1-a.(4%)Why we need dual mode of operation?
1-b.(16%)Which of the following instructions should be priviledged?

a. Set value of time.
b. Read the clock.
c. Clear memory.
d. Issue a trap instruction.
e. Turn off interrupts.
f. Modify entries in device-status table.
g. Switch from user to kernel mode.
h. Access I/O device.

2.There are two example program, the following one(figure 5.9, 5.10) uses semaphore to implement it.(p240, Fig 6.10/6.11 8th edition,p269, Fig 6.9/6.10 9th edition, p290, fig 7.1/fig7.2 10th edition)

```
do {
   . . .
   /* produce an item in next_produced */
   . . .
   wait(empty);
   wait(mutex);
   . . .
   /* add next_produced to the buffer */
   . . .
   signal(mutex);
   signal(full);
} while (true);
```

Figure 5.9 The structure of the producer process.

```
do {
   wait(full);
   wait(mutex);
   . . .
   /* remove an item from buffer to next_consumed */
   . . .
   signal(mutex);
   signal(empty);
   . . .
   /* consume the item in next_consumed */
   . . .
} while (true);
```

Figure 5.10 The structure of the consumer process.

In another consumer-producer example program figure 3.13 and 3.14(fig 3.12/fig 3.13 10th edition), a ring buffer queue is used to store the produced item that will be take off by the consumer later.

The following variables reside in a region of memory shared by the producer and cosumer processes:

```
#define BUFFER_SIZE 10

typedef struct {
    . . .
}item;

item buffer[BUFFER_SIZE];
int in = 0;
int out = 0;
```

=====

```
while (true) {
     /* produce an item in next_produced */

     while (((in + 1) % BUFFER_SIZE) == out)
        ; /* do nothing */

     buffer[in] = next_produced;
     in = (in + 1) % BUFFER_SIZE;
}
```

Figure 3.13 The producer process using shared memory.

```
item next_consumed;

while (true) {
     while (in == out)
        ; /* do nothing */

     next_consumed = buffer[out];
     out = (out + 1) % BUFFER_SIZE;

     /* consume the item in next_consumed */
}
```

Figure 3.14 The consumer process using shared memory.

2-a.(5%) Assume multiprocessors will be used in these two examples. Without using semaphore functions (fig 3.13 3.14) , Explain why more than one producer processes , producer-1 , producer-2 …producer-i programs could not be correctly executed concurrently ?

2-b.(5%) Assume only one producer and only one consumer can be assigned to execute concurrently. Show that even in more than two processors(multi-core) parallel processing system, without using any semaphore operation functions (fig 3.13 3.14) the result is still correct?

2-c. (5%)The critical section problem is caused by the race condition that the timer interrupt will break the updating statement of list queue pointer into non-atomic operation.
If there are only one producer and only one consumer program in running, Please indicate what is (or none) of the critical region in 3.13 and 3.14 example program?

2-d.(5%) If kernel monitor(R.C.A. Hoare Monitor) approach is adopted to support producer and consumer program. Any processes that can not call consumer or producer function in monitor simultaneously. That means the monitor restricts the parallel processing of these two functions. What is your suggestion to use 3.13/3.14 or 5.9/5.10 algorithm in monitor to support multiprocessor parallel processing? Why?

3-a.(5%)Show how to implement the mutual exclusion operations in multiprocessor shared memory environments using the swap() instruction.

3-b.(5%)The two process mutual exclusive lock algorithm(Petersons algorithm) in current multi-core architecture, the hardware processors and/or software compilers may reorder read and write operations that have no dependencies. The reordering of instructions may render inconsistent or unexpected results. That cause this algorithm will not be correct.
For queue/list pointer update operation may cause data race on a single variable, Please considering the multi-processors and shared memory environments, using the compare and swap () operation and atomic variables to implement the atomic interger sequence function: increment(&sequence) [Intel X86 using a special prefix instruction to assure atomic function]

4. (20 %) Are the following statements true or false? For each statement, you will get 4 points for correct answer, zero point for blank, or -2 point for incorrect answer.

(a) If the resource allocation graph contains a cycle, then deadlock exists.
(b) If a system is in unsafe state, then deadlock exists.
(c) The subnet mask for the subnet 200.23.16.0/23 is 255.255.255.0.
(d) Address Resolution Protocol (ARP) can be used to acquire IP addresses.
(e) Network Address Translation (NAT) is used to map IP addresses to MAC addresses.

5. (30%) Suppose that the following processes arrive for execution at the times indicated. Each process will run the listed amount of time. You may make some reasonable assumptions and write them down explicitly, if they are necessary to answer the following questions.

(a) Please draw Gantt charts that illustrate the execution of these processes using the following scheduling algorithms: FCFS, non-preemptive SJF, and preemptive SJF.

(b) Which of the algorithms in (a) results in the minimum average turnarround time (over all processes)? Be sure to justify your answer.

(c) Which of the algorithms in (a) results in the minimum average waiting time (over all processes)? Be sure to justify your answer.

Process	Arrival Time	Burst Time
P1	0	10
P2	5	3
P3	3	5
P4	4	4