St. Francis Institute of Technology

Department of Computer Engineering

Academic Year: 2021-2022 **Semester: VIII**

Subject: **Class/Branch/: BE/CMPNA**

Name :- Nithin Menezes **Roll Number: 56**

Experiment 5

Aim: Implementation of election algorithm

Theory:

Purpose of election algorithm

Election algorithms choose a process from 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 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 send to every active process in the distributed system.

Types of election algorithm

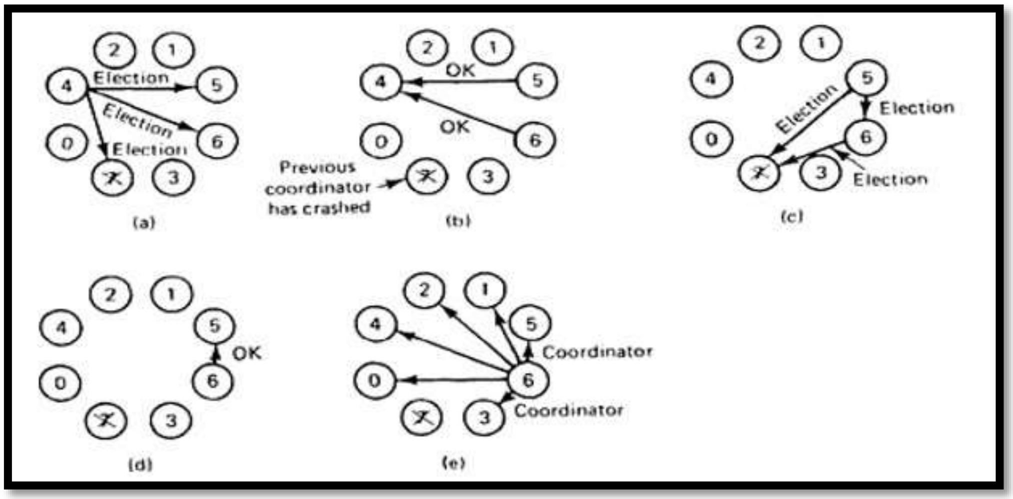
We have two election algorithms for two different configurations of distributed system.

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 coordinator does not respond to it within a time interval T, then it is assumed that coordinator has failed.

2. Now process P sends 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,
    - (I) Process P again waits for time interval T' to receive another message from Q that it has been elected as coordinator.
    - (II) If Q doesn't responds 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 are 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 priority number of all active processes in the system.

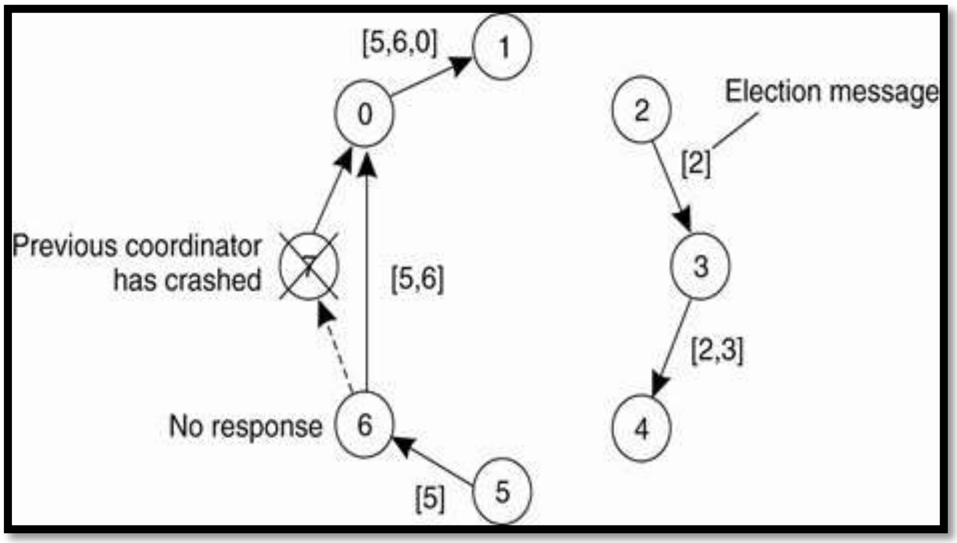
   Algorithm –

1. If process P1 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 1 to its active list.
2. If process P2 receives message elect from processes on left, it responds in 3 ways:

○ (I) If message received does not contain 1 in active list then P1 adds 2 to its active list and forwards the message.

○ (II) If this is the first election message it has received or sent, P1 creates new active list with numbers 1 and 2. It then sends election message 1 followed by 2.

○ (III) If Process P1 receives its own election message 1 then active list for P1 now contains numbers of all the active processes in the system. Now Process P1 detects highest priority number from list and elects it as the new coordinator.

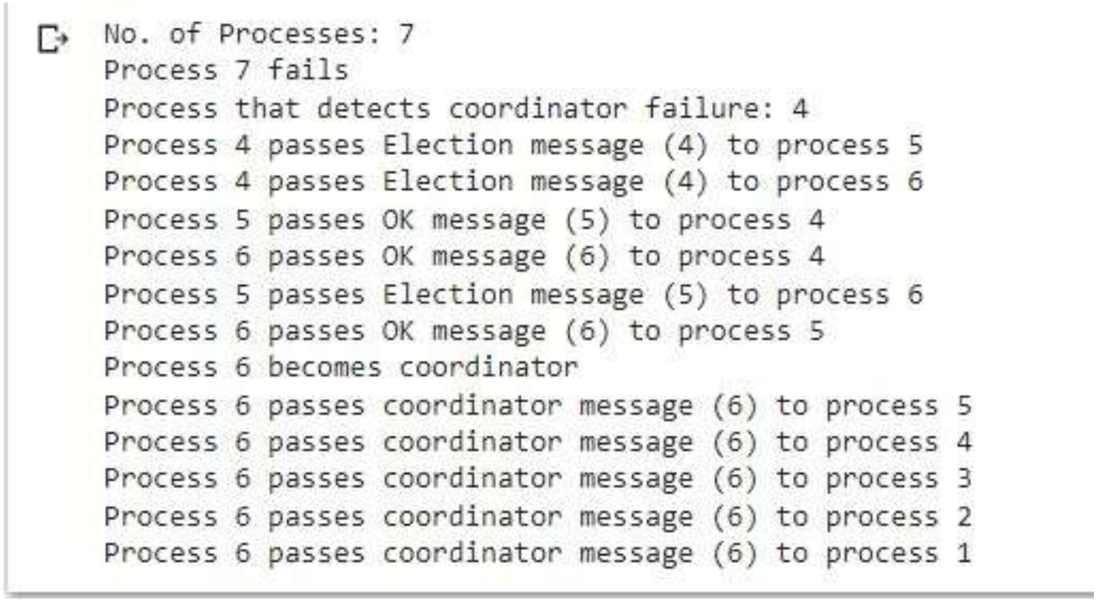


Code

```
import time
global process
class Process:
    def __init__(self, id):
        self.id = id
        self.status = "active"
def getMax():
    m = 0
    for i in process:
        if i.status == "active" and i.id > m:
            m = i.id
    return m
def Election():
    time.sleep(2)
    m = getMax()
    process[m-1].status = "inactive"
    print(f"Process {m} fails")
    initiator = int(input("Process that detects coordinator failure: ")
    while True:
        higherProcess = False
        for i in process[initiator:]:
            if i.status == "active":
                print(f"Process {initiator} passes Election message
({initiator}) to process {i.id}")
```

```
                higherProcess = True
        if higherProcess:
            for i in process[initiator:]:
                if i.status == "active" and i.id != initiator:
                    print(f"Process {i.id} passes OK message ({i.id}) t
process {initiator}")
            initiator += 1
        else:
            coordinator = getMax()
            print(f"Process {coordinator} becomes coordinator")
            for i in process[coordinator - 2::-1]:
                print(f"Process {coordinator} passes coordinator message
({coordinator}) to process {i.id}")
            break
n = int(input("No. of Processes: "))
process = []
for i in range(n):
    process.append(Process(i+1))
Election()
```

OUTPUT:



```
No. of Processes: 7
Process 7 fails
Process that detects coordinator failure: 4
Process 4 passes Election message (4) to process 5
Process 4 passes Election message (4) to process 6
Process 5 passes OK message (5) to process 4
Process 6 passes OK message (6) to process 4
Process 5 passes Election message (5) to process 6
Process 6 passes OK message (6) to process 5
Process 6 becomes coordinator
Process 6 passes coordinator message (6) to process 5
Process 6 passes coordinator message (6) to process 4
Process 6 passes coordinator message (6) to process 3
Process 6 passes coordinator message (6) to process 2
Process 6 passes coordinator message (6) to process 1
```

Conclusion: In this experiment we have implemented the election algorithm. There are two types of election algorithm - Bully algorithm and Ring algorithm. The bully algorithm is a type of election algorithm in distributed computing for dynamically selecting a coordinator by process Id number. In the ring algorithm, all the processes are arranged in a unidirectional logical ring.