## Experiment 5

**Aim:** Develop a program for Election Algorithm

**Tool**s: Java / python

**Theory:**

Several distributed algorithms require that there be a coordinator process in the entire system that performs some type of coordination activity needed for the smooth running of other processes in the system.
Two examples of such coordinator processes encountered

1. The coordinator in the centralized algorithm for mutual exclusion.
2. The central coordinator in the centralized deadlock detection algorithm.

Since all other processes in the system have to interact with the coordinator, they all must unanimously agree on who the coordinator is. Furthermore, if the coordinator process fails due to the failure of the site on which it is located, a new coordinator process must be elected to take up the job of the failed coordinator. Election algorithms are meant for electing a coordinator process from among the currently running processes in such a manner that at any instance of time there is a single coordinator for all processes in the system.

Election algorithms are based on the following assumptions:

1. Each process in the system has a unique priority number.
2. Whenever an election is held, the process having the highest priority number among the currently active processes is elected as the coordinator.
3. On recovery, a failed process can take appropriate actions to rejoin the set of active processes.

Therefore, whenever initiated, an election algorithm basically finds out which of the currently active processes has the highest priority number and then informs this to all other active processes.

**Code:**

```
class Pro:
    def __init__(self, id):
        self.id = id
        self.act = True

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

    def initialiseNode(self):
        print("No of processes 5")
        self.TotalProcess = 5
        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
        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 = Node()
    object.initialiseNode()
    object.Election()

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

**Output:**

```
PS C:\Users\comp lab-325\Desktop\DAA 11> & C:/Python312/python.exe
"c:/Users/comp lab-325/Desktop/DAA 11/election-ring-algo.py"
No of processes 5
Process no 4 fails
Election Initiated by 2
Process 2 pass Election(2) to3
Process 3 pass Election(3) to0
Process 0 pass Election(0) to1
Process 3 becomes coordinator
Process 3 pass Coordinator(3) message to process 0
Process 0 pass Coordinator(3) message to process 1
Process 1 pass Coordinator(3) message to process 2
End Of Election
```

**Learning Objective:** Students should have the ability to

LO1: Describe Election algorithms in distributed computing

LO2: Write a program to demonstrate election algorithm

**Course Outcomes:** Upon Completion of this students will be able to understand the concept of Election Algorithms in distributed computing.

**Conclusion:**

We studied the concepts of election algorithms and understood how to implement them through a program. Several concepts of distributed systems were revised during the implementation.

For Faculty Use:

| **Correction Parameters** | **Formative Assessment [40%]** | **Timely completion of Practical [ 40%]** | **Attendance/ Learning Attitude [20%]** | |
|---|---|---|---|---|
| **Marks Obtained** | | | | |