1-a.(4%) Why we need dual mode of operation?

1-b.(5%) Some CPUs do not have a clean separation of privileged and nonprivileged instructions. The Intel x86 CPU line is one of them. The command popf loads the flag register from the contents of the stack. If the CPU is in user mode,then only some flags are replaced and others are ignored. Because no trap is generated if popf is executed in user mode, the trap-and-emulate procedure is rendered useless.The virtualization of x86 CPU was considered impossible in 1998.
What is the current approach to solve the problems of "special instructions such as x86 popf instruction" ?

1-c.(16%) Which of the following instructions should be priviledged?

a. Issue a trap instruction.
b. Turn off interrupts.
c. Modify entries in device-status table.
d. Switch from user to kernel mode.
e. Access I/O device.
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?

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.(30%) 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" );
}
```

5.(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) Pretending that deadlocks never occur in the system is a method to deal with deadlocks

(c) Banker's algorithm is a deadlock prevention algorithm

(d) It is not allowed for routers to fragment IPv6 packets.

(e) DHCP allows a host to dynamically obtain its IP address from network server when it joins the network.