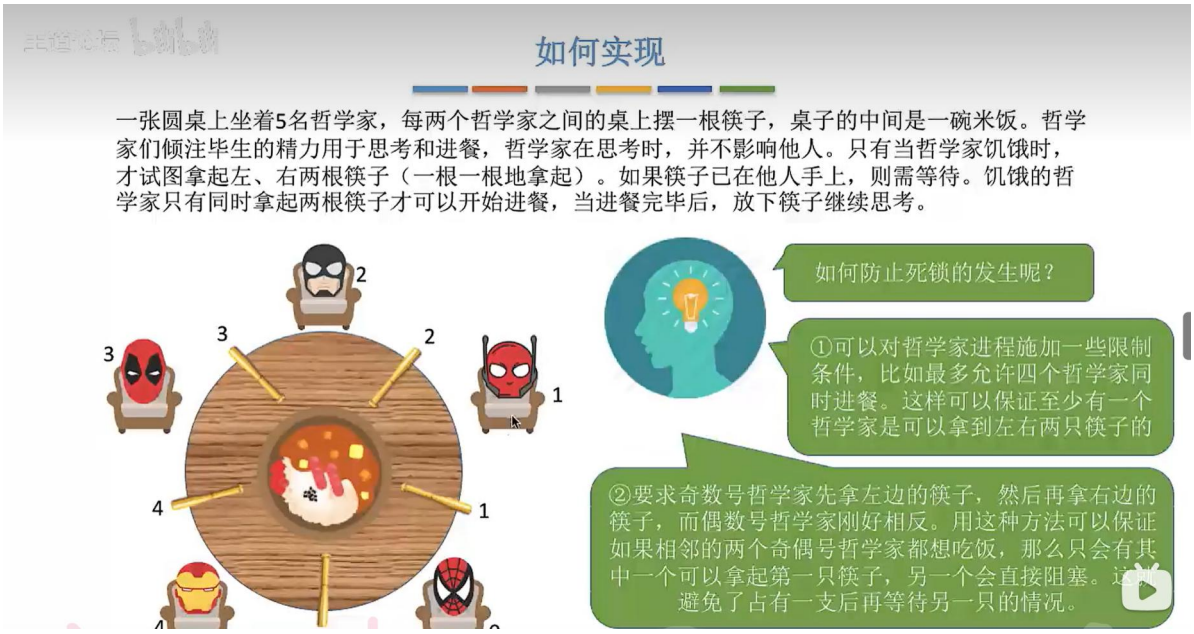
1,2：一个lock，读者共享这个lock，写者每个都要这个lock

```
struct semaphore{
    int value;
}
semaphore lock1 = 1;//只有一个Lock1会导致reader不同时，原因是，如果两个同时都是count==0要求lock1时。
```

```
semaphore lock2 = 1;
semaphore lock3 = 1; //只有两个lock时，如果读者获取了lock1，并且读者很多，那写者会饿死。
int count;//global
void reader()
{
    P(lock3);
    P(lock2);
    if count==0 P(lock1);
    count++;
    V(lock2);
    V(lock3);
    read;
    P(lock2);//count--需要lock2的原因是，否则两个reader的count也会同时为0，直接把lock2变成大于1了。
    count--;
    if count==0 V(lock1);
    V(lock2);
}
void writer()
{
    P(lock3);
    //lock3可以让read不能无限的占有lock1，假设读者1-读者2-写者1-读者3，
    //首先读者1，读者2会依次进入read状态，
    //这时读者3不能更改count，然后read过程结束后，最终会释放lock1，
    //写者就可以运行。
    P(lock1);
    write;
    V(lock1);
    V(lock3);
}

```

哲学家进食

哲学家问题就是拿筷子问题+避免死锁。



让奇数号拿左边筷子，偶数号拿右边。

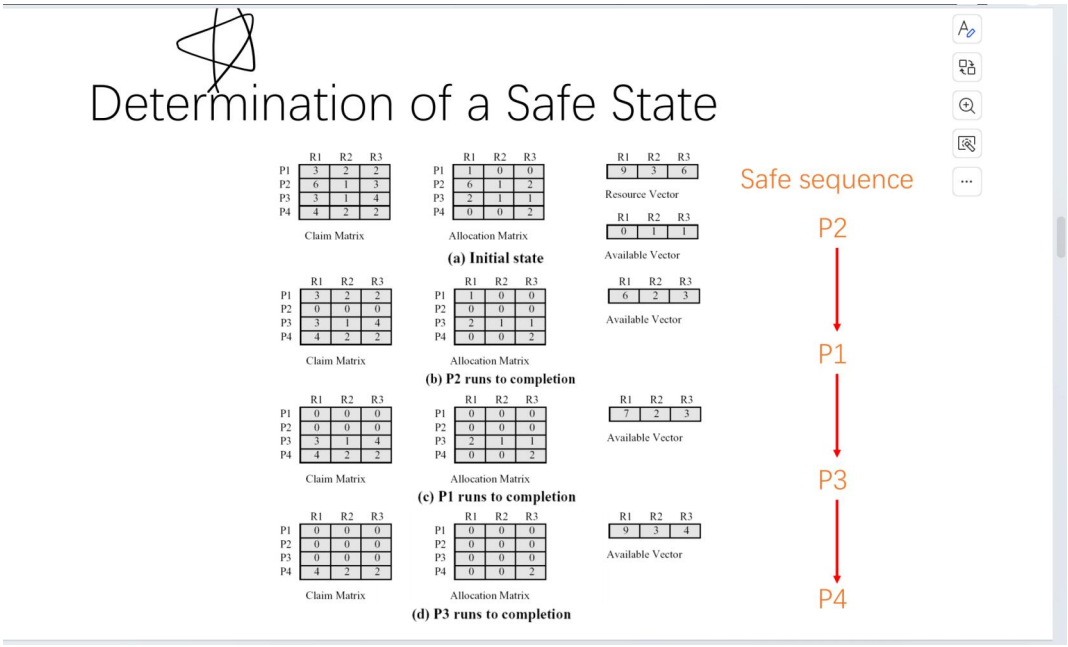
管程

我的理解是，把生产者消费者问题打包成一个数据结构，数据结构中包含了insert和takekout。
然后这个数据结构中自然实现了互斥和同步。

死锁

1. 死锁，饥饿，死循环：死锁指hold and wait,饥饿指永不被执行，死循环指进程代码写错了。
2. 预防死锁：1.破坏互斥 2.破坏hold and wait 3.抢占式 4.破坏循环等待
3. 避免死锁：银行家算法banker algorithm
   安全序列：对于多个进程P1,P2,...，每个进程要求不同的资源，如果存在一个序列使得所有的进程都能够平安无事的使用资源释放资源，并且最终完成的话，那么这就是一个安全序列。

存在安全序列的状态叫做安全状态。

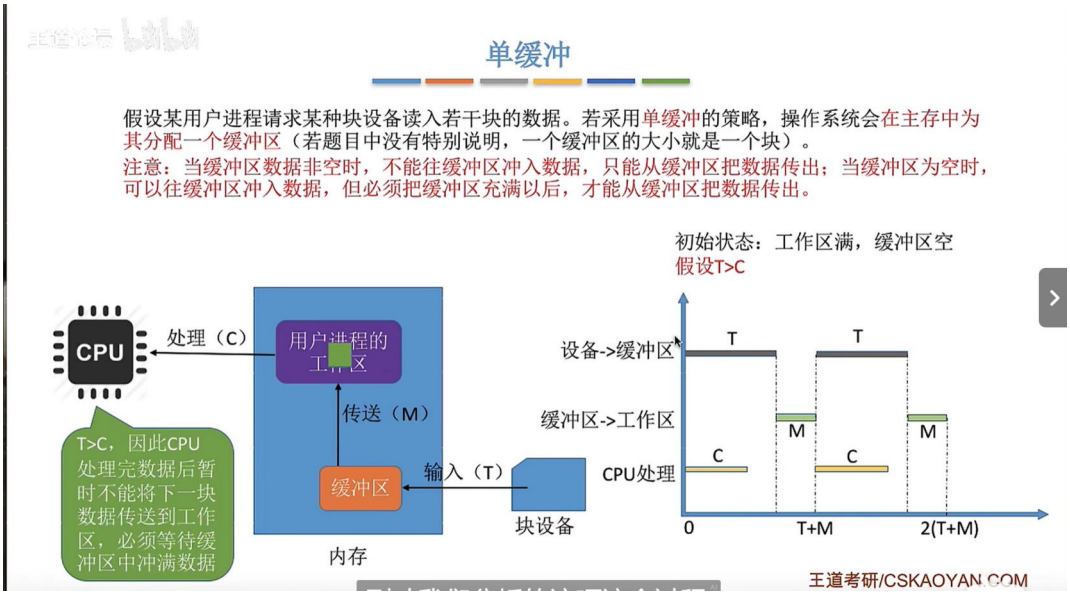


4. 死锁检测，死锁检测和找安全序列的区别是一样的。如果找不到安全序列就说明死锁了。
   检测到死锁后，操作系统一般会采取1.挂起某个进程，释放其资源供其他进程 2.回退，让多个进程回退到没有抢占资源的时候 3.撤销，撤销某些进程，这方法太严重了。

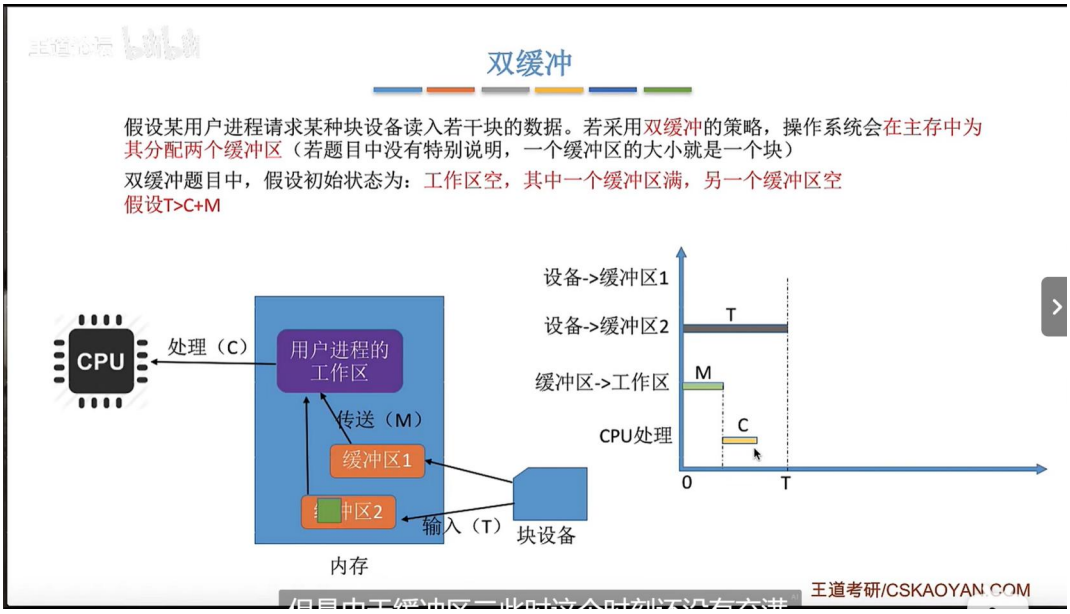
IO

缓冲：主要是要会画这个缓冲的过程图，首先是IO传到缓冲区，然后缓冲区才能传到工作区，工作区再传到CPU。

1. 单缓冲：

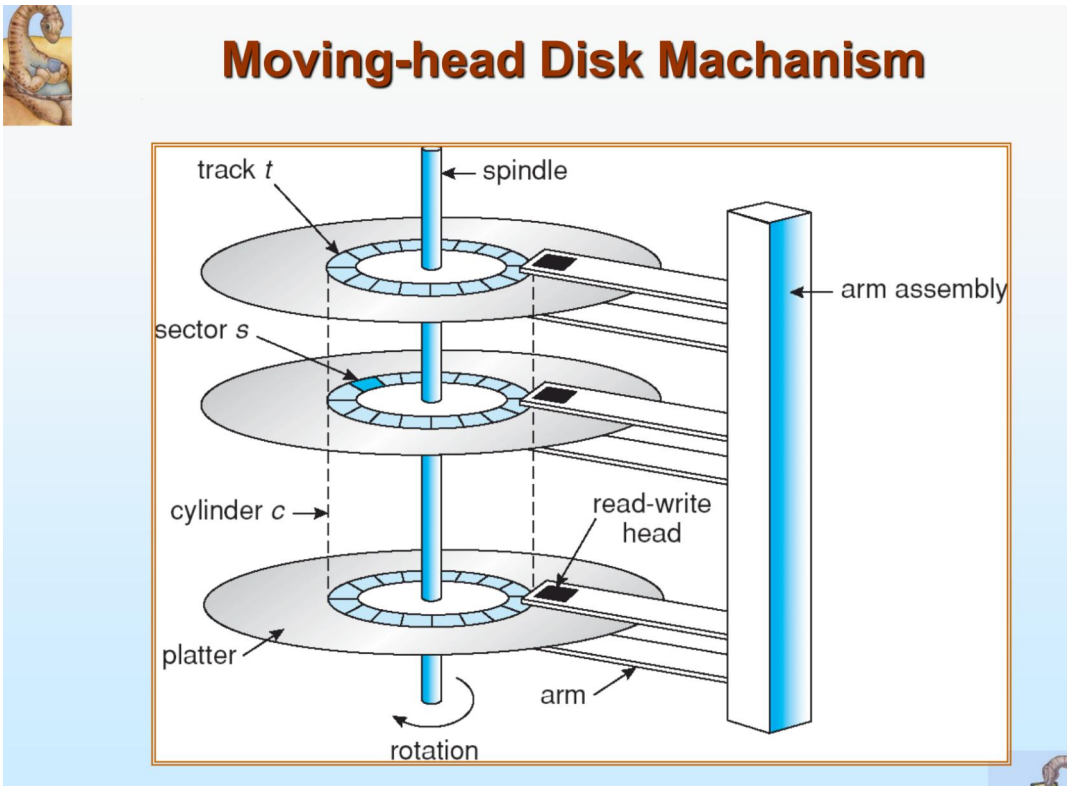


2. 双缓冲：

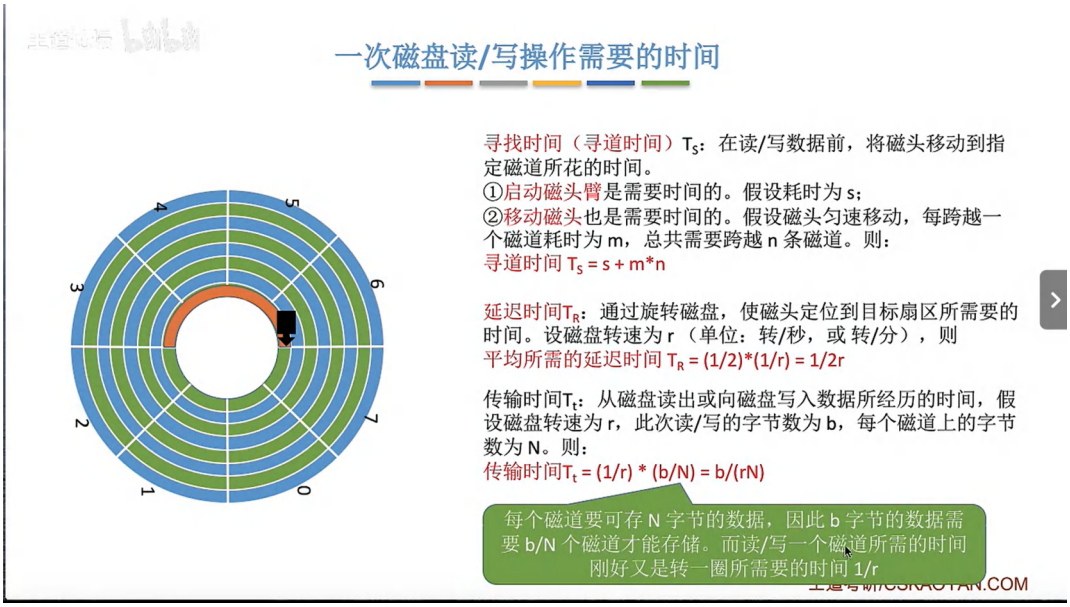


Disk

1. 柱面：cylinder
2. 盘面:platter
3. 盘区:sector



4. 寻道时间，延迟时间，传输时间，访问时间:



disk调度策略

1. FCFS：先来先服务
2. SSTF：最短寻道时间优先
3. SCAN：回到头，去到尾
4. C-SCAN：不撞南墙不回头
5. C-LOOK：撞到南墙前就回头
6. N-STEP-SCAN：FCFS-SCAN
7. FSCAN：两个queue，一个用SCAN，一个存新请求

6，7可以避免粘连问题，arm-stickness

disk格式化:

1. 物理格式化：给磁盘分区，并且添加一些检验文件是否完整的校验部分。一个扇区分为头部，尾部，和数据部。头部尾部就放这种检验的东西。
2. 逻辑格式化：创建文件系统，创建一个根目录和维系目录的一堆东西。初始化存储空间的数据结构。
3. BOOT Block：引导块，用以初始化系统，开启os，初始化寄存器等。

note8考点:

1. 操作系统是如何管理空闲内存的：链表
2. 交换：backing store是什么?
3. 页表是如何记录页的帧的，LA如何变成PA。LA如何记录。valid位，modified位，contro位。
4. 多级页表如何根据LA查找页号。
5. TLB计算平均访问时间。

6. 段表和页表的不同：段表记录<开始地址,段长度>。
7. 段页表如何找到PA。

note9考点：

1. 缺页中断机制
2. 缺页策略：FIFO,LRU,CLOCK,LFU,MFU
3. 页面缓存
4. 内存分配策略：局部/全局，可变/固定，固定和全局不能等同
5. 工作集如何帮助全局分配。

note10考点：

1. 文件内部数据存储：(记录式only)，如何存储；顺序，索引，顺序索引，哈希。
2. RWX，111表示owner，110表示group，001表示public。
3. FCB，file control block。

note11考点：

1. disk属性之：file allocation policy：continugous,linked,indexed.
2. FAT项：文件名，起始块，长度
3. 不同policy的FAT差异：c和l都是2，indexed是文件名，索引表块。
4. 空闲空间存储：bit法，链表法，索引法。
5. open file table，进程和os的oft区别。
6. 硬链接软连接：索引节点|符号链。