1. Provide relevant parts of C-like codes for processes X1, X2, X3, X4 and X5 in the following synchronization problem. Each process has a special (synchronization) point in its code, and:
   - Process X1 may cross its synchronization point unconditionally
   - Process X4 has to wait for processes X3 and X1 to cross or reach their synchronization points, and only then X4 may cross its synchronization point,
   - Process X2 has to wait for process X1 to cross or reach its synchronization point, and only then X2 may cross its synchronization point,
   - Process X3 has to wait for process X4 to cross or reach its synchronization point, and only then X3 may cross its synchronization point,
   - Process X5 has to wait for processes X1, X3 and X4 to cross their synchronization points, and only then X5 may cross its synchronization point.
   Assume that each process will cross its synchronization point only once.

2. Provide relevant parts of C-like codes for N processes (N is an arbitrary integer larger than 1) in N process N way synchronization, i.e. each process has a synchronization point in its code, and any of processes has to wait for other N-1 processes to cross or reach their synchronization points, and only then that process may cross its synchronization point. One solution, that follows ideas from the similar problems we discussed in the class, would be to use N semaphores all initialized to 0. But for this problem, your solution may use only 2 semaphores. Hint: All processes but one would have the same code for this synchronization.

3. Provide relevant parts of C-like codes for processes X, Y, Z, and W in the following synchronization problem. Each process has a special (synchronization) point in its code, and each process may cross its synchronization point only if at least two other processes reached or crossed their synchronization points. Assume that each process will cross its synchronization point only once.

4. This problem is an extension of ‘Consumer-Producer Problem’. Provide C-like codes for processes in the problem ‘One Consumer and both Producers’. In this problem, there are two producers and each producer has its own buffer (with X, and Y slots, respectively). Producers behave in the usual way. The only consumer takes, in each iteration, one item from each buffer.

5. This problem is an extension of ‘Consumer-Producer Problem’. Provide C-like codes for processes in the problem ‘One Consumer and any of two Producers’. In this problem, there are two producers and each producer has its own buffer (with M, and N slots, respectively). Producers behave in the usual way. In each iteration, the consumer takes one item from either buffer. Hint: Only if both buffers are empty, then the consumer is blocked, and it should be unblocked when any of buffers receives an item.

In all problems above, use semaphores as only synchronization mechanisms. Busy waiting solutions are not allowed. Do not over-synchronize, i.e. do not unnecessary block a process.

6. Let function getValue() returns the current semaphore value. This function may be invoked prior to calling wait() e.g.
if (getValue(&sem)>0) wait(&sem)

thereby preventing blocking on wait(). Describe a potential problem that could occur when using getValue() in this scenario.

7. You are given codes for the producer and the consumer. Codes use two hypothetical system calls: sleep() and wakeup(X).
- sleep() - The issuing process X is blocked and it will stay blocked until waken up by some process that issues wakeup(X) system call.
- wakeup(X) - This system call does not effect in any way the issuing process. If the process X has been blocked because it previously issued sleep(), the process X is unblocked and put in a ready state, otherwise this system call does not have any effect.

```c
#define N = 100
int count =0;
typedef struct
    { ...
    } item;
item buffer [N];
```

**Producer code:**
```c
{item Produced;
while (true)
    { /*produce an item
        in Produced*/
            if (count == N) sleep();
        
    
    /* put item from Produced
    into buffer */
        count = count + 1;
        if (count == 1) wakeup (Consumer);
    }
}
```

**Consumer code:**
```c
{item Consumed;
while (true)
    {if (count == 0) sleep();
    /*take an item from
    buffer an put into
    Consumed*/
        count = count - 1;
        if (count == N-1)wakeup(Producer)
        /* consume the item from
    Consumed*/
    }
}
```

a. The codes have an unresolved critical section problem related with the following two statements:
- count = count + 1 /* in producer code
- count = count - 1 /* in consumer code

We discussed that problem in the class and it is not of concern for this problem, i.e. assume that the variable count is updated in those two statements in such way that there is no critical section problem.

b. It is also assumed that access to the buffer is orderly, so no problem there.

But, there is still a problem with the given codes. It is possible that both processes may execute sleep() system call, thus without any of them being able to execute a wakeup system call.

Provide one such scenario illustrating that something like that can happen. Your scenario should be along the following lines: one process runs first and at certain point it is interrupted, and then the second process runs and after some time is interrupted and so on, until both processes issue sleep() system call. At the beginning, you should assume that Producer started first and produces some number (<N) of items. Note: You are not asked to fix this code.