Tutorial week 6

Question 1

Two processes P and Q are connected in a ring using two channels, and they constantly rotate a message m. At any one time, there is only one copy of m in the system. Each process's state consists of the number of times it has received m, and P sends m first. At a certain point, P has the message and its state is 101. Immediately after sending m, P initiates the snapshot algorithm. Explain the operation of the algorithm in this case, giving the possible global state(s) reported by it.

Question 1 solution

- P sends msg m
- P records state (101)
- P sends marker (see initiation of algorithm in page 406)
- Q receives m, making its state 102
- Q receives the marker and by marker-receiving rule, records its state (102) and the state of the channel from P to Q as {}
- Q sends marker (marker-sending rule)
- (Q sends m again at some point later)
- P receives marker
- P records the state of the channel from Q to P as set of messages received since it saved its state = {} or empty set (marker-receiving rule).

Question 2

Is it possible to implement either a reliable or an unreliable (process) failure detector using an unreliable communication channel?

Question 2 solution

- An unreliable failure detector can be built on an unreliable channel
  - Since all that changes from using of a reliable channel, is that dropped messages may increase the number of false suspicions of process failure
- A reliable failure detector cannot be built on an unreliable channel
  - It requires a synchronous system
  - Since a dropped message and a failed process cannot be distinguished – unless the unreliability of the channel can be masked while providing a guaranteed upper bound on message delivery times

Question 2 solution

- In principle, a channel can be used to create a reliable failure detector
  - Provided that it dropped messages with some probability but, say, guaranteed that at least one message in a hundred was not dropped

Question 3

In the central server algorithm for mutual exclusion, describe a situation in which two requests are not processed in happened before order.

Question 3 solution

- Process A sends a request r_A for entry then sends a message m to B.
- On receipt of m, B sends request r_B for entry.
- To satisfy happened-before order, r_A should be granted before r_B.
- However, due to the vagaries of message propagation delay, r_B arrives at the server before r_A, and they are serviced in the opposite order
  - Because when the queue of waiting processes is not empty, then the server chooses the oldest entry in the queue, removes it and replies to the corresponding process

Question 4

Adapt the central server algorithm for mutual exclusion to handle the crash failure of any client (in any state), assuming that the server is correct and given a reliable failure detector. Comment on whether the resultant system is fault tolerant. What would happen if a client that possesses the token is wrongly suspected to have failed?

Question 4 solution

- The server uses the reliable failure detector to determine whether any client has crashed
- If the client has been granted the token then the server assumes/acts as if the client had returned the token
- In case it subsequently receives the token from the client (which may have sent it before crashing), it ignores the receiving
- The resultant system thereby is not fault-tolerant

Question 4 solution

- If a token-holding client crashed then the application-specific data protected by the critical section (whose consistency is at stake) may be in an unknown state at the point when another client starts to access it
- If a client that possesses the token is wrongly suspected to have failed then there is a danger that two processes will be allowed to execute in the critical section concurrently