**Chapter 3 exercises**

16281262 麻锦涛

1. A multithreaded web server wishes to keep track of the number of requests it services (known as hits). Consider the two following strategies to prevent a race condition on the variable hits . The first strategy is to use a basic mutex lock when updating hits : （多线程Web服务器希望跟踪其服务的请求数（称为命中）。考虑以下两种策略来防止变量命中的竞态条件。第一个策略是在更新匹配时使用基本互斥锁：）

int hits**;**

mutex\_lock hit\_lock**;**

hit\_lock**.**acquire**();**

hits**++;**

hit\_lock**.**release**();**

A second strategy is to use an atomic integer: （第二种策略是使用原子整数操作）atomic\_thits;

atomic\_inc**(&**hits**);**

Explain which of these two strategies is more efficient.

**（解释互斥锁和原子整数操作这两种策略中的哪一种更有效。）**

答：**原子整数策略比互斥锁更有效**。从技术层面上讲，原子会在大多数平台上锁定内存总线。但是，有两个改进的细节在内存总线锁定期间暂停线程是不可能的，但是可以在互斥锁定期间挂起线程。这就是让你获得一个无锁保证的东西。所谓原子操作是指不会被线程调度机制打断的操作；这种操作一旦开始，就一直运行到结束，中间不会有任何 context switch （切换到另一个线程）原子操作是不可分割的，在执行完毕之前不会被任何其它任务或事件中断

2.Consider the code example for allocating and releasing processes shown in the following figure.

（请考虑用于分配和释放下图所示进程的代码示例。）

#define MAX\_PROCESSES 255 int number\_of\_processes **=**0**;**

/\* the implementation of fork() calls this function 实现调用fork()\*/

int allocate process**() {**

int new\_pid**;**

**if (**number\_of\_processes **==** MAX PROCESSES**)**

**return -**1**;**

**else {**

/\* allocate necessary process resources分配必要进程资源 \*/

**++**number\_of\_processes**;**

**}**

**}**

**return** new\_pid**;**

/\* the implementation of exit() calls this function调用函数exit() \*/

void release\_process**() {**

/\* release process resources释放进程 \*/

**--**number\_of\_processes**;**

**}**

1. Identify the race condition(s).（确定竞态条件）
2. Assume you have a mutex lock named mutex with the operations acquire() and release() .

Indicate where the locking needs to be placed to prevent the race condition(s).（假设您有一个名为mutex的互斥锁，其操作为acquire（）和release（）。指示需要锁定的位置以防止竞态条件。）

c. Could we replace the integer variable（我们可以替换整数变量吗？）

int number of processes = 0

with the atomic integer（用原子整数）

Atomic\_t number\_of\_processes = 0

to prevent the race condition(s)?（避免竞态条件）

答：a. 变量被两个进程同时读写，创建的进程将具有竞态条件

b. acquire()将在进入每个函数前被执行，release()在退出每个函数前将立刻执行

c. 在函数调用的前后加mutex，加了互斥锁操作就不会同时进行了，acquire()后需要等待release()才行，atomic是不能用的。因为在判断等于条件的时候无论是不是atomic都会进入条件判断。

1. Servers can be designed to limit the number of open connections. For example, a server may wish to have only N socket connections at any point in time. As soon as N connections are made, the server will not accept another incoming connection until an existing connection is released. Explain how semaphores can be used by a server to limit the number of concurrent connections.

答：信号量被初始化为允许的开放套接字连接的数量。当接受连接时，将调用acquire（）方法，当释放连接时，将调用release（）方法。如果系统达到允许的套接字连接数，则后续对acquire（）的调用将被阻塞，直到现有连接终止并调用release方法。

1. Show how to implement the wait() and signal() semaphore operations in multiprocessor environments using the test and set() instruction. The solution should exhibit minimal busy waiting.

答：这是我用于实现操作的伪代码：

int guard = 0;

int semaphore value = 0;

wait()

{ while (TestAndSet(&guard) == 1);

if (semaphore value == 0)

{ atomically add process to a queue of processes waiting for the semaphore and set guard to 0; }

else { semaphore value--; guard = 0; }

}

signal()

{ while (TestAndSet(&guard) == 1);

if (semaphore value == 0 && there is a process on the wait queue)

wake up the first process in the queue of waiting processes;

else semaphore value++; guard = 0; }

1. Explain the difference between preemptive and nonpreemptive scheduling.**（解释抢先和非抢先调度之间的区别）**

答：(1)非抢占式调度：一旦CPU被分配给一个进程，那么该进程会一直使用CPU直到进程终止或切换到等待状态时主动释放CPU。

（2）可抢占式调度：当一个进程在CPU上运行时，该程序的CPU使用权可以被另一个新进程剥夺，被抢占的程序的状态被保存在PCB中，并进入就绪队列，等待下一次运行。

如果一个系统采用的是非可抢占式调度，并采用优先级调度，那么优先级比较低的进程可能永远都在等待被执行，造成饥饿。这种情况就需要采用老化技术来防止饥饿。因此一个计算机中心最好不要采用严格的非抢占式调度。

1. Suppose that the following processes arrive for execution at the times indicated. Each process will run for the amount of time listed. In answering the questions, use nonpreemptive scheduling, and base all decisions on the information you have at the time the decision must be made.

Process Arrival Time Burst Time

P 1 0.0 8

P 2 0.4 4

P 3 1.0 1

1. What is the average turnaround time for these processes with the FCFS scheduling algorithm?
2. What is the average turnaround time for these processes with the SJF scheduling algorithm?
3. The SJF algorithm is supposed to improve performance, but notice that we chose to run process P 1 at time 0 because we did not know that two shorter processes would arrive soon. Compute what the average turnaround time will be if the CPU is left idle for the first 1 unit and then SJF scheduling is used. Remember that processes P 1 and P 2 are waiting during this idle time, so their waiting time may increase. This algorithm could be called future-knowledge scheduling.

答：a.执行顺序为：P1，P2，P3；平均周转时间为：(8+(12-0.4)+(13-1))/3=10.53

b.执行顺序为:P1，P3，P2；平均周转时间为：(8+(9-1)+(13-0.4))/3=9.53

c.执行顺序为P3，P2，P1；平均周转时间为：(14+(6-0.4)+(2-1))/3=6.87

1. What advantage is there in having different time-quantum sizes at different levels of a multilevel queueing system?

答：这里需要更频繁服务的进程（例如编辑器之类的交互式进程）可以在具有较小时间片的队列中。同时，不需要频繁服务的进程可以在具有较大时间片的队列中实现，需要较少的上下文切换来完成处理，从而来更有效地使用计算机。

8. Suppose that a scheduling algorithm (at the level of short-term CPU scheduling) favors those processes that have used the least processor time in the recent past. Why will this algorithm favor I/O-bound programs and yet not permanently starve CPU-bound programs?**（假设调度算法（短期CPU调度级别）支持那些在最近使用最少处理器时间的进程。为什么这个算法会支持受I / O约束的程序，却又不会永久地饿死CPU约束的程序？）**

答：它有利于I / O约束程序，因为它们的CPU突发请求相对比较短;但是，受CPU限制的程序不会饿死，因为受I / O限制的程序会相对频繁地放弃CPU以进行I / O操作。

### 9.Why is it important for the scheduler to distinguish I/O -bound programs from CPU -bound programs?（[为什么对调度程序而言,区分CPU约束程序和I/O约束程序很重要?](http://www.baidu.com/link?url=G2bI7uwTTsd5eLSlwBPjtfNTZvmbSuSA7_PQTj_lTApOLUOqirV6T2Il5nAunDk9uToyVYe4be16QXMcjfclM_)）

答：**I/O**约束程序具有在执行**I/O**之前仅执行少量计算的属性。此类程序通常不会耗尽其整个CPU量。另一方面，CPU约束程序使用它们的整个量程而不执行任何阻塞I/O的操作。因此，通过为I/O约束程序提供更高优先级并允许它们在CPU约束程序之前执行，可以更好地利用计算机资源。CPU的成功调度依赖于进程的如下属性：进程执行由CPU执行和I/O等待周期组成，进程在这两个状态之间切换。进程执行从CPU区间开始，在这之后是I/O区间，接着是另一个CPU区间，如此进行下去；最终，最后的CPU区间通过系统请求终止执行。这些CPU区间的长度呈现出指数或超指数形式的频率曲线，具有大量短CPU区间和少量长CPU区间。I/O约束程序通常具有很多短CPU区间。CPU约束程序可能有少量的长CPU区间。所以区分出CPU约束程序和I/O约束程序能够使调度程序更好的执行和选择合适的CPU调度算法。

10.Discuss how the following pairs of scheduling criteria conflict in certain settings.

1. CPU utilization and response time
2. Average turnaround time and maximum waiting time
3. I/O device utilization and CPU utilization

答：a. CPU使用率和响应时间：如果与上下文切换相关的开销最小化，则CPU使用率会增加。可以通过不频繁地执行上下文切换来降低上下文切换开销。但是，这可能会增加进程的响应时间。

b.平均周转时间和最长等待时间：通过首先执行最短任务来最小化平均周转时间。但是，这样的调度策略可能会使长时间运行的任务匮乏，从而增加了他们的等待时间。

c. I / O设备利用率和CPU使用率：通过运行长时间运行的CPU约束任务而不执行上下文切换，可以最大限度地提高CPU使用率。 I / O设备利用率通过在I / O约束任务准备好运行时立即调度来最大化，从而导致上下文切换的开销。

11. Consider the following set of processes, with the length of the CPU burst given in milliseconds:

Process Burst Time Priority

P1 2 2

P2 1 1

P3 8 4

P4 4 2

P5 5 3

The processes are assumed to have arrived in the order P1 , P2 , P3 , P4 , P5 , all at time 0.

1. Draw four Gantt charts that illustrate the execution of these processes using the following scheduling algorithms: FCFS , SJF , nonpreemptive priority (a larger priority number implies a higher priority), and RR (quantum = 2).
2. What is the turnaround time of each process for each of the scheduling algorithms in part a?
3. What is the waiting time of each process for each of these scheduling algorithms?
4. Which of the algorithms results in the minimum average waiting time (over all processes)?

（哪种算法导致最小平均等待时间（在所有过程中）？）

**答：a）**

P1

P2

P3

P4

P5

0

10

11

13

14

19

**FCFS**

P1

P2

P3

P4

P5

0

4

1

9

2

19

**非抢占式SJF**

P1

P2

P3

P4

P5

0

6

1

16

18

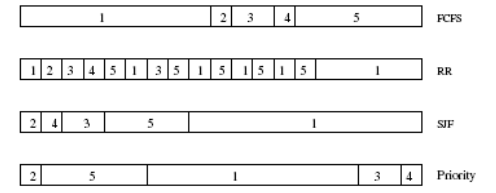
19

**非抢占式优先级**

P1 P2 P3 P4 P5 P1 P3 P5 P1 P5 P1 P5 P1 P5 P1 P1 P1 P1 P1

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19

**RR（时间片=1）**

****

**b）周转时间：**

**FCFS：**

P1=10；P2=11；P3=13；P4=14；P5=19；

**SJF：**

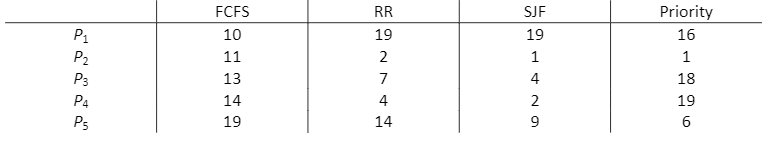
P1=19；P2=1； P3=4； P4=2； P5=9；

**非抢占式优先级：**

P1=16；P2=1； P3=18； P4=19； P5=6；

**RR：**

P1=19；P2=2； P3=7； P4=4； P5=14；



c）**等待时间：**

**FCFS：**

P1=0；P2=10；P3=11；P4=13；P5=14；

**SJF：**

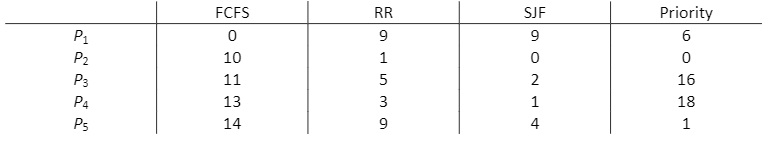
P1=9；P2=0； P3=2； P4=1； P5=4；

**非抢占式优先级：**

P1=6；P2=0； P3=16； P4=18； P5=1；

**RR：**

P1=9；P2=1； P3=5； P4=3； P5=9；



**d）平均等待时间：**

**FCFS：**

(0+10+11+13+14)/5=9.6

**SJF：**

(9+0+2+1+4)/5=3.2

**非抢占式优先级：**

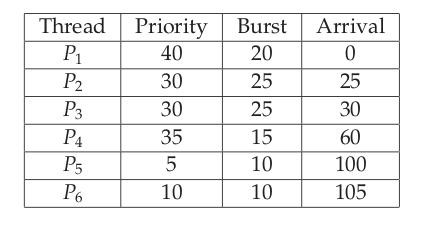
(6+0+16+18+1)/4=10.25

**RR：**

(9+1+5+3+9)/4=6.75

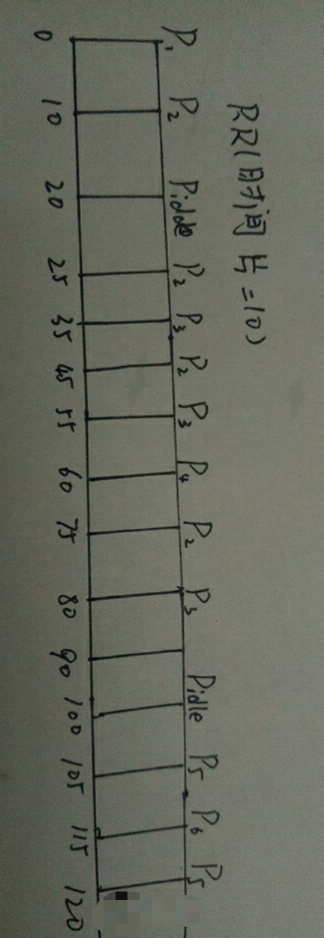
所以，SJF算法的平均等待时间最少。

12. The following processes are being scheduled using a preemptive, round-robin scheduling algorithm. Each process is assigned a numerical priority, with a higher number indicating a higher relative priority. In addition to the processes listed below, the system also has an idle task (which consumes no CPU resources and is identified as P idle ). This task has priority 0 and is scheduled whenever the system has no other available processes to run. The length of a time quantum is 10 units. If a process is preempted by a higher-priority process, the preempted process is placed at the end of the queue.**(轮转法，且时间片的长度为10个单位)**



1. Show the scheduling order of the processes using a Gantt chart.（甘特图）
2. What is the turnaround time for each process?（周转时间）
3. What is the waiting time for each process?（等待时间）
4. What is the CPU utilization rate?（CPU使用率）

答： a. 手画图如下（比较丑）



b. p1: 20-0 =20, p2: 80-25 = 55, p3: 90 - 30 = 60, p4: 75-60 = 15, p5: 120-100 = 20, p6: 115-105 = 10

c. p1: 0, p2: 40, p3: 35, p4: 0, p5: 10, p6: 0

d. 105/120 = 87.5 %

13. Which of the following scheduling algorithms could result in starvation?

1. First-come, first-served**（先到先服务）**
2. Shortest job first**（最短时间优先）**
3. Round robin**（轮转法）**
4. Priority**（优先级调度算法）**

答：这里我选择b和d

短时间优先和优先级调度算法可能导致饥饿。我这里的分析如下：

a.先到先服务（FCFS）：

FCFS调度算法是一种最简单的调度算法，该调度算法既可以用于作业调度也可以用于进程调度。在作业调度中，算法每次从后备作业队列中选择最先进入该队列的一个或几个作业，将它们调入内存，分配必要的资源，创建进程并放入就绪队列。

在进程调度中，FCFS调度算法每次从就绪队列中选择最先进入该队列的进程，将处理机分配给它，使之投入运行，直到完成或因某种原因而阻塞时才释放处理机。

FCFS调度算法属于不可剥夺算法。从表面上看，它对所有作业都是公平的，但若一个长作业先到达系统，就会使后面许多短作业等待很长时间，因此它不能作为分时系统和实时系统的主要调度策略。但它常被结合在其他调度策略中使用。例如，在使用优先级作为调度策略的系统中，往往对多个具有相同优先级的进程按FCFS原则处理。

FCFS调度算法的特点是算法简单，但效率低；对长作业比较有利，但对短作业不利（相对SJF和高响应比）；有利于CPU繁忙型作业，而不利于I/O繁忙型作业。

所以先来先服务算法不会引起饥饿现象。

b.最短作业优先（SJF）：

短作业（进程）优先调度算法是指对短作业（进程）优先调度的算法。短作业优先(SJF)调度算法是从后备队列中选择一个或若干个估计运行时间最短的作业，将它们调入内存运行。而短进程优先(SPF)调度算法，则是从就绪队列中选择一个估计运行时间最短的进程，将处理机分配给它，使之立即执行，直到完成或发生某事件而阻塞时，才释放处理机。

SJF调度算法也存在不容忽视的缺点：

该算法对长作业不利， SJF调度算法中长作业的周转时间会增加。更严重的是，如果有一长作业进入系统的后备队列，由于调度程序总是优先调度那些 (即使是后进来的）短作业，将导致长作业长期不被调度。该算法完全未考虑作业的紧迫程度，因而不能保证紧迫性作业会被及时处理。由于作业的长短只是根据用户所提供的估计执行时间而定的，而用户又可能会有意或无意地缩短其作业的估计运行时间，致使该算法不一定能真正做到短作业优先调度。

由此可见，SJF算法会导致长作业一直在等待的情况，会导致饥饿现象。

c.轮转法（RR）：

时间片轮转调度算法主要适用于分时系统。在这种算法中，系统将所有就绪进程按到达时间的先后次序排成一个队列，进程调度程序总是选择就绪队列中第一个进程执行，即先来先服务的原则，但仅能运行一个时间片，如100ms。在使用完一个时间片后，即使进程并未完成其运行，它也必须释放出（被剥夺）处理机给下一个就绪的进程，而被剥夺的进程返回到就绪队列的末尾重新排队，等候再次运行。

在时间片轮转调度算法中，时间片的大小对系统性能的影响很大。如果时间片足够大，以至于所有进程都能在一个时间片内执行完毕，则时间片轮转调度算法就退化为先来先服务调度算法。如果时间片很小，那么处理机将在进程间过于频繁切换，使处理机的开销增大，而真正用于运行用户进程的时间将减少。因此时间片的大小应选择适当。

时间片的长短通常由以下因素确定：系统的响应时间、就绪队列中的进程数目和系统的处理能力。

轮转法调度中各个作业轮换执行，所以不会导致饥饿现象。

d.优先级：

优先级调度算法又称优先权调度算法，该算法既可以用于作业调度，也可以用于进程调度，该算法中的优先级用于描述作业运行的紧迫程度。

在作业调度中，优先级调度算法每次从后备作业队列中选择优先级最髙的一个或几个作业，将它们调入内存，分配必要的资源，创建进程并放入就绪队列。在进程调度中，优先级调度算法每次从就绪队列中选择优先级最高的进程，将处理机分配给它，使之投入运行。

根据新的更高优先级进程能否抢占正在执行的进程，可将该调度算法分为：

非剥夺式优先级调度算法。当某一个进程正在处理机上运行时，即使有某个更为重要或紧迫的进程进入就绪队列，仍然让正在运行的进程继续运行，直到由于其自身的原因而主动让出处理机时（任务完成或等待事件），才把处理机分配给更为重要或紧迫的进程。

剥夺式优先级调度算法。当一个进程正在处理机上运行时，若有某个更为重要或紧迫的进程进入就绪队列，则立即暂停正在运行的进程，将处理机分配给更重要或紧迫的进程。

而根据进程创建后其优先级是否可以改变，可以将进程优先级分为以下两种：

静态优先级。优先级是在创建进程时确定的，且在进程的整个运行期间保持不变。确定静态优先级的主要依据有进程类型、进程对资源的要求、用户要求。

动态优先级。在进程运行过程中，根据进程情况的变化动态调整优先级。动态调整优先级的主要依据为进程占有CPU时间的长短、就绪进程等待CPU时间的长短。

优先级调度会导致优先级高的作业先执行，而优先级低的作业可能会一直等待，所以优先级算法会导致饥饿现象。

综上所述，b.SJF算法和d.优先级调度算法会导致饥饿现象。

14. Consider a system running ten I/O -bound tasks and one CPU -bound task. Assume that the I/O -bound tasks issue an I/O operation once for every millisecond of CPU computing and that each I/O operation takes 10 milliseconds to complete. Also assume that the context-switching overhead is 0.1 millisecond and that all processes are long-running tasks. Describe the CPU utilization for a round-robin scheduler when:

a. The time quantum is 1 millisecond

b. The time quantum is 10 milliseconds

答：a.时间量是1毫秒：无论调度哪个进程，调度程序都会为每个上下文切换产生0.1毫秒的上下文切换成本。导致CPU使用率为1 / 1.1 \* 100 = 91％。

b.时间量是10毫秒：I / O约束任务在耗尽仅1毫秒的时间量后会产生上下文切换。因此，循环所有进程所需的时间是10 \* 1.1 + 10.1（因为每个I / O约束任务执行1毫秒然后引发上下文切换任务，而CPU约束任务执行10毫秒之后才会产生上下文切换）。因此CPU使用率为20 / 21.1 \* 100 = 94％。

15. Explain the differences in how much the following scheduling algorithms discriminate in favor of short processes: （解释以下调度算法在有利于短进程时的差异：）

a.FCFS（先来先服务调度算法）

b.RR（[轮转调度算法](http://www.baidu.com/link?url=Sw5Ra8sTpquTXgcVhwK90vIXjXo23O41nIDnu3v2U39lf82LxJElCJE3x7zMDJeyGiwjPop4sCFqWGTD67y-_ZbS-peusR4WyrlxH97GxIK)）

c.Multilevel feedback queues（多级反馈队列调度算法）

答：

a.先来先服务调度算法-按照作业进入系统的先后次序来挑选作业，先进入系统的作业优先被挑选。效率不高，只顾及作业等候时间，没考虑作业要求服务时间的长短。因此优待了长作业而不利于短作业；有利于CPU繁忙型作业，而不利于I/O繁忙型作业。

b. [轮转调度算法](http://www.baidu.com/link?url=Sw5Ra8sTpquTXgcVhwK90vIXjXo23O41nIDnu3v2U39lf82LxJElCJE3x7zMDJeyGiwjPop4sCFqWGTD67y-_ZbS-peusR4WyrlxH97GxIK)-平等对待所有工作（给它们相同时间的CPU突发），这样短作业就能更快地离开系统，因为它们将首先完成。

c.多级反馈队列的工作类似于[轮转调度算法](http://www.baidu.com/link?url=Sw5Ra8sTpquTXgcVhwK90vIXjXo23O41nIDnu3v2U39lf82LxJElCJE3x7zMDJeyGiwjPop4sCFqWGTD67y-_ZbS-peusR4WyrlxH97GxIK) - 它们有利于短作业。

1. Suppose that a system is in an unsafe state. Show that it is possible for the processes to complete their execution without entering a deadlocked state.

（假设一个系统是在一个不安全的状态。说明这可能是一个进程完成地执行而没有进入一个死锁状态。）

答：一个不安全的状态不一定导致死锁,它仅仅意味着我们不能保证死锁不会再次发生。因此,可能是一个系统在一个不安全的状态还可以让所有的进程完成而没有死锁发生。考虑到这样的情况，一个系统有12个资源要分配给进程P0，P1和P2。资源按照以下政策分配:

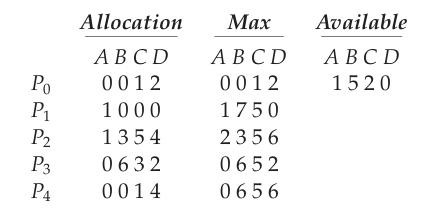
当前的最大需求

*P*0 10 5 5

*P*1 4 2 2

*P*2 9 3 6

1. Consider the following snapshot of a system:



Answer the following questions using the banker’s algorithm: a. What is the content of the matrix Need?

1. Is the system in a safe state?
2. If a request from process P1 arrives for (0,4,2,0), can the request be granted immediately?

答：

a. Need矩阵的内容是P0（0 0 0 0） P1（0 7 5 0） P2（1 0 0 2） P3（0 0 2 0） P4（0 6 4 0）。

b. 系统处于安全状态，因为Available矩阵等于（1 5 2 0），进程P0和P3都可以运行，当进程P3运行完时，它释放它的资源，而允许其它进程运行。

c. 可以被满足，满足以后，Available矩阵等于（1 1 0 0），当以次序P0，P2, P3, P1 ,P4运行时候，可以完成运行。

1. Consider a system consisting of four resources of the same type that are shared by three processes, each of which needs at most two resources. Show that the system is deadlock free.

（考虑一个由4种相同资源、3个共享进程，每个需要2个该资源的系统，试证明这个系统不会出现死锁）

证明：根据“抽屉原理”，将4个资源分配给3个进程，则说明至少有一个进程分配到2个资源，那么该进程就得以运行；运行结束时，该进程会释放已分配的2个资源给系统里剩下的两个进程，这时4个资源分配给2个进程，因此系统不会出现死锁。

1. Consider a system consisting of m resources of the same type being shared by n processes. A process can request or release only one resource at a time. Show that the system is deadlock free if the following two conditions hold:
2. The maximum need of each process is between one resource and m resources.
3. The sum of all maximum needs is less than m + n.

证明：使用Section7.6.2的术语,可以有：

a.<m+n

b. ≥ 1 for all *i*

Proof: = − 

如果存在死锁状态，则有:

c. = *m*

Use a. to get:+ = *< m* + *n*

Use c. to get:+ *m < m* + *n*

Rewrite to get:**

这意味着存在一个Pi的进程，其=0.如果 >=1，那么Pi进程至少有一个资源可以释放。从而系统就不会进入死锁状态。