Homework-6

Mohan Srinivas Naga Satya

NUID: 001821918

Question 15.1

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

Failure detectors is a service that processes queries about whether a particular process has failed. A failure detector is not-entirely accurate and based on this assumption, they are categorized into: *Unreliable* and *Reliable* failure detectors.

Unreliable failure detectors may produce one of two values when given the identity of a process: Unsuspected or Suspected. Both of these results are hints, which may or may not accurately reflect whether the process has actually failed. A result of Unsuspected signifies that the detector has recently received evidence suggesting that the process has not failed; for example, a message was recently received from it. But of course, the process may have failed since then. A result of Suspected signifies that the failure detector has some indication that the process may have failed.

A reliable failure detector is one that is always accurate in detecting a process's failure. It answers processes' queries with either a response of Unsuspected or Failed. A result of Failed means that the detector has determined that the process has crashed.

The failure detector executes on an algorithm which assumes that T+D (time taken to transmit + recurrence time) are needed for a message to be received from another process. As a result, in an unreliable communication channel, the message transmission time might vary or the message might be dropped altogether and as a result, it is impractical to implement an failure detector on a unreliable channel.

Question 15.5

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?

The central server algorithm achieves mutual exclusion by assigning a token to the client utilizing the resources in the zone. For achieving it, a process sends a request message to the server and awaits a reply from it. Conceptually, the reply constitutes a token signifying permission to enter the critical section. If no other process has the token at the time of the request, then the server replies immediately, granting the token. If the token is currently held by another process, then the server does not reply, but queues the request. When a process exits the critical section, it sends a message to the server, giving it back the token.

To modify the central server algorithm to handle any crash failures, let us consider that the server uses a reliable failure detector. So, in the case when the exit message is received from the client, the server removes it from queue. But in the case when the exit message is not received from the client, the server can consider that the client has responded with the token and remove the binding to the client. In doing so, the server handles crash failures and also provides mutual exclusion.

The fault-tolerance of the system with respect to the process failures is dependent on the reliability of the detector. Assuming the system is established on a reliable communication channel, the underlying design of reliable failure detector makes the central server tolerant to process failures.

When the client possessing the token is wrongly suspected to have failed, then the resources will be continued to be utilized by the client while another client now possessing token waits in the queue. This causes a deadlock within the system and leads to system crash.

Question 15.20

Construct a solution to reliable, totally ordered multicast in a synchronous system, using a reliable multicast and a solution to the consensus problem.

In synchronous systems, a consensus for a system is achieved when the group of processes which are correct arrive at a common proposed value after each round. As the algorithm guarantees that though a threshold of f processes fails, the remaining processes which reach a decisive proposed value.

Similarly, we can arrive at a solution for reliable, totally ordered multicast group of messages using the above specified consensus solution. Let us consider that a process wants to communicate a message *m* within the system. In doing so, to achieve total ordering, the process attached a unique identifier with each message. Thus, using a reliable multicast, the process sends a message within the group.

To reach consensus about the same, the reliable multicast message from different process is received by other processes in the group. In doing so, when all the processes within the multicast group arrive at a common proposed unique identifier, a totally ordered and reliable multicast is achieved after each round. In doing so, the agreed order can be propagated back to the processes and the messages can be transmitted in the same order achieving reliable totally ordered multicast.

Question 15.23

Show that Byzantine agreement can be reached for three generals, with one of them faulty, if the generals digitally sign their messages.

In Byzantine generals' problem, the difficulty of processes to identify a faulty or traitor with less than three generals is difficult as the correct processes cannot distinguish between faulty and correct processes messages. But as the messages are digitally signed, it would be possible for each process to identify the faulty process amongst the midst.

Assumptions:

There would be no message loss.

- In a synchronous system, the dropped messages are identified.
- The source of the messages is not masked.

- A digitally signed message cannot be forged/modified.
- Authenticity of a signature can be verified.

Algorithm:

A process P sends a message m to every other process by signing it.
When a process P1 receives a message from process P, it verifies the message m from P.
Only once the verification is complete, it adds the message to its queue and propagates the message when the number of failures is less.
After m+2 rounds, the decision is reached on the action to be performed.

In this process, with the byzantine generals M (where M<=3, with one faulty general), if the commander is faulty, after m+1 round, the lieutenants identify the mistake and discard it. But in the case lieutenants is wrong, the system achieves consistency after round 1 itself.