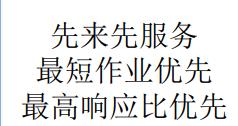
线程相关：

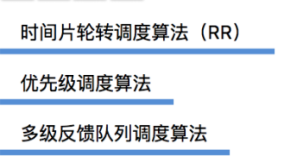
1：线程调度算法：



最高响应比：计算公式： 等待时间+执行时间/执行之间

执行时间越短， 响应比越高

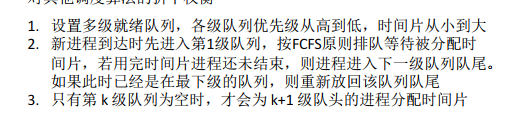
等待时间越久，响应比越高

2：

时间片算法： 分成多个时间片，时间片结束后 换下一个

优先级调度：优先级高的算法优先得到调度

多级反馈队列：



第一级是按照先来先到执行，

接下来的k级是时间片轮转算法；

死锁

死锁的4个条件

互斥 建立池的概念

循环等待 给出一个序列号，按照顺序分配;

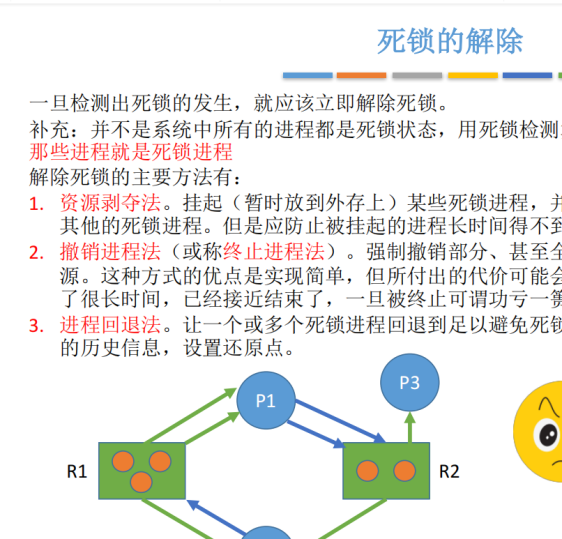
不可剥夺

:允许按照优先级 剥夺，或者请求新资源失败的时候释放自己的所有资源

请求且保持 在已有资源的时候 请求新的资源

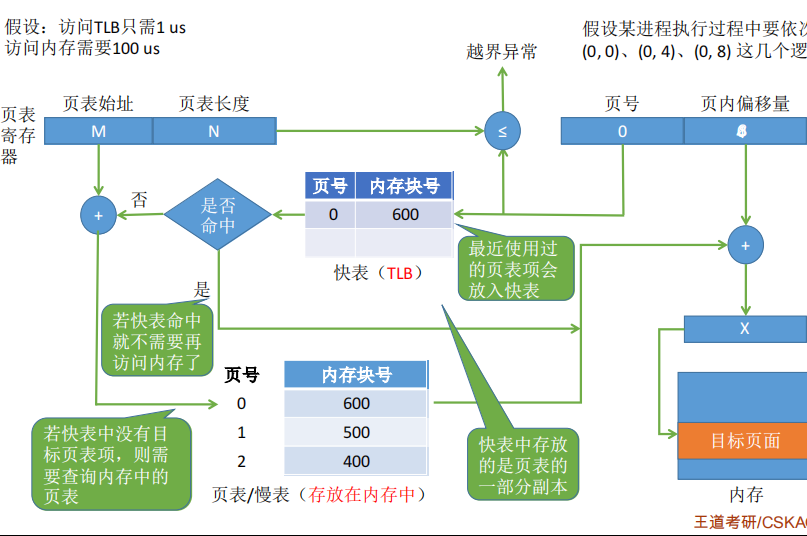
:>>请求所有的资源才可以开始执行

死锁的预防：银行家算法 给出一个安全的执行序列



内存相关：

TLB快表的访问过程：

最近使用过的内存块会放到TLB表里面；  


内存页置换的算法：



OPT找到 内存已有的 内存页将来不会使用的那个

LRU 找出最近最少使用， 就是时间线向前找最远用过的

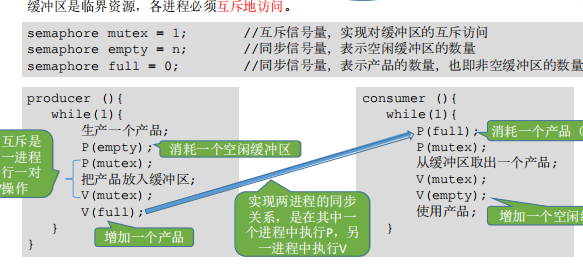
CLOCK 时钟转着 为1则

就是利用软件技术， 把已有的物理资源做成可以共享的资源；

生产者消费者模型；

1： 设置一个锁 然后 缓冲区大小为3；

生产的速度很快，消费的速度很慢；



java代码：

public static void main(String[] args) {

Semaphore full = new Semaphore(0);//表示 empty();

Semaphore empty = new Semaphore(3);//表示 empty();

Deque<Integer> dq = new LinkedList<>();

Object lock=new Object();

new Thread(new Runnable(){

//生产者代码

public void run(){

while(true){

try {

empty.acquire();

} catch (InterruptedException e) {

e.printStackTrace();

}

synchronized (lock){

dq.offerFirst(1);

System.out.println("放入了一个资源");

}

full.release();

}

}

}).start();

new Thread(new Runnable() {

//这里是消费者代码

@SneakyThrows

@Override

public void run() {

while (true){

try {

full.acquire();

} catch (InterruptedException e) {

e.printStackTrace();

}

synchronized (lock){

Thread.sleep(500);

Integer first = dq.pollFirst();//直接取，保证有;

System.out.println(first);

}

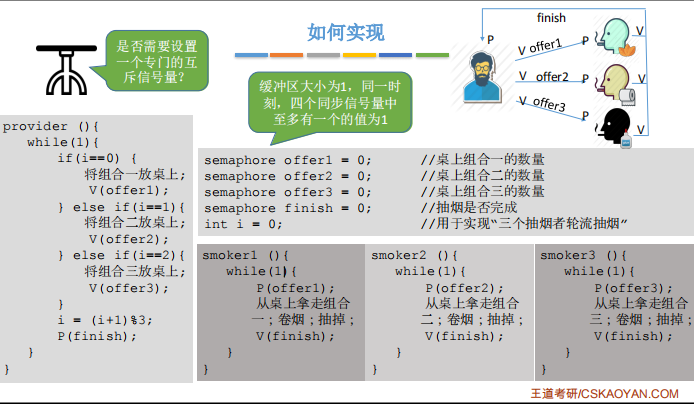
empty.release();

}

}

}).start();

}



把组合抽象出来就可以了;

