1. (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)

   1-a.(10%) What are the differences(or applied applications) between figure 5.9, 5.10 and figure 3.13, 3.14? for example, how many producers or consumers can be correctly used in these algorithm?

   1-b(5%)Please complete the code in comment statement. Such that producers and consumers can be parallel running correctly in 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.

1-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 the critical region in 3.13 and 3.14 example program?

1-d.(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.

1-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 use 3.13/3.14 or 5.9/5.10 algorithm in monitor to support multiprocessor parallel processing? Why?

2. (15%)-a. (5%)Show how to implement the mutual exclusion operations in multiprocessor environments using the swap() instruction.

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

3. (15%) Suppose a thread is running in a critical section of code. It means that the thread has acquired all the locks through proper arbitration. Can this thread get context switched? Please explain the reasons.

4. (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) IPv6 addresses are 128 bits long.

5. (20%) Assume you have a system with a static priority CPU scheduler. Assume the scheduler supports preemption. Describe what the program below prints in sequence, where the priorities are set such that T2 has a high priority, T1 has the middle priority, and T0 has the low priority. Assume the system starts with only T0 executing. Assume the semaphore mutex is initialized to 1.

```
void T0 ( )  {
   printf ( “T0-Start \n” );
   StartThread (T1);
   printf ( “T0-End \n” );
}
```

```
void T1 ( )  {
   printf ( “T1-Start \n” );
   P (mutex);
   printf ( “T1-A \n” );
   StartThread ( T2 );
   printf ( “T1-B \n” );
   V (mutex);
   printf ( “T1-End \n” );
}
```

```
void T2 ( )  {
   printf ( “T2-Start \n”);
   P (mutex);
   printf ( “T2-A \n” );
   V (mutex);
   printf ( “T2-End \n” );
}
```