1. (15 %) Please explain (a) deadlock prevention and (b) deadlock avoidance.

2. (20 %) Suppose that a disk drive has 5000 cylinders, numbered 0 to 4999. The drive is currently serving a request at cylinder 134, and the previous request was at cylinder 125. The queue of pending requests, in FIFO order, is 86, 1470, 913, 3774, 948, 4509, 1022, 2750, 130. Starting from the current head position, what is the total distance (in cylinders) that the disk arm moves to satisfy all the pending requests, for SSTF and SCAN disk scheduling algorithms?

3. (15 %) Are the following statements about IP addresses 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) Address Resolution Protocol (ARP) can be used to acquire IP addresses.
   (e) Network Address Translation (NAT) is used to map MAC addresses to IP addresses.

4.(30%)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)

4-a.(5%)Why multiple producers and consumers can be parallel running correctly in figure 5.9, 5.10?

4-b.(5%)What is the critical section in figure 5.9, 5.10 ?

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.

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

背面還有 Please Turn Over

```
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.

4-c.(5%)The following figure 3.13 and 3.14 are another example program. What are the shared variables in these producer and consumer program?

4-d.(5%) In what conditions the program can running correctly without using the lock synchronization primitive to support the required processing? (p118, fig 3.14/3.15 8th 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.

4-e.(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 support multiprocessor parallel processing?

4-f.(5%). Why the problems addressed in the chapter Synchronizaion are nonexistent in functional languages, but there are existed in any procedural languages.

5.(20%)Show how to implement the wait() and signal() semaphore operations in multiprocessor environments using the swap() instruction. The solution should exibit minimal busy waiting.