Chapter 1 Introduction

Homework (P.36)

1.10

1.11

1.13

1.17

1.10 What are the differences between a trap and interrupt? What is the use of each function?

Answer:

- **An interrupt is a hardware-generated change-of-flow within the system. An interrupt handler is summoned to deal with the cause of the interrupt; control is then returned to the interrupted context and instruction.**
- **A trap is a software-generated interrupt.**

- **An interrupt can be used to signal the completion of an I/O to obviate the need for device polling.**
- **A trap can be used to call operating system routines or to catch arithmetic errors.**

1.11

a) Host writes a DMA command block into memory, CPU writes the address of this command block to the DMA controller, DMA controller operates the memory bus directly, placing address on the bus to perform transfers

b) DMA interrupts CPU to signal transfer completion

c) yes. Because cycle stealing（周期挪用） is used to transfer data on the system bus, so the instruction cycle is suspended so data can be transferred, the execution of the user programs is slowdown.

1.13 Give two reasons why caches are useful. What problems do they solve? What problems do they cause? If a cache can be made as large as the device for which it is caching (for instance, a cache as large as a disk), why not make it that large and eliminate the device?

Answer:

- **Caches are useful when two or more components need to exchange data, and the components perform transfers at differing speeds. Caches solve the transfer problem by providing a buffer of intermediate speed between the components. If the fast device finds the data it needs in the cache, it need not wait for the slower device.**
- **The data in the cache must be kept consistent with the data in the components. If a component has a data value change, and the datum is also in the cache, the cache must also be updated.**

1.13 (cont.)

This is especially a problem on multiprocessor systems where more than one process may be accessing a datum.

A component may be eliminated by an equal-sized cache, but only if:

(a) the cache and the component have equivalent state-saving capacity (that is, if the component retains its data when electricity is removed, the cache must retain data as well), and

(b) the cache is affordable, because faster storage tends to be more expensive.

1.17 Define the essential properties of the following types of operating systems.

a. Batch
b. Interactive
c. Time sharing
d. Real time
e. Network
f. Parallel
g. Distributed
h. Clustered
i. Handheld

1.17 Answer:

a. Batch: Jobs with similar needs are batched together and run through the computer as a group by an operator or automatic job sequencer. Performance is increased by attempting to keep CPU and I/O devices busy at all times through buffering, off-line operation, spooling, and multiprogramming. Batch is good for executing large jobs that need little interaction; it can be submitted and picked up later.

b. Interactive: This system is composed of many short transactions where the results of the next transaction may be unpredictable. Response time needs to be short (seconds) since the user submits and wait for the results.

c. Time sharing: This systems uses CPU scheduling and multiprogramming to provide economical interactive use of a system. The CPU switches rapidly from one user to another. Instead of having a job defined by spooled card images, each program reads its next control card from the terminal, and output is normally printed immediately to the screen.

d. Real time: Often used in a dedicated application, this system reads information from sensors and must respond within a fixed amount of time to ensure correct performance.

e. Network: In the simplest terms, is a communication path between two or more systems. Networks vary by the protocols used, the distances between nodes, and the transport media.

f. Parallel: Such systems have more than one processor in close communication, sharing the computer bus, the clock, and sometimes memory and peripheral devices.

g. Distributed: This system distributes computation among several physical processors. The processors do not share memory or a clock. Instead, each processor has its own local memory. They communicate with each other through various communication lines, such as a high-speed bus or telephone line.

h. Clustered: Clustered systems gather together multiple CPUs to accomplish computational work. Clustered systems coupled together.

i. Handheld: Handheld systems include personal digital assistants (POAs), such as Palm-Pilots or cellular telephones with connectivity to a network such as the Internet. Most handheld devices have a small amount of memory, include slow Processors, and feature small display screen.

Chapter 2 Operating-System Structures Homework (Page 73)

2.1

2.3

2.7

2.8

2.15

- 2.1 (P.40)
  - Helpful to the user
  - Helpful to ensuring the efficient operation of the system
- 2.3(P.47)
  - Register
  - Parameters are stored in a block in memory, and the address of the block is passed as a parameter in a register
  - stack

2.7 What is the purpose of the command interpreter? Why is it usually separate from the kernel?

Answer: (P.41-42)

It reads commands from the user or from a file of commands and executes them, usually by turning them into one or more system calls. It is usually not part of the kernel since the command interpreter is subject to changes.

2.8 (P.54)

– **Message passing**
  - **Useful for exchanging smaller amount of data**
  - **Processes can be running in different computer**

– **shared memory**
  - **Allows maximum speed and convenience of communication**
  - **Processes must have shared memory**

2.15(P.68)

Chapter 3 Processes Homework (page 116)

3.1

3.2

3.4

3.1 Describe the differences among short-term, medium-term, and long-term scheduling.

Answer:

Short-term (CPU scheduler)—selects from processes in memory those processes that are ready to execute and allocates the CPU to them.

Medium-term—used especially with time-sharing systems as an intermediate scheduling level. A swapping scheme is implemented to remove partially run programs from memory and reinstate them later to continue where they left off.

Long-term (job scheduler)—determines which jobs are brought into memory for processing.

The primary difference is in the frequency of their execution. The short-term must select a new process quite often. Long-term is used much less often since it handles placing jobs in the system and may wait a while for a job to finish before it admits another one.

3.2 Describe the actions a kernel takes to context switch between processes.

Answer:

In general, the operating system must save the state of the currently running process and restore the state of the process scheduled to be run next. Saving the state of a process typically includes the values of all the CPU registers in addition to memory allocation. Context switches must also perform many architecture-specific operations, including flushing data and instruction caches.

- **3.4 (P.92)**
  - **5**

Chapter 4 Threads
Homework (page 146)

4.1 4.2 4.3

4.1 (P.127-128)

Multithreaded web browser. user can interaction in one thread, while an image was being loaded in another thread.

Word processing, editing thread and spelling checking.

4.2 Describe the actions taken by a thread library to context switch between user-level threads.

Answer:

Context switching between user threads is quite similar to switching between kernel threads, although it is dependent on the threads library and how it maps user threads to kernel threads. In general, context switching between user threads involves taking a user threads of its LWP and replacing it with another threads. This act typically involves saving and restoring the state of the registers.

4.3 on multi-processor system

Chapter 5 CPU Scheduling Homework (page 186)

5.4 5.5

Thinking:

1. Define the difference between preemptive and nonpreemptive scheduling.
2. Is a nonpreemptive scheduling algorithm a good choice for an interactive system? Briefly, why?
3. 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?

5.4 Consider the right set of processes, with the length of the CPU-burst time given in milliseconds:

Process	Burst Time	Priority
P1	10	3
P2	1	1
P3	2	3
P4	1	4
p5	5	2

The processes are assumed to have arrived in the order *P*1, *P*2, *P*3, *P*4, *P*5, all at time 0.

a. Draw four Gantt charts illustrating the execution of these processes using FCFS, SJF, a nonpreemptive priority (a smaller priority number implies a higher priority), and RR (quantum = 1) scheduling.

b. What is the turnaround time of each process for each of the scheduling algorithms in part a?

c. What is the waiting time of each process for each of the scheduling algorithms in part a?

d. Which of the schedules in part a results in the minimal average waiting time (over all processes)?

5.4 Answer:

a The four Gantt charts are:

FCFS	1	1	1	1	1	1	1	1	1	2	3	3	4	5	5	5	5	5	5
RR	1	2	3	4	5	1	3	5	1	5	1	5	1	5	1	1	1	1	1
SJF	2	4	3	3	5	5	5	5	1	1	1	1	1	1	1	1	1	1	1
Priority	2	5	5	5	5	5	1	1	1	1	1	1	1	1	1	1	3	3	4

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

b Turnaround time:

	FCFS	RR	SJF	Priority
P1	10	19	19	16
P2	11	2	1	1
P3	13	7	4	18
P4	14	4	2	19
P5	19	14	9	6

c Waiting time (turnaround time minus burst time):

	FCFS	RR	SJF	Priority
P1	0	9	9	6
P2	10	1	0	0
P3	11	5	2	16
P4	13	3	1	18
P5	14	9	4	1

d Short Job First.

- **5.5**
  - **b**
  - **d**

1 Define the difference between preemptive and nonpreemptive scheduling..

Answer:

- Preemptive scheduling allows a process to be interrupted in the midst of its execution, taking the CPU away and allocating it to another process.
- Nonpreemptive scheduling ensures that a process relinquishes control of the CPU only when it finishes with its current CPU burst.

2 Is a nonpreemptive scheduling algorithm a good choice for an interactive system? Briefly, why?

Answer:

No.

Once a process gains control of CPU, it retains control until it blocks or terminates.

A process could execute for an extended period of time doing neither. Other processes on the system would not be able to execute, producing unacceptable response time.

4. Considering a real-time system, in which there are 4 real-time processes P1, P2, P3 and P4 that are aimed to react to 4 critical environmental events e1, e2, e3 and e4 in time respectively.

The arrival time of each event ei, 1≤i≤4, (that is, the arrival time of the process Pi), the length of the CPU burst time of each process Pi, and the deadline for each event ei are given below. Here, the deadline for ei is defined as the absolute time point before which the process Pi must be completed.

The priority for each event ei (also for Pi) is also given, and a smaller priority number implies a higher priority.

Events	Process	Arrival Time	Burst Time	Priorities	Deadline
e1	P1	0.00	4.00	3	7.00
e2	P2	3.00	2.00	1	5.50
e3	P3	4.00	2.00	4	12.01
e4	P4	6.00	4.00	2	11.00

(1) Suppose that priority-based preemptive scheduling is employed

Draw a Gantt chart illustrating the execution of these processes

What are the average waiting time and the average turnaround time

Which event will be treated with in time, that is, the process reacting to this event will be completed before its deadline?

(2) Suppose that FCFS scheduling is employed

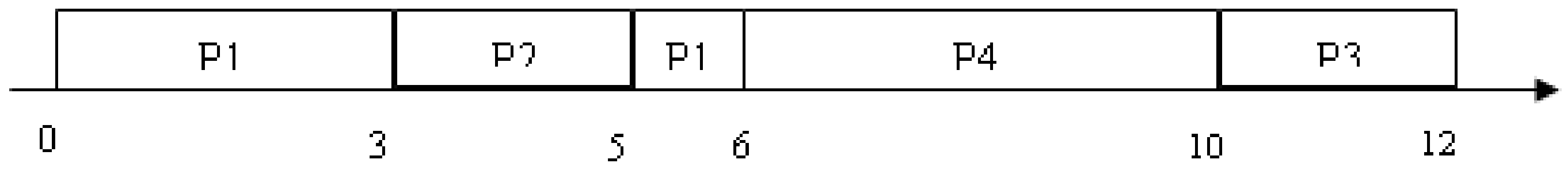
Draw a Gantt chart illustrating the execution of these processes

What are the average waiting time and the average turnaround time

Which event will be treated with in time?

4. Answers:

(1) 甘特图如下

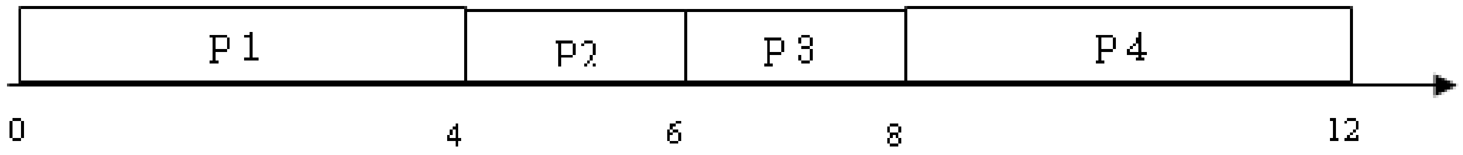


平均等待时间=[(5—3) + (3—3) + (10—4) + (6—6)] /4 = 2

平均周转时间=[(6—0) + (5—3) + (12—4) + (10—6)] /4 = 5

根据各进程的完成时间点和所对应的事件的deadline可知，全部4 个事件均可得到及时响应.

(2) 甘特图如下



平均等待时间=[(0—0) + (4—3) + (6—4) + (8—6)] /4 = 1.25

平均周转时间=[(4—0) + (6—3) + (8—4) + (12—6)] /4 = 4.25

根据各进程的完成时间点和所对应的事件的deadline可知，事件e1和e3可得到及时响应.

Chapter 6 Process Synchronization Homework(P.231)

6.1 6.11 6.16

- 6.1
  - 互斥：turn=i, Pi enter into its critical section
  - Progress: i不在临界区，则
    - 1) flag[i]=false 或
    - 2) Flag[i]=true , but turn=j

    - Pj请求，flag[j]=true,
      - 1) 则Pj 进入
      - 2) Pj可进入，Pi等待Pj退出后，将turn=i后，进入
  - Bounded waiting：i在临界区中，flag[i]=true turn=I
    - Pj 等待，Pi退出后，设置turn=j，则Pj进入。

6.11 Answer:

```
Semaphore cuthair :=0;
Semaphore waiting :=0;
Semaphore countMutex :=1
int count :=0;
Void barber ()
{
   while (1)
   {
   wait (cuthair);
   givehaircut ();
   }
}
```

```
Void customer ()
{
   wait (countMutex);
   if (count==n+1) {
   // n chairs plus the barber
   chair
     signal (countMutex);
     exit(); }
   count=count+1;
   if (count > 1) {
     // take a chair
     signal (countMutex);
     wait(waiting);  }
  else   {
     signal (countMutex);
     signal (cutHair); }
     //------------------
     receiveHaircut();
     //-------------------

     wait(countMutex);
     count=count-1;
     if (count>0)
        signal(waiting);
     signal(countMutex);
}
```

- **6.16**
  - In monitor, The x.signal() operation resumes exactly one suspended process. If no process is suspended, then the signal operation has no effect.
  - For semaphore, signal() operation removes one process from the list of waiting processes and awakens that process. If there is no process in the list of waiting processes, the value of the semaphore increase 1.

Exercise 4

- 考虑下面各图中的3个进程 P1, P2, 和 P3.

(1)

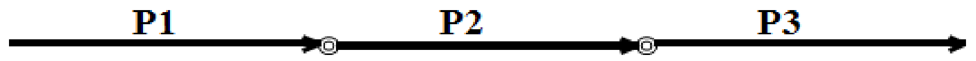


图 (a)

(2)

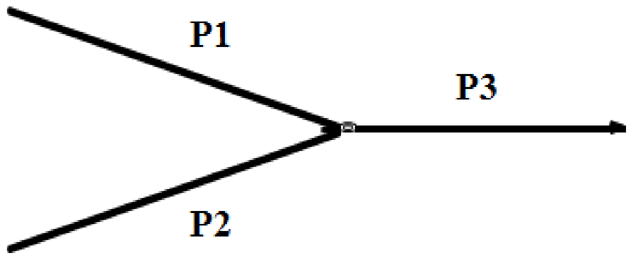


图 (b)

(3)

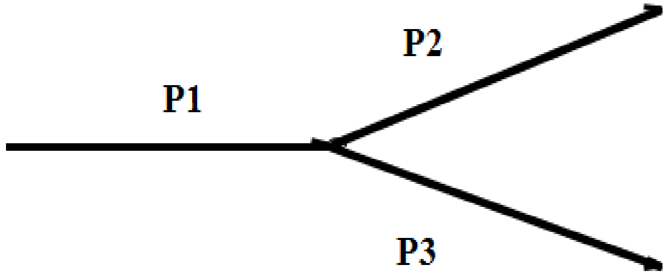


图 (c)

试定义信号量，通过对这些信号量的wait/signal操作实现进程P1，P2和P3 的同步，假设P1、P2和P3各自的动作如下：

```
P1:         P2:         P3:
begin       begin       begin
  S1;         S3;         S5;
  S2;         S4;         S6;
end         end         end
```

Answer:

(1)

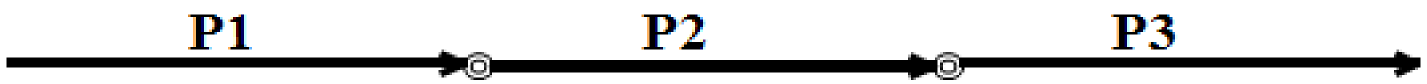


图 (a)

Semaphore: S12=0, S23=0

```
P1:
 begin
    S1;
    S2;
    signal(S12);
 end

P2:
 begin
    wait(S12);
    S3;
    S4;
    signal(S23);
 end

P3:
 begin
    wait(S23);
    S5;
    S6;
 end
```

Answer:

(2)

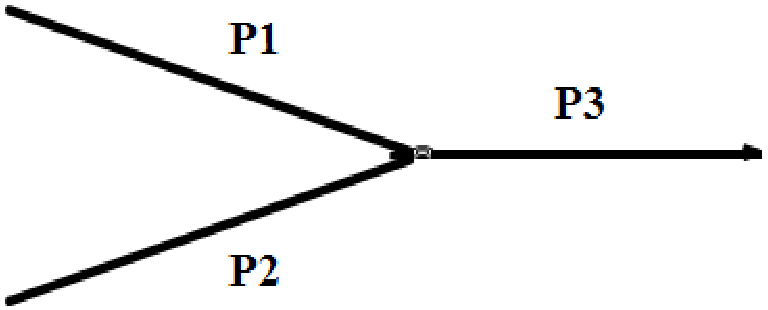


图 (b)

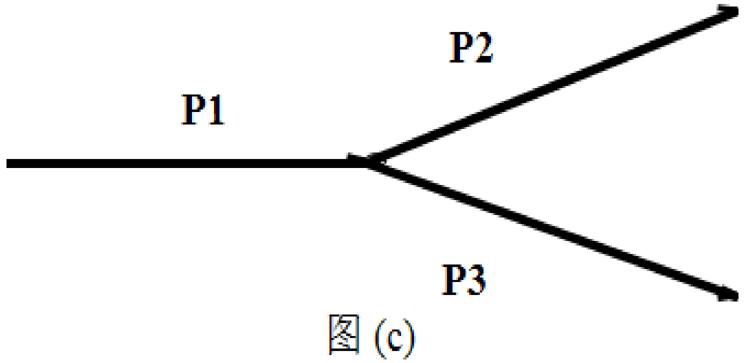
Semaphore: S13=0, S23=0

```
P1:
 begin
    S1;
    S2;
    signal(S13);
 end
```

```
P2:
 begin
    S3;
    S4;
    signal(S23);
 end
```

```
P3:
 begin
    wait(S13);
    wait(S23);
    S5;
    S6;
 end
```

Answer:

(3) 

Semaphore: S12=0, S13=0

```
P1:
 begin
    S1;
    S2;
    signal(S12);
    signal(S13);
 end

P2:
 begin
    wait(S12);
    S3;
    S4;
 end

P3:
 begin
    wait(S13);
    S5;
    S6;
 end
```

Ch6 exercise 5：

某应用有三个进程，C、M、和P。

- 数据采集进程 C 把采集到的数据放到buf1中。
- 数据处理进程 M 从buf1中取数据进行处理，并把结果放入buf2中。
- 数据输出进程 P 从buf2中取出结果打印输出。

考虑以下两种情况，用信号量机制实现进程C、M、和P之间的同步，分别给出进程C、M、和P的代码结构

(1) Buf1、buf2都只能保存一个数据

(2) Buf1可以保存m个数据、buf2可以保存n个数据

（假设各进程对缓冲区的操作需要互斥）

(1) buf1、buf2都只能存放一个数据

Semaphore： cm=1; mc=0; mp=1; pm=0;

```
Process C
{
  While (1) {
    采集数据;
     wait(cm);
    数据存入 buf1;
     signal(mc)
  }
}
```

```
Process M
{
  While (1) {
     wait(mc);
    从buf1中取出数据;
     signal(cm);
    对数据进行处理;
     wait(mp);
    处理结果存入buf2;
     signal(pm)
  }
}
```

```
Process P
{
  While (1) {
     wait(pm);
    从buf2中取出结果;
    signal(mp);
    打印结果
  }
}
```

(2) buf1可以存放m个,buf2能存放n个

Semaphore： empty1=m; full1=0; empty2=n; full2=0; mutex1=1; mutex2=1

```
Process C
{
 While (1) {
   采集数据;
   wait(empty1);
   wait(mutex1);
   把数据存入Buf1;
   signal(mutex1);
   signal(full1)
 }
}
```

```
Process M
{
 While (1) {
   wait(full1);
   wait(mutex1);
   从Buf1中取出数据;
   signal(mutex1);
   signal(empty1);
   对数据进行处理;
   wait(empty2);
   wait(mutex2);
   把处理结果存入Buf2;
   signal(mutex2);
   signal(full2)
 }
}
```

```
Process P
{
 While (1) {
   wait(full2);
   wait(mutex2);
   从Buf2中取出结果;
   signal(mutex2);
   signal(empty2);
   打印结果
 }
}
```

Ch6 exercise 6

- 某系统中有m个进程并发执行，分为A、B两组，他们共享文件F，A组进程对文件F进行写操作，B组进程对文件F进行只读操作。写操作需要互斥进行，读操作可以同时发生，即当有B组的某进程正在对文件F进行读操作时，若没有A组的进程提出写请求，则B组的其他进程可以同时对文件F进行读操作，如果有A组的进程提出写请求，则阻止B组的其他进程对文件F进行读操作的后续请求，等当前正在读文件的B组进程全部执行完毕，则立即让A组提出写请求的进程执行对文件F的写操作。

（1）定义信号量及变量，给出其初值，说明其作用

（2）写出A组和B组进程的代码结构

Answer 1:

```
int reader_count=0;
Semaphore
    mutex=1;
    RW_mutex=1;
    W_first=1;

writers(){
       wait(W_first);
       wait(RW_mutex);
       writing file F;
       signal(RW_mutex);
       signal(W_first);
}
```

```
Readers(){
       wait(W_first);
       wait(mutex);
       if (reader_count==0
           wait(RW_mutex);
       reader_count++;
       signal(mutex);
       signal(W_first);
       reading file F;
       wait(mutex);
       reader_count--;
       if (reader_count==0
           signal(RW_mutex);
       signal(mutex);
}
```

Answer 2:

```
Int  R-num=0;     W_num=0;
Semaphore  wrt=1; W_first=1;     mutex_w=1;     mutex_R=1;
```

```
Writer(){
    Wait(mutex_w);
    W_num++;
    If (W_num==1)  wait(W_first);
    signal(mutex_w);
    Wait(wrt);
    写文件;
    signal(wrt);
    Wait(mutex_w);
    W_num--;
    If (W-num==0) signal(W_first);
    signal(mutex_w);
}
```

```
Reader(){
    Wait(W_first);
    Wait(mutex_R);
     R_num++;
     if (R_num==1)  wait(wrt);
     signal(mutex_R);
     signal(W_first);
     读文件;
     Wait(mutex_R);
     R_num--;
     If (R_num==0)  signal(wrt);
     signal(mutex_R);
}
```

Chapter 7 Deadlocks
Homework (page 268)

7.1

7.5

7.6

7.7

7.11

7.1 Consider the traffic deadlock depicted in Figure 8.11.

a. Show that the four necessary conditions for deadlock indeed hold in this example.

b. State a simple rule that will avoid deadlocks in this system.

Answer:

a.

① Mutual exclusion: Two cars cannot cross the roads at the mean time.

② Hold and wait: Any car is on this road and waiting for another road.

③ No preemption: No car can give up the resource of road;

④ Circular wait: The waiting queue is a cycle.

b. By using preemption to avoid deadlock, that's mean let the cars quit the system at the crossroad.

- **7.5**
  - **a.** √
  - **d.** √
  - **f.** √

7.6 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.

Answer:

Suppose the system is deadlocked. This implies that each process is holding one resource and is waiting for one more.

Since there are three processes and four resources, one process must be able to obtain two resources. This process requires no more resources and, therefore it will return its resources when done.

7.7 Consider a system consisting of *m* resources of the same type, being shared by *n* processes. Resources can be requested and released by processes only one at a time. Show that the system is deadlock free if the following two conditions hold:

a. The maximum need of each process is between 1 and *m* resources

b. The sum of all maximum needs is less than *m* + *n*

Answer:

a. ∑i=1~n Maxi <m+n

b. Maxi>= 1 for all I

Proof: Needi= Maxi - Allocatini

If there exits a deadlock state then:

c. ∑i=1~n Allocationi=m

Use a. to get: ∑ Needi + ∑ Allocationi = ∑ Maxi<m+n

Use b. to get: ∑ Needi + m < m+n

Rewrite to get: ∑i=1~n Needi<n

This implies that there exists a process P1 such that Needi=0. since Maxi>=1 it follows that P1 has at least one resource that it can release. Hence the system cannot be in a deadlock state.

7.11 Consider the following snapshot of a system:

	Allocation	Max	Available
	A B C D	A B C D	A B C D
P0	0 0 1 2	0 0 1 2	1 5 2 0
P1	1 0 0 0	1 7 5 0	
P2	1 3 5 4	2 3 5 6	
P3	0 6 3 2	0 6 5 2	
p4	0 0 1 4	0 6 5 6	

Answer the following questions using the banker's algorithm:

a. **What is the content of the matrix Need?**

b. **Is the system in a safe state?**

c. **If a request from process P1 arrives for (0,4,2,0), can the request be granted immediately?**

7.11 Answer:

a.

	Allocation A B C D	Max A B C D	Need A B C D
P0	0 0 1 2	0 0 1 2	0 0 0 0
P1	1 0 0 0	1 7 5 0	0 7 5 0
P2	1 3 5 4	2 3 5 6	1 0 0 2
P3	0 6 3 2	0 6 5 2	0 0 2 0
p4	0 0 1 4	0 6 5 6	0 6 4 2

b. **Exist a execute sequence p0,p2,p1,p3,p4,so the system is safe.**

c. **Exist a sequence p0,p2,p1,p3,p4,which can make the system in security. So the requirement can be fulfilled in time.**