Some of the answers to these questions can be found using the lecture slides, the recommended textbooks or other sources (question marked with *). Some questions may have more than one possible answer, or be more or less open for discussion. Note that no answers (solutions) will be given to these questions, but if help is needed the assistants will be available to answer questions. The concepts marked with yellow are important and should be fully understood.

Races & Mutual Exclusion

1. Describe the following requirements of the critical-section problem:
   (a) Mutual exclusion
   (b) Progress
   (c) Bounded waiting
2. Which possibilities to achieve mutual exclusion (ME) have been presented in the lecture? Describe them briefly.
3. What is the meaning of the term *busy waiting*? What other kinds of waiting are there in an operating system? Can busy waiting be avoided altogether? Explain your answer.
4. Explain why implementing synchronization primitives by disabling interrupts is not appropriate in a single-processor system if the synchronization primitives are to be used in user-level programs.
5. Servers can be designed to limit the number of open connections. For example, a server may wish to have only N socket connections at any point in time. As soon as N connections are made, the server will not accept another incoming connection until an existing connection is released. Explain how semaphores can be used by a server to limit the number of concurrent connections.
6. Show that, if the `wait()` and `signal()` semaphore operations are not executed atomically, then mutual exclusion may be violated.
7. The following program for solving the Producer - Consumer problem can have a race condition. Under which circumstances can this happen (the buffer is bounded)?

```
# define N 100                    //nr of slots in the buffer
int count = 0;                    //nr of items in the buffer

void producer(void){
    int item;
    while (TRUE){                 //repeat forever
        item = produce_item();    //generate next item
```

```
        if (count==N) sleep();    //if buffer full, go to sleep
        insert_item(item);      //put item in buffer
        count = count + 1;      //increment count of items
        if (count==1)
            wakeup(consumer)    //was buffer empty?
    }
}

void consumer(void){
    int item;
    while (TRUE){               //repeat forever
        if (count==0) sleep();  //if buffer empty, sleep
        item = remove_item();   //take out of buffer
        count = count - 1;      //decrement count of items
        if (count==N-1)
            wakeup(producer);   //was buffer full?
        consume_item(item);     //use item
    }
}
```