1. Design a data link protocol with the following characteristics: The sender receives a sequence of messages from the network layer to be transmitted to the receiver. The physical layer may lose messages but no two consecutive messages can be lost. To overcome message loss, for each message received from the network layer, the sender sends two messages consecutive messages, each containing the same data. If one of them is lost, the other is guaranteed to be received. Design a sliding window protocol for this case.

2. Design a data link protocol under the following assumptions: (a) in the physical layer, data messages may get lost but not ack messages, (b) messages are not reordered, (c) timeouts are non-premature and (d) window size is one (as in the alternating bit protocol). Use the least amount of extra information in the message (that is, least range of sequence numbers possible or perhaps even without sequence numbers) and the minimal number of messages.

3. Design a two-process protocol with the following properties: There are three processes, A, B and C. Process A and B can be one of the following states: idle and reading. Process C can be one of the following states: idle and writing. Thus, A and B can only read whereas C can only write. It is required that a process cannot be in state reading when the other one is in state writing. The protocol must allow the two processes to be in state reading at the same time. You may add new states to the processes (such as a state in which a process has made a request to read or write). You may use either the state machine notation or the guarded command notation.

4. In the class, we designed a token protocol in which the token is passed between two processes in a cyclic manner (that is, it is not request based). Redesign the protocol under the assumption that the messages can get lost and timeouts are non-premature. Also, assume that messages cannot get reordered.

5. The following is a sliding window protocol with block acknowledgement. In this protocol, the receiver has the option of sending acknowledgements for a block of data items. Thus, an ack$(x,y)$ message is an acknowledgement for all data items numbered from $x$ to $y$. The receiver has an additional variable $sa$ ($sa = k$ implies that R has sent acknowledgement for all data items up to (but not including) $k$). Assume message loss and non-premature timeouts. Initially, $recd[1..∞] = ackd[1..∞] = false$
$S:$  
\[
ns := 1; na := 1; \\
*[
ns < na + W \rightarrow R!(sdata[ns], ns); ns := ns + 1; cancel timer; 
\]
\[\text{if } ns = na + W \text{ then start timer}\]
\[
\text{timeout } \rightarrow R!(sdata[na], na)
\]
\[
R?ack(x, y) \rightarrow \text{while } x \leq y \text{ do } ackd[x] := true; x := x + 1 \text{ od}
\]
\[
ackd[na] \rightarrow na := na + 1;
\]

$R:$  
\[
nr := 1; sa := 1; \\
*[
S?(x, y) \rightarrow \text{if } y \geq sa \text{ then } rdata[nr] := x; recd[y] := true
\]
\[
recd[nr] \rightarrow nr := nr + 1
\]
\[
sa < nr \rightarrow S!ack(sa, nr); sa := nr
\]

Find possible errors in the protocol.