EXPERIMENT NO : 06

AIM: Implement an Election Algorithm.

Theory:

Distributed Algorithm is an algorithm that runs on a distributed system. Distributed system is a collection of independent computers that do not share their memory. Each processor has its own memory and they communicate via communication networks. Communication in networks is implemented in a process on one machine communicating with a process on another machine. Many algorithms used in the distributed system require a coordinator that performs functions needed by other processes in the system.

Election algorithms are designed to choose a coordinator.

Election Algorithms: Election algorithms choose a process from a group of processors to act as a coordinator. If the coordinator process crashes due to some reasons, then a new coordinator is elected on other processor. Election algorithm basically determines where a new copy of the coordinator should be restarted. Election algorithm assumes that every active process in the system has a unique priority number. The process with highest priority will be chosen as a new coordinator. Hence, when a coordinator fails, this algorithm elects that active process which has highest priority number. Then this number is sent to every active process in the distributed system.

We have two election algorithms for two different configurations of a distributed system.

1. **The Bully Algorithm** — This algorithm applies to system where every process can send a message to every other process in the system.

Algorithm —

Suppose process P sends a message to the coordinator.

1. If the coordinator does not respond to it within a time interval T, then it is assumed that coordinator has failed.
2. Now process P sends an election message to every process with high priority number.
3. It waits for responses, if no one responds for time interval T then process P elects itself as a coordinator.
4. Then it sends a message to all lower priority number processes that it is elected as their new coordinator.
5. However, if an answer is received within time T from any other process Q,
   - Process P again waits for time interval T' to receive another message from Q that it has been elected as coordinator.

- (Il) If Q does not respond within time interval T' then it is assumed to have failed and algorithm is restarted.

2. **The Ring Algorithm** — This algorithm applies to systems organized as a ring(logically or physically). In this algorithm we assume that the link between the process is unidirectional and every process can message to the process on its right only. Data structure that this algorithm uses is active list, a list that has a priority number of all active processes in the system.

Algorithm —

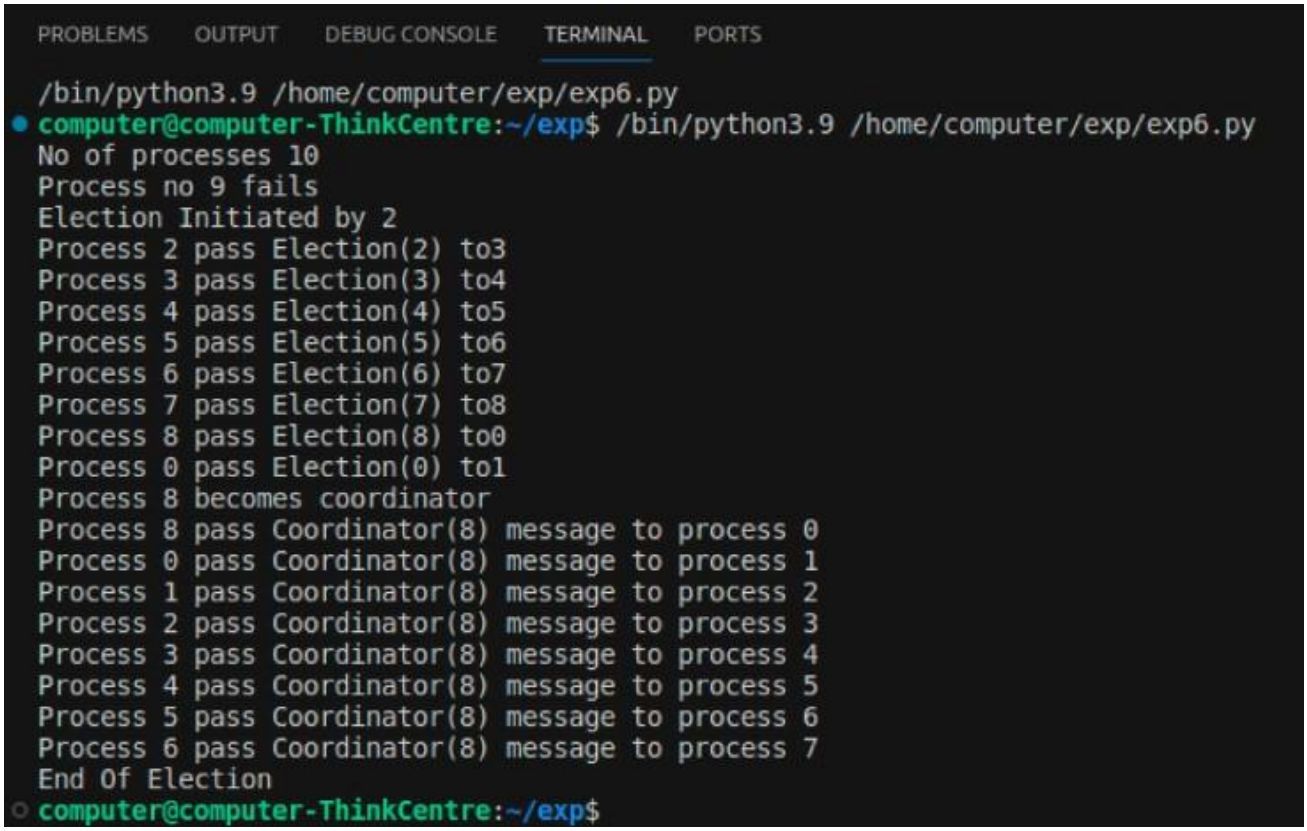
1. If process PI detects a coordinator failure, it creates new active list which is empty initially. It sends election message to its neighbour on right and adds number I to its active list.
2. If process P2 receives message elect from processes on left, it responds in 3 ways:
   - If message received does not contain I in active list, then PI adds 2 to its active list and forwards the message.
   - (Il) If this is the first election message it has received or sent, PI creates new active list with numbers I and 2. It then sends election message I followed by 2.
   - (Ill) If Process PI receives its own election message I then active list for PI now contains numbers of all the active processes in the system. Now Process PI detects highest priority number from list and elects it as the new coordinator.

Code :

```
exp6.py > GFG > initialiseGFG
class Pro:
    def __init__(self, id):
        self.id = id
        self.act = True

class GFG:
    def __init__(self):
        self.TotalProcess = 0
        self.process = []

    def initialiseGFG(self):
        print("No of processes 10")
        self.TotalProcess = 10
        self.process = [Pro(i) for i in range(self.TotalProcess)]

    def Election(self):
        print("Process no " + str(self.process[self.FetchMaximum()].id) + " fails")
        self.process[self.FetchMaximum()].act = False
        print("Election Initiated by 2")
        initializedProcess = 2

        old = initializedProcess
        newer = old + 1

        while (True):
            if (self.process[newer].act):
                print("Process " + str(self.process[old].id) + " pass Election(" + str(self.process[old].id) + ") to" + str(self.process[newer].id))
                old = newer
            newer = (newer + 1) % self.TotalProcess
            if (newer == initializedProcess):
                break

        print("Process " + str(self.process[self.FetchMaximum()].id) + " becomes coordinator")
        coord = self.process[self.FetchMaximum()].id

        old = coord
        newer = (old + 1) % self.TotalProcess
        while (True):
            if (self.process[newer].act):
                print("Process " + str(self.process[old].id) + " pass Coordinator(" + str(coord) + ") message to process " + str(self.process[newer].id))
                old = newer
            newer = (newer + 1) % self.TotalProcess
            if (newer == coord):
                print("End Of Election ")
                break

    def FetchMaximum(self):
        maxId = -9999
```

```
    def FetchMaximum(self):
        maxId = -9999
        ind = 0
        for i in range(self.TotalProcess):
            if (self.process[i].act and self.process[i].id > maxId):
                maxId = self.process[i].id
                ind = i
        return ind

def main():
    object = GFG()
    object.initialiseGFG()
    object.Election()

if __name__ == "__main__":
    main()
```

Output :



Conclusion:

Thus, we had Successfully Implemented Election Algorithm.