第4章 处理器调度

一、操作系统中的调度

1.什么是调度

调度(Scheduling)是管理的一种方法、是一种决策，资源（如工作、人力、车辆等）经过管理得到合理、有效地利用。调度的目标是找出一种合理的、有效的安排方法，提高资源的利用率。

2.操作系统中的调度

- 作业调度
- 进程调度
- 交换调度
- 设备调度

3.调度的性能指标

- 周转时间和平均周转时间

Ti = 作业Ji的完成时刻 - 作业Ji的提交时刻

$$T = (\sum_{i=1}^{n} Ti) / n$$

- 响应时间

  R = 请求处理过程第1次得到结果的时刻 - 请求提交的时刻
- 评价调度性能的其他指标
  - 公平合理
  - 提高资源利用率
  - 吞吐量

二、作业调度

1.作业状态

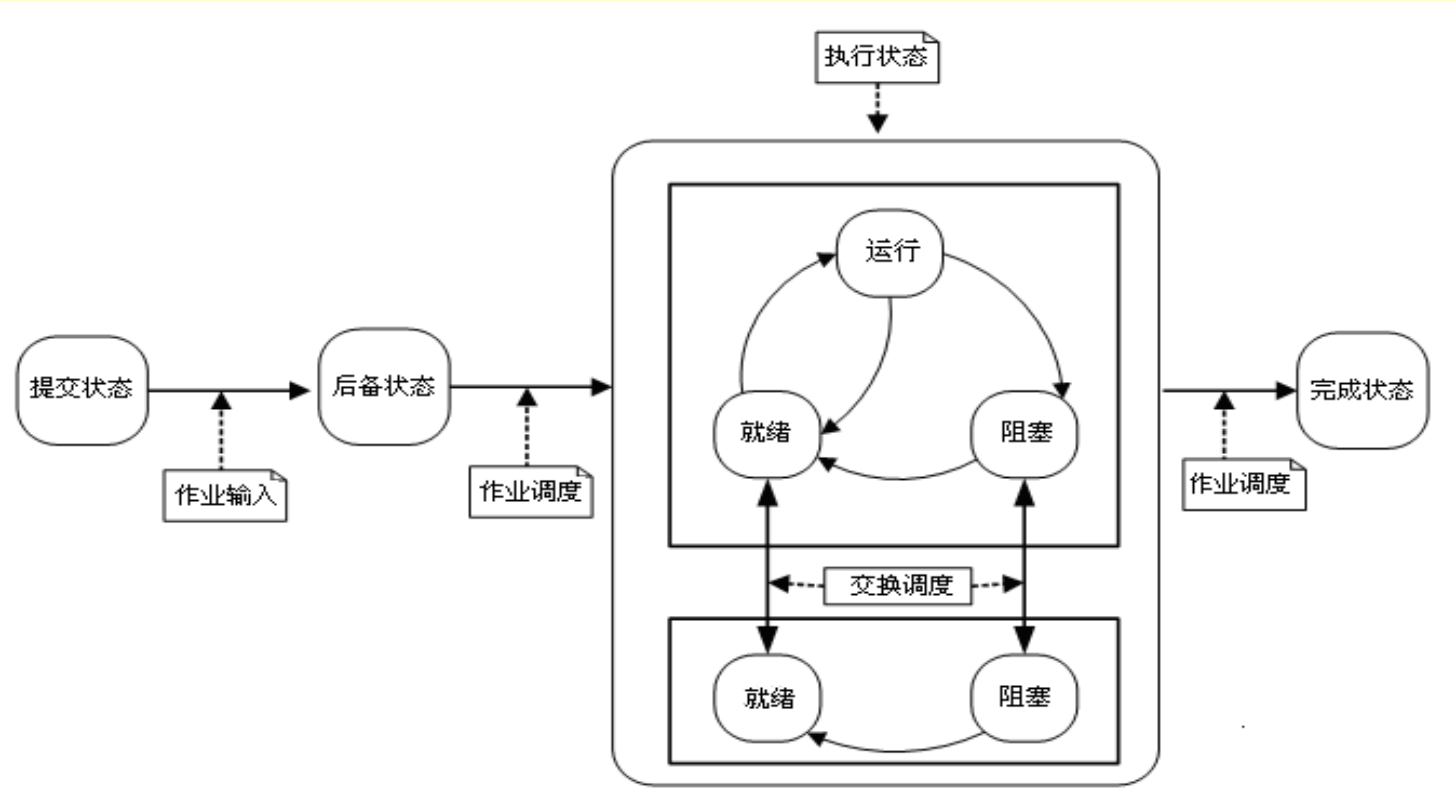

图4-1 作业状态及其转换

2.批处理系统为什么需要作业调度？

3.作业调度的主要功能

- ◆ 设计数据结构，登记调度所需要的参数
- ◆ 执行指定的算法，从作业的后备队列中选择一个作业
- ◆ 为选中的作业分配资源，创建进程
- ◆ 作业完成时的资源回收

4.作业调度算法

- ◆ 先来先服务算法(FCFS)
  - 思想：排队
  - 特点
    - 公平合理
    - 算法简单，容易实现
    - 服务质量欠佳(有于大作业，不利于小作业)

例子： 假定在某单道批处理系统中，一批作业A、B、C和D在同一时间先后几乎同时到达。已知它们都是纯计算性的简单任务，运行时需要占用处理器时间分别是10、3、2和5。把到达时间(提交时间)设为0。

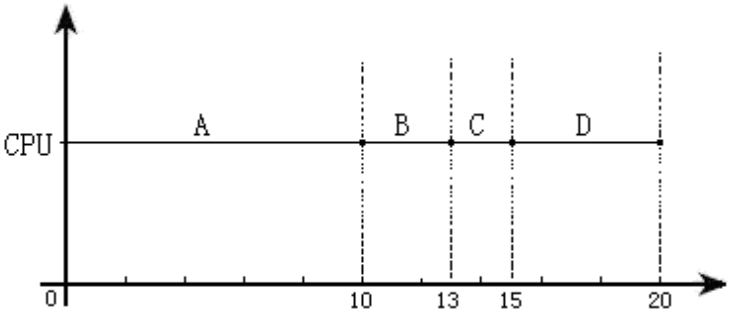

图4-2 FCFS算法例子调度图

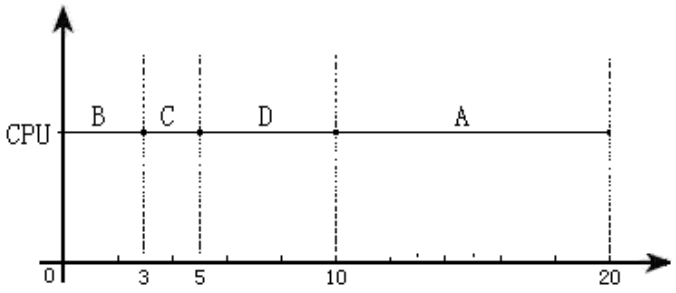

图4-3 非FCFS算法时的一种调度图

TA=10，TB=13，TC=15，TD=20；

T=(TA+ TB+ TC+ TD)/4

=(10+13+15+20)/4

=14.5

TA=20，TB=3，TC=5，TD=10；

T=(TA+ TB+ TC+TD)/4

=(20+3+5+10)/4

=9.5

◆ 短作业优先算法(SJF)

- 一个作业运行时所需的处理器时间总和，简称为作业大小
- SJF思想
- SJF特点
  - 算法思想简单，但实现困难
  - 拥有最小平均周转时间,吞吐量大
  - 存在“饥饿”现象

◆ 高响应比优先算法(HRN)

- 一个作业的响应比R是

$$\frac{\text{作业等待时间}}{\text{作业大小}}$$

其中，作业等待时间 = 系统当前时间 - 作业提交时刻

- HRN思想
- HRN特点
  - 综合了先来先服务算法(FCFS)和短作业优先算法(SJF)
  - 响应比R与作业的大小成反比，体现SJF算法
  - 响应比R与作业的等待时间成正比，体现FCFS算法

4.作业调度算法例子

例4-1 假定在某脱机单道批处理系统中，有一批作业，它们的提交时刻和作业大小如表4-1所示，假定在10:00时开始调度，求分别采用FCFS、SJF、HRN作业调度算法时的调度顺序、各作业的周转时间、各算法的平均周转时间。

表4-1 例1的作业信息

作业号	提交时刻	作业大小(小时）
J_1	9:00	0.8
J_2	9:10	1
J_3	9:45	0.6
J_4	10:00	0.4

例4-2 在某联机单道批处理系统中，有一批作业，它们的提交时刻和作业大小如表4-5所示。分别采用FCFS、SJF、HRN作业调度算法时的调度顺序、各作业的周转时间、各算法的平均周转时间。

表4-5 例2的作业信息

作业号	提交时刻	作业大小(小时）
J_1	9:00	0.8
J_2	9:10	1
J_3	9:45	0.6
J_4	10:00	0.4

三、进程调度

1.进程调度含义

2.进程调度功能

- 进程调度方式：运行状态的进程何时以什么方式停止或暂时停止运行
- 进程调度算法：从就绪队列中按照指定的算法选择一个进程，准备执行
- 处理器切换
- 进程结束时资源回收

3.进程调度方式

- 非抢占方式(Nonpreemptive Scheduling)
- 抢占方式(Preemptive Scheduling)

常见的原则有：时间片原则、优先级原则、任务紧迫性、重要性原则等等。

进程调度方式实现进程之间的轮流交替的一个方面。

4.进程调度算法

- 先来先服务算法(FCFS)
- 时间片轮转算法(RR)

RR算法需要设计一个定时器，定时器的值为0时将产生一个中断。系统用分配给进程的时间片设置定时器的初值，之后进程开始执行。进程运行过程有三种可能情况：

例4-3：假定某分时系统有3个同时依次到达的进程A、B和C，它们的任务如下：

进程A：	进程B：	进程C：
2ms CPU 10ms I/O 2ms CPU	9ms CPU 5ms I/O 2ms CPU	8ms CPU

在采用简单RR算法，时间片为3ms时，请画出RR算法的调度图。

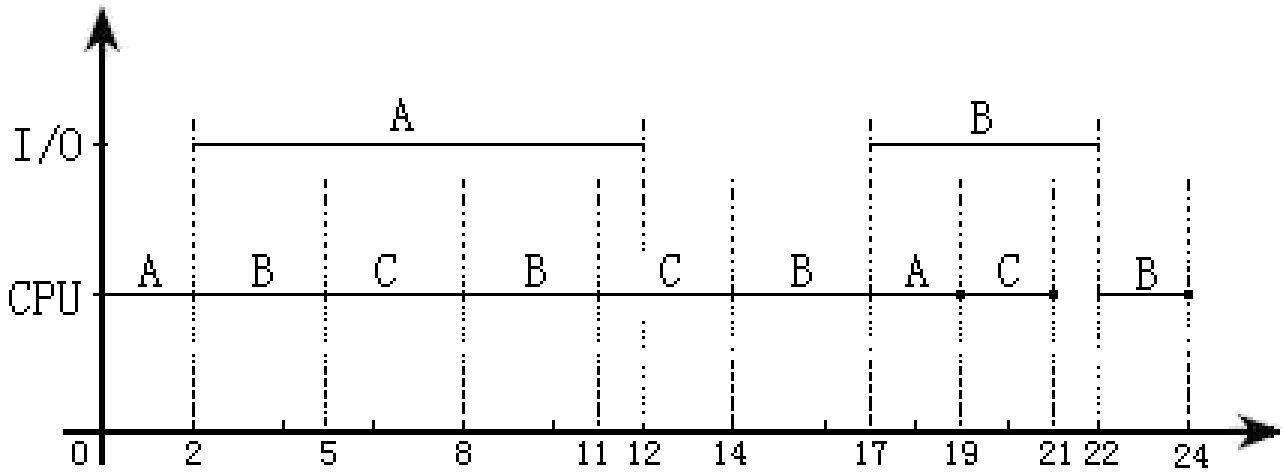

图4-4 简单RR算法的调度图

➢ 响应时间

简单RR算法，假设就绪队列中的进程数为n，时间片为T，那么，响应时间R，则

$$R = T*n$$

- 优先级算法(Priority)
  - 思想
  - 实现关键
    - 静态优先级/动态优先级
    - 非抢占优先级/抢占优先级
    - 优先数的确定
- 多级队列算法

  UNIX系统为用户设置两个典型的队列：前台队列和后台队列

四、实时系统的进程调度算法

1.实时系统中的任务通常分为周期性任务和非周期性的任务

2.实时系统的时间参数

- 任务就绪时限
- 任务开始时限
- 任务完成时限
- 任务处理时间

3.实时系统的可调度

- 一组事件或进程是可调度的(Schedulable)
- 系统是可调度的

设λ是单位时间内到达的请求数，μ是处理器单位时间可处理的请求数(处理器的处理能力)，那么，系统是可调度的必要条件是

$$\mu \geq \lambda$$

假定系统只有一个周期性任务，任务的周期为P，处理时间为C，那么

$$\frac{C}{P} \leq 1$$

那么，系统是可调度的 。

某实时系统要求处理n个周期性的任务，它们的时间周期分别是P1、P2、…、Pn，而处理时间分别是C1、C2、…、Cn，那么，在不考虑系统开销的理想情况下，如果满足

$$\sum_{i=1}^{n} \frac{Ci}{Pi} \leq 1$$

那么，这n个任务是可调度的。

4.时限调度算法

例如，有2个周期性任务A、B，它们的周期分别是20ms和56ms，处理时间分别是8ms和32ms。以完成时限为调度参数，那么，如何画出采用时限进程调度算法的调度图呢？

表4-9 任务A和B的事件产生时间表

任务A	发生时间	完成时限	任务B	发生时间	完成时限
A1	0	20	B1	0	56
A2	20	40	B2	56	112
A3	40	60	B3	112	162
A4	60	80	…		
A5	80	100			
A6	100	120			
…					

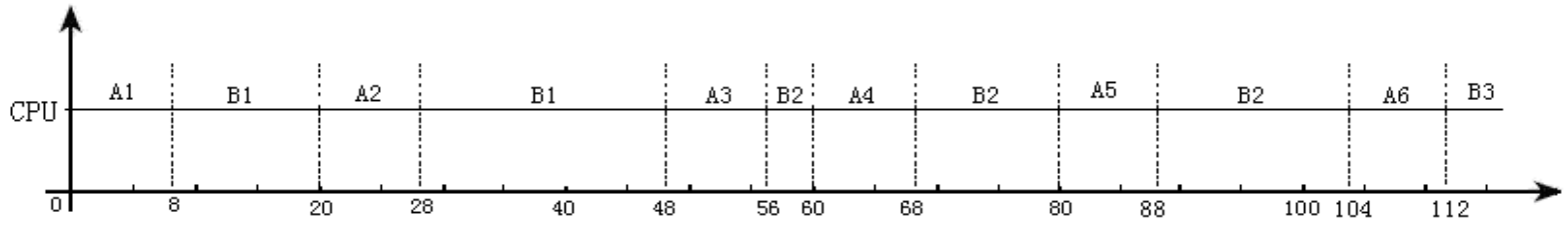

图4-5 时限调度算法的调度图

五、死锁

1.死锁(Deadlock)的含义

- 例子

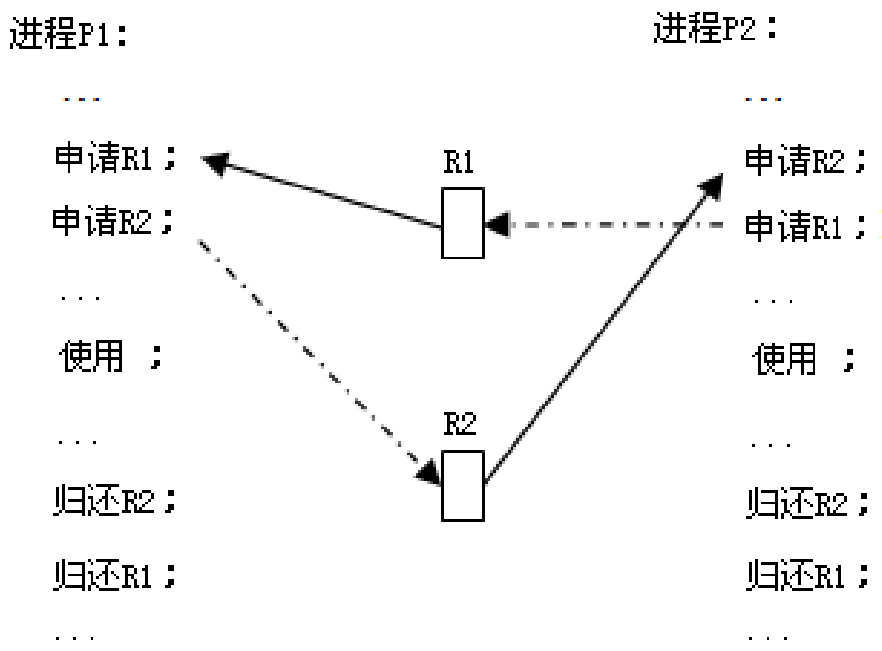

图4-6 进程死锁的例子

- 死锁的含义
- 死锁与程序设计中死循环有什么区别

	死循环	**死锁**
产生	必然性	偶然性
状态	执行状态	阻塞状态
原因	程序设计不当或编写错误	操作系统的管理、控制

2.死锁产生的根本原因

- 系统拥有的资源数量小于各进程对资源的需求总数

3.死锁的四个必要条件

- 互斥条件
- 不剥夺条件
- 请求与保持条件
- 环路等待条件

死锁解决方法有：预防、避免、检测与恢复等三种

4.死锁预防(Deadlock Prevention)

- 含义
- 方法
  - 互斥条件

    原则上不能被破坏，打印等个别资源可以采取虚拟技术
  - 不剥夺条件

    原则上不能被破坏。
  - 请求与保持条件

    静态分配：具有一般性，但事先很难准确地估计进程运行所要全部资源，且降低了资源的利用率

    资源暂时释放：仅限于个别资源的操作；进程不稳定，
  - 环路等待条件

    按序分配：具有一般性，但存在与静态分配的问题，且编号管理困难。

    单请求方式：不适用于复杂任务的进程

例4-4 进程P运行过程依次申请编号为：A2、A3、A5和A4。则采用按序分配时，进程P的资源应该怎样申请资源？

例4-5 [经典同步问题]哲学家用餐问题：有5位哲学家围坐在一张桌子周围共同讨论一问题。他们各自独立地或拿起筷子用餐或独自思考问题。假定桌上的每两位相邻的哲学家之间放一支筷子，每位哲学家在用餐时需要得到左右两边的筷子，然后才能用餐，用餐后放下筷子，又开始独立思考问题。如此反复。由于两位哲学家共享他们之间的一支筷子，哲学家们用餐和思考问题又具有随机性，那么，如何用信号量机制实现5个哲学家进程的并发执行。

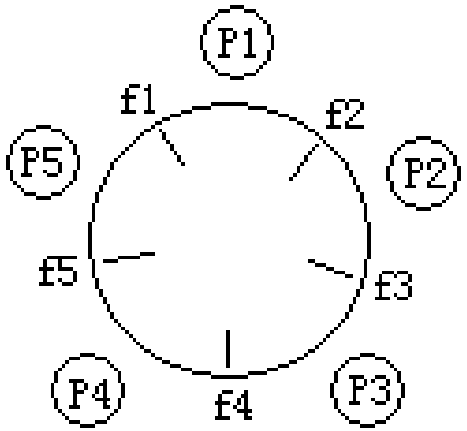

图4-8　哲学家用餐问题

表4-10 哲学家进程并发程序设计

```
P1()
{
思考;
p(s1);
拿右边筷子f1;
p(s2);
拿左边筷子f2;
用餐;
v(s2);
放下左边筷子f2;
v(s1);
放下右边筷子f1;
}
```

```
P2()
{
思考;
p(s2);
拿右边筷子f2;
p(s3);
拿左边筷子f3;
用餐;
v(s3);
放下左边筷子f3;
v(s2);
放下右边筷子f2;
}
```

```
P3()
{
思考;
p(s3);
拿右边筷子f3;
p(s4);
拿左边筷子f4;
用餐;
v(s4);
放下左边筷子f4;
v(s3);
放下右边筷子f3;
}
```

```
P4()
{
思考;
p(s4);
拿右边筷子f4;
p(s5);
拿左边筷子f5;
用餐;
v(s5);
放下左边筷子f5;
v(s4);
放下右边筷子f4;
}
```

```
P5()
{
思考;
p(s5);
拿右边筷子f5;
p(s1);
拿左边筷子f1;
用餐;
v(s1);
放下左边筷子f1;
v(s5);
放下右边筷子f5;
}
```

```
main()
{
cobegin
{
repeat P1();
repeat P2();
repeat P3();
repeat P4();
repeat P5();
}
}
```

4.死锁避免(Deadlock Avoidance)

- 安全状态和安全序列

Process	Max	Used
A	8	3
B	3	1
C	10	2
D	7	4

当前资源R的可用数：2

图4-9 安全状态的例子

Process	Max	Used	Need
A	8	3	5
B	3	1	2
C	10	2	8
D	7	5	2

当前资源R的可用数：1

图4-10系统不安全状态的例子

系统不安全状态：此时系统不存在进程安全序列。

- 死锁避免的含义
- 银行家算法(the Banker's Algorithm)

例4-7 某系统有4类资源A、B、C、D，数量分别为8、10、9、12。当前有5个进程P1、P2、P3、P4、P5，已经最大需求矩阵Max和当前分配矩阵Used如图4-11所示。

问：

1)当前系统是否为安全状态？

2)在图4-11状态下，如果进程P1申请request=(1，0，1，0)，系统能否分配？

3)在图4-11状态下，如果进程P3申请request=(1，0，0，1)，系统能否分配？

$$\text{Max}=\begin{bmatrix}4&6&3&8\\3&3&5&2\\6&6&0&9\\3&4&8&7\\4&3&2&5\end{bmatrix}\qquad \text{Used}=\begin{bmatrix}1&1&0&1\\2&2&4&1\\0&5&0&1\\1&1&1&5\\2&0&2&2\end{bmatrix}$$

图4-11 银行家算法例子:Max和Used矩阵

表4-10 例子的安全序列查找的初始状态

进程	Max				Used				Need				Available				Finished
	A	B	C	D	A	B	C	D	A	B	C	D	2	1	2	2	
P_1	4	6	3	8	1	1	0	1	3	5	3	7					
P_2	3	3	5	2	2	2	4	1	1	1	1	1	4	3	6	3	1
P_3	6	6	0	9	0	5	0	1	6	1	0	8					
P_4	3	4	8	7	1	1	1	5	2	3	7	2					
P_5	4	3	2	5	2	0	2	2	2	3	0	3	6	3	8	5	2

4.死锁检测与恢复(Deadlock Detection and Recovery)

- 死锁检测的含义
- 资源分配图

资源分配图定义如下：

资源分配图G=(V，E)，其中V为结点集，E为边集。V=P∪R，这里，P是进程结点子集，R是资源结点子集；

边$e_{ij}=(p_i，r_j)$表示申请边，即进程p_i申请一个单位的资源r_j，

边$e_{ij}=(r_i，p_j)$表示分配边，即已经分配一个单位的资源r_i给进程p_j。

在画图时，用圆“○”表示进程结点，用正方形“□”表示资源结点，如果某类资源的数量有多个，则在对应的正方形“□”内用实心圆点表示其数量。

- 资源分配图的简化

边消除操作的条件：当前结点P_i不是孤立点，同时，不存在含有P_i结点的申请边，或者存在P_i结点的申请边但其资源申请都能得到满足。

例4-8 系统拥有的资源有R1、R2、R3、R4、R5和R6数量分别为2、1、1、1、1和2，当前进程有A、B、C、D和E，已知当前进程和资源的申请、分配关系，如表4-14所示。
请画出当前系统的资源分配图，并给出简化过程。

表4-14 系统当前的进程和资源的申请、分配关系

进程	分配得到的资源	申请的资源
A	2个单位的R1	1个单位的R3
B	1个单位的R5	1个单位的R1
C	1个单位的R2	1个单位的R4
D	1个单位的R3，1个单位的R6	1个单位的R5
E	1个单位的R4	1个单位的R6

- 死锁恢复(Deadlock Recovery)
  - 剥夺资源
  - 撤销进程
  - 重新启动系统

鸵鸟算法： 死锁的预防、避免，还是检测与恢复，都还没有找到简单、实用的有效方法。在考虑到死锁产生的可能性很小，与平常的机器硬件故障、系统死机等其他错误相比，死锁还是微不足道，因此，现有的操作系统通常不处理死锁问题，由程序员在设计开发软件过程根据实际应用自行处理，或由其他系统软件、开发平台等进行死锁检测。

例4-9 假设某一道程序运行时需要访问临界资源R，该程序可供多个用户同时运行，如果系统拥有资源R的数量为k，而程序申请使用资源R的数量为x(假定程序每次只申请1个，先后分x次申请)，有n个用户同时运行该程序。那么，k、x和n满足什么条件下，可以保证用户运行时不会产生进程死锁？