1. (21 %) Are the following statements true or false? For each statement, you will get 3 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) The subnet 200.23.16.0/23 could accommodate up to 256 hosts.
   (e) Address Resolution Protocol (ARP) can be used to acquire IP addresses.
   (f) Network Address Translation (NAT) is used to map MAC addresses to IP addresses.
   (g) IPv6 addresses are 128 bits long.

2. (29%) 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

3. (35%)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)

   3-a.(5%) What are the differences in numbers applied applications between case1 0f figure 5.9, 5.10 and case2 of figure 3.13, 3.14? for example ,how many prondducers or consumers can be correctly parallel executed in these algorithm?

   3-b(5%)What is “pure code”?

   3-c(5%)What are these share variables in producers and consumers figure 5.9, 5.10?

```
int n;
semaphore mutex = 1;
semaphore empty = n;
semaphore full = 0

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, a ring buffer queue is used to store the produced item that will be take off by the consumer later.

3-d. (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 the critical region in 3.13 and 3.14 example program?

3-e.(10%)If the bounded buffer sharing is supported with hardware test&set locking. Please implement the primitive that can support multiple cooperation programs concurrently and effectively running in multiprocessor/multicore system. (p118, fig 3.14/3.15 8th edition,p123/124 9th edition)

```
#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.

3-f.(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?

4. (15%)-a.(5%)Show how to implement the mutual exclusion operations in multiprocessor environments using the swap() instruction.

   -b.(10%)The solution should exhibit minimal busy waiting . Please proof the three requirements for this lock operation.