**ROLL NO. : BATCH: DATE:**

**NAME:**

**EXPERIMENT NO: 05**

**EXPERIMENT TITLE:** Implementation of Election Algorithm.

**Problem statement**

Your bank account details replicated at a few servers and out of these one of the server is responsible for receiving all reads and writes. That means it is the leader who coordinating among all replicas.

* What if there are two leaders per customer?
* What if servers disagree about who the leader is?
* What if leader crashes?

Each of the above scenarios leads to the inconsistency.

Leader is useful for coordinating among distributed servers. In a nutshell; [2]

* Group of processes, elect a ,Leader’ to undertake special task
  + Let everyone know in this group about this leader
* What happens if leader fails or Crashes?
  + Some processes detects this using failure detector!
  + What next…. Motivation for Election Algo.

**Election Algorithm**:

* Elect only one leader among the non-faulty processes
* All non-faulty processes agree on who is the leader

**System Model**

* N processes
* Each process has unique ID (IP addr, Port no.)
* Messages are eventually delivered
* Failures may during election protocol - When you tried to elect leader, a leader may fail or other processes in system might fail, Election algo. Should be tolerant to such failures

**Requirement: Calling for an Election**

* Any process can call for an election
* A process can call for at most one election at a time
* Multiple processes can call an election simultaneously
  + All of them together must yield one a single leader
* The result of election should not depend on which process calls for it

At the end of the election protocol the non-faulty process with (=best (highest) election attribute is elected.

* Common attribute: Leader has highest ID
* Other attribute examples: leader has highest IP addr or fastest CPU, more disk space or most no. of files etc.

**Bully Algorithm: [1]**

The bully algorithm is a method in distributed computing for dynamically electing a coordinator by process ID number. The process with the highest process ID number is selected as the coordinator. When a process P determines that the current coordinator is down because of message timeouts or failure of the coordinator to initiate a handshake, it performs the following sequence of actions:

* P broadcasts an election message (inquiry) to all other processes with higher process IDs, expecting an "I am alive" response from them if they are alive.
* If P hears from no process with a higher process ID than it, it wins the election and broadcasts victory.
* If P hears from a process with a higher ID, P waits a certain amount of time for any process with a higher ID to broadcast itself as the leader. If it does not receive this message in time, it re-broadcasts the election message.
* If P gets an election message (inquiry) from another process with a lower ID it sends an "I am alive" message back and starts new elections.

*Assumptions:*

* Each process knows the ID and address of every other process
* Communication is reliable
* A process initiates an election if it just recovered from failure or it notices that the coordinator has failed
* Three types of messages: *Election, OK, Coordinator*
* Several processes can initiate an election simultaneously
* Need consistent result

**Details:**

* Any process P can initiate an election
* P sends *Election* messages to all process with higher IDs and awaits *OK* messages
  + If no *OK* messages, P becomes coordinator and sends *Coordinator* messages to all processes with lower IDs – If it receives an *OK*, it drops out and waits for an *Coordinator* message
* If a process receives an *Election* message
  + Immediately sends *Coordinator* message if it is the process with highest ID – Otherwise, returns an *OK* and starts an election
* If a process receives a *Coordinator* message, it treats sender as the coordinator.

Example



****

**Deliverables: (Place your work here directly under given steps)**

A solution description document (Times New Roman, 12pt, single spacing) containing:

1. Conceptual architecture of the distributed system you have considered for applying Bully algorithm for election.
2. Source files
3. Show process/event call steps
4. Test Procedure for objective validation, i.e show the performance analysis for following cases:
   1. Case 1: Failure of the coordinator,
      * Worst case: The process having the lowest priority (in the system) noticed that the coordinator has just crashed
      * Best case: The process having the priority just below the failed coordinator, detects that the coordinator has failed.
   2. Case 2: A failed process (either a former coordinator or a normal process) recovers from its failed state.
      * *Worst case:* the process with the lowest priority must initiate recovery action, and hence requires ***O (n2****)* messages.
      * *Best case:* In the best case the bully algorithm requires *(n-1)* messages.
5. Answer following questions:
   1. Give a diagram and explain 3 messages - election, vote, and coordinator.
   2. What happens if two processes notice at the same time, that a process has crashed?