Experiment No: - 6

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.

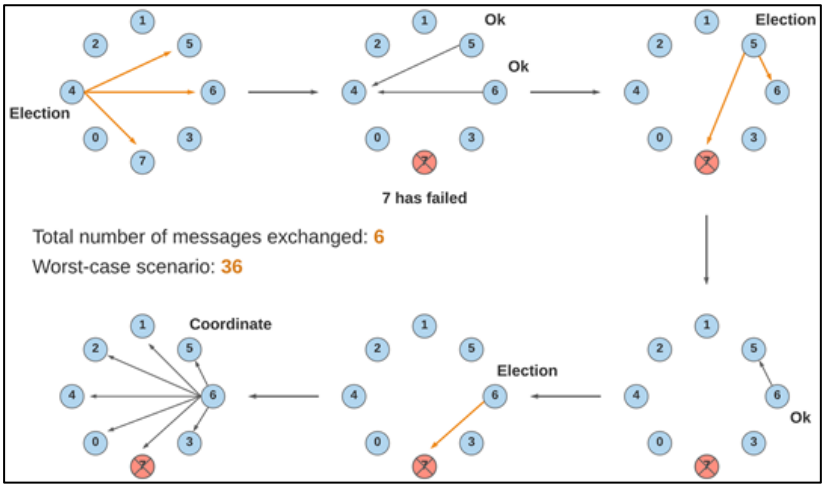
Election Algorithms:

Election algorithms are designed to choose a coordinator. 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 send 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 messages 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.

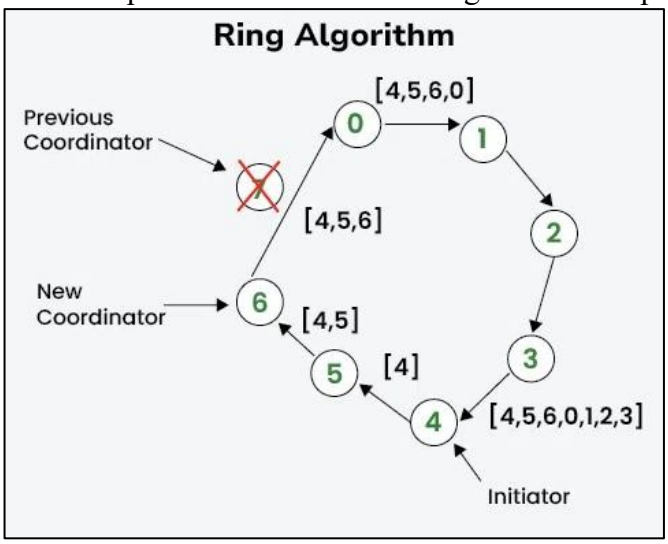


4. Then it sends a message to all lower priority number processes that it is elected as their new coordinator.
5. However, if an answer is received within time T from any other process Q, • (I) Process P again waits for time interval T' to receive another message from Q that it has been elected as coordinator.
   • (II) If Q doesn't respond within time interval T' then it is assumed to have failed and algorithm is restarted.

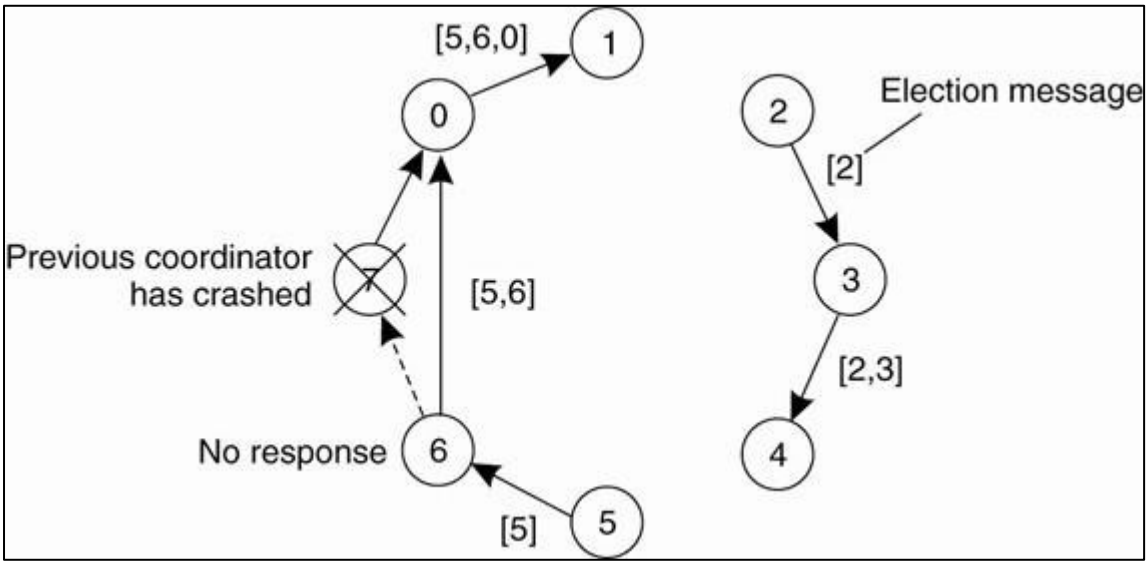
2. The Ring Algorithm – This algorithm applies to systems organized as a ring(logically or physically). In this algorithm we assume that the link between the process are unidirectional and every process can message to the process on its right only. Data structure that this algorithm uses is **active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



- 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.



(I) If message received does not contain 1 in

Comparison of Bully and Ring Algorithms –

Feature	Bully Algorithm	Ring Algorithm
Structure	Any process can communicate with any other	Processes communicate in a logical ring
Election Trigger	Lowest-ID process detects failure	Any process detects failure
Message Complexity	$O(n^2)$ in worst case	O(n)
Time Complexity	O(n)	O(n)
Coordinator Selection	Highest-ID process	Highest-ID process

Fault Tolerance	More resilient to failures	Can fail if the ring is broken

Both algorithms ensure that a new coordinator is elected when the current one fails, but the Bully Algorithm is faster, while the Ring Algorithm is more structured.

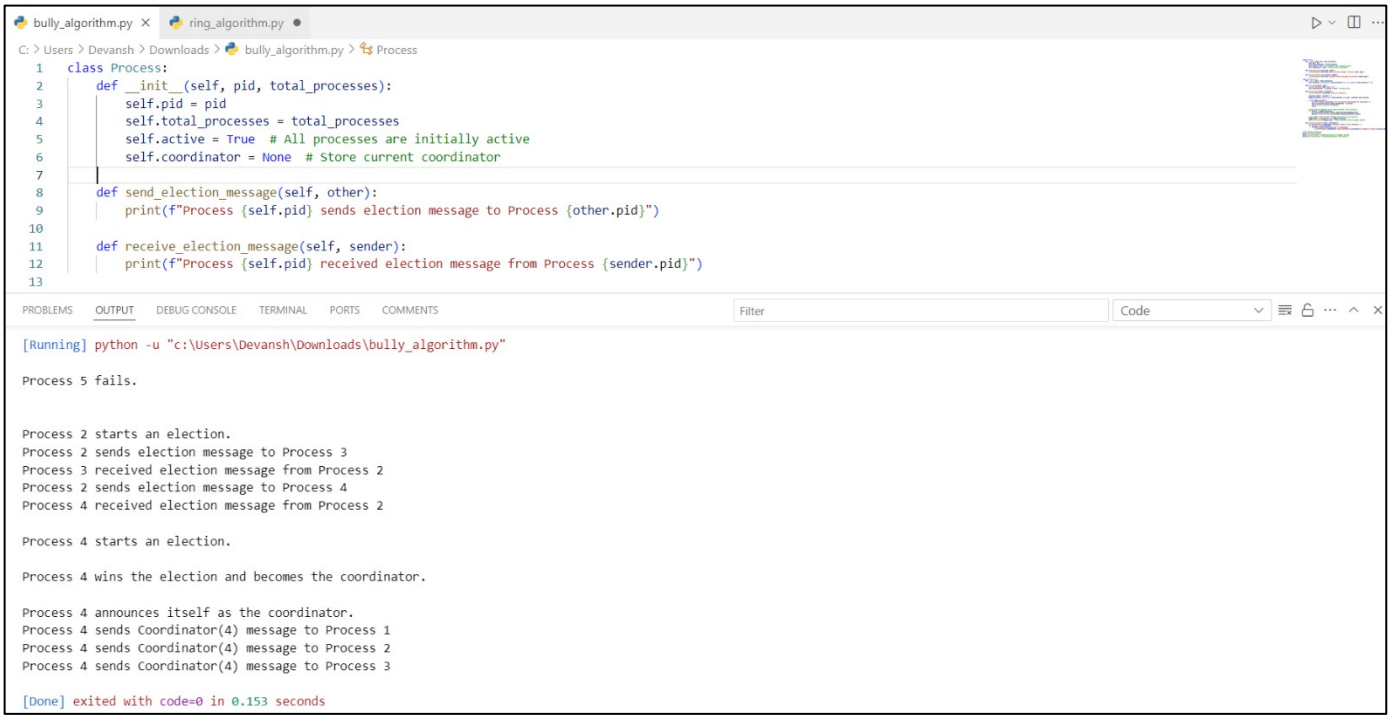
Program & Output:

1. Bully Algorithm –

```
class Process:     def __init__(self, pid,
total_processes):         self.pid = pid
        self.total_processes = total_processes
self.active = True  # All processes are initially active
self.coordinator = None  # Store current coordinator

    def send_election_message(self, other):         print(f"Process {self.pid}
sends election message to Process {other.pid}")

    def receive_election_message(self, sender):         print(f"Process {self.pid}
received election message from Process {sender.pid}")

class BullyElection:     def __init__(self, total_processes):         self.processes =
[Process(i, total_processes) for i in range(1, total_processes + 1)]

    def fail_process(self, pid):
print(f"\nProcess {pid} fails.\n")
        self.processes[pid - 1].active = False  # Process fails

    def start_election(self, initiator):
print(f"\nProcess {initiator} starts an election.")

        initiator_index = initiator - 1
        higher_processes = [p for p in self.processes if p.pid > initiator and p.active]

        if not higher_processes:             print(f"\nProcess {initiator} wins the election
and becomes the coordinator.")             self.processes[initiator_index].coordinator =
initiator             self.announce_coordinator(initiator)             return
        # Send election messages to all higher-numbered active processes
for process in higher_processes:
            self.processes[initiator_index].send_election_message(process)
process.receive_election_message(self.processes[initiator_index])

        # The highest active process responds and starts its own election
highest_pid = max(p.pid for p in higher_processes)
        self.start_election(highest_pid)  # Start election from the highest process

    def announce_coordinator(self, coordinator):         print(f"\nProcess {coordinator}
announces itself as the coordinator.")         for process in self.processes:             if
```

process.active and process.pid != coordinator: print(f"Process {coordinator} sends Coordinator({coordinator}) message to Process {process.pid}")

Run the Bully Algorithm bully = BullyElection(5)
bully.fail_process(5) # Simulating failure of highest process
bully.start_election(2) # Initiating election from Process 2



2. Ring Algorithm _

class Process:
def __init__(self, pid):
self.pid = pid # Process ID
self.active = True # Process is active

class RingElection: def __init__(self, total_processes):
self.processes = [Process(i) for i in range(total_processes)]

def fetch_maximum(self):
max_id = -1 index = 0 for
i, p in enumerate(self.processes):
if p.active and p.pid > max_id:
max_id = p.pid index = i
return index

def start_election(self, initiator):
print(f"\nProcess {self.processes[self.fetch_maximum()].pid} fails.")
self.processes[self.fetch_maximum()].active = False # Simulate a failure

```
        print(f"\nElection initiated by Process {initiator}")
old = initiator
        new = (old + 1) % len(self.processes)

        active_list = [self.processes[initiator].pid]  # Include initiator in the list

        while True:             if self.processes[new].active:
active_list.append(self.processes[new].pid)                 print(f"Process
{self.processes[old].pid} passes Election({active_list}) to Process
{self.processes[new].pid}")
                old = new               new = (new + 1)
% len(self.processes)               if new ==
initiator:
                break

        new_coordinator = max(active_list)
        print(f"\nProcess {new_coordinator} becomes the new coordinator.")

        old = new_coordinator          new =
(old + 1) % len(self.processes)          while
True:             if
self.processes[new].active:
                print(f"Process {old} passes Coordinator({new_coordinator}) message to Process {new}")
old = new               new = (new + 1) % len(self.processes)               if new == new_coordinator:
print("End Of Election")
                break

# Running the Ring Election ring
= RingElection(5)
ring.start_election(2)
```

bully_algorithm.py ring_algorithm.py

C: > Users > Devansh > Downloads > ring_algorithm.py > ...

```python
class Process:
    def __init__(self, pid):
        self.pid = pid  # Process ID
        self.active = True  # Process is active

class RingElection:
    def __init__(self, total_processes):
        self.processes = [Process(i) for i in range(total_processes)]

    def fetch_maximum(self):
        max_id = -1
        index = 0
        for i, p in enumerate(self.processes):
            if p.active and p.pid > max_id:
                max_id = p.pid
                index = i
        return index

    def start_election(self, initiator):
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS Filter Code

```
[Running] python -u "c:\Users\Devansh\Downloads\ring_algorithm.py"

Process 4 fails.

Election initiated by Process 2
Process 2 passes Election([2, 3]) to Process 3
Process 3 passes Election([2, 3, 0]) to Process 0
Process 0 passes Election([2, 3, 0, 1]) to Process 1

Process 3 becomes the new coordinator.
Process 3 passes Coordinator(3) message to Process 0
Process 0 passes Coordinator(3) message to Process 1
Process 1 passes Coordinator(3) message to Process 2
End Of Election

[Done] exited with code=0 in 0.134 seconds
```

Conclusion:

In this experiment, we successfully implemented and analyzed two leader election algorithms: Bully Algorithm and Ring Algorithm. The Bully Algorithm ensures that the highest-priority process becomes the coordinator, even if failures occur, while the Ring Algorithm follows a circular election process where the highest active process is elected. Both methods efficiently handle distributed system coordination, ensuring fault tolerance and proper leader selection.