Name : Vaishank Mistry course : DCC Panel : C

PRN : 1032212268 ROLL no : 58 (01)

Lab Assignment no. 5 (Bully Algorithm) (2nd)

* Title : Write a C program to implement Bully Election Algorithm

* Aim : To study and implement Bully Election Algorithm.

* Theory :

Elections in Distributed Systems

* In a distributed system, many processes often need to cooperate to perform a task. Sometimes, a single process is required to take on a special role as a coordinator or leader. This coordinator might be responsible for tasks like managing access to a shared resource (mutual exclusion), sequencing operations, or acting as a central point of contact

* The problem arises when this coordinator process fails or becomes unreachable. To ensure the system continues to function correctly, the remaining processes must elect a new coordinator. This process of selecting a new leader is called an election. Election algorithms are protocols that guide this selection process, ensuring that a new, single leader is chosen and all active processes agree on who the new leader is.

Bully Algorithm

* The Bully Algorithm, proposed by Hector Garcia-Molina, is a classic method for electing a leader. It works by assuming that every process in the system has a unique priority number or ID. The core principle is simple: the active process with the highest ID becomes the new coordinator.

* The algorithm is triggered when a process notices that the current leader is no longer responding. The name "Bully" comes from its behaviour. If a process with a lower ID tries to declare itself the leader, a process with a higher ID will challenge it and take over the election process, effectively "bullying" the lower ID into submission.

* The algorithm proceeds as follows :-

1) Election initiation: when a process P sends an ELECTION message to all other processes that have a higher ID than itself, after detecting the coordinator / leader has failed, thus starting the election process.

2) Sending Election Messages: Process P sends an election message to all other processes that have a higher ID than itself.

3) Waiting for a response :

* Scenario A, No Response : If 'P' sends 'ELECTION' messages and recieves no 'RESPONSE' within a set timeout period, it assumes that all higher-ID processes have failed. It declares itself the new coordinator and broadcasts a 'COORDINATOR' message to all other processes, announcing its victory.

* Scenario B, Recieves a Response : If 'P' recieves a RESPONSE from a process with a higher ID, it means a more "powerful" process is active and will take over the election. P's role in the election is now over, and it simply waits for the final COORDINATOR message.

4) Responding to election message : When process Q recieves an ELECTION message from a lower process-ID P :

* Q sends a RESPONSE message back to P to let it known it is still alive

* Q then starts its own election by sending ELECTION messages to all processes with IDs higher than its own, This step is crucial, as it ensures the "biggest bully" eventually takes over.

This process continues until a process sends out ELECTION messages, and recieves no responses, at which point it becomes the coordinator.

Types of Messages

There are three fundamental message types used in the algorithm

1) Election Messages : Sent by a process to announce an election. It is sent to all processes with a higher ID.

2) Response (OK/Alive) Message : Sent in reply to an ELECTION message. It is sent by a higher ID process to a lower ID process to signal that it is alive and take over the election

3) Coordinator (victory) message : Sent by the winning process after an election has concluded. This message is broadcast to all other processes to inform them of the new coordinator's ID.

FAQ's

Q1) What is the time complexity (best, avg, worst) of the bully Algorithm?

Ans 1) * Time complexity in distributed algorithms is often measured by the number of messages exchanged.

* Worst case : $O(N^2)$

The worst case scenario occurs when the process with the lowest ID detects the coordinators failure and initiates an election. It sends (N-1) election messages to all higher-ID processes. The next-to-lowest ID process then responds and starts its own election, sending (N-2) messages. This continues up the chain, creating a cascade of messages. Thus resulting in a message complexity proportional to the sum of $1+2+\ldots.+(N-1)$, which is $O(N^2)$

* Best case : $O(N)$

The best case scenario happens when the process with the second highest ID is the one to detect the coordinator's failure. It sends a single ELECTION message to the process with the highest ID. The highest ID process with an OK message sends (N-1) coordinator message to all other process to announce its victory. The total number of messages is $1+1+(N-1)$, which is $N+1$, giving us $O(N)$.

* Average case : $O(N^2)$

On average, a process somewhere in the middle will likely initiate the election, leading to a complexity which is also $O(N^2)$, as multiple rounds of elections will still need to take place.

Que 2) why do we have to elect the coordinator process?

Ans 2) * In a distributed system, a coordinator is essential for centralizing certain operations to avoid chaos and inconsistency. Without a coordinator, simple tasks can become incredibly complex. Key responsibilities of a coordinator include :

* Mutual Exclusion: Ensuring that only process can access a critical shared resource at a [redacted] time. The coordinator acts as a gatekeeper.

* Synchronization : Ordering events and transactions across different machines to maintain a consistent state.

* Failure Detection : Acting as a central monitor that checks the health of other processes

* Group Managment : Keeping track of which processes are part of the system (e.g., joining or leaving).

* If the coordinator process fails, the system could stall, produce incorrect results, or become partitioned. Electing a new coordinator is a critical recovery mechanism that allows the system to continue functioning reliably and autonomously without manual intervention.

Que 3) How did the ~~name~~ name of the "Bully" approach come up?

Ans 3) * The name is an analogy for its behaviour. The algorithm operates on the principle that "might makes right", where "might" is determined by process ID.

* whenever a process (P1) starts an election, any process with a higher ID (P2) can immediatly shut down P1's attempt by sending a RESPONSE message. This message essentially communicates "I'm stronger than you, so I will take over." P2 then starts its own election, effectively "bullying" P1 out of the race.

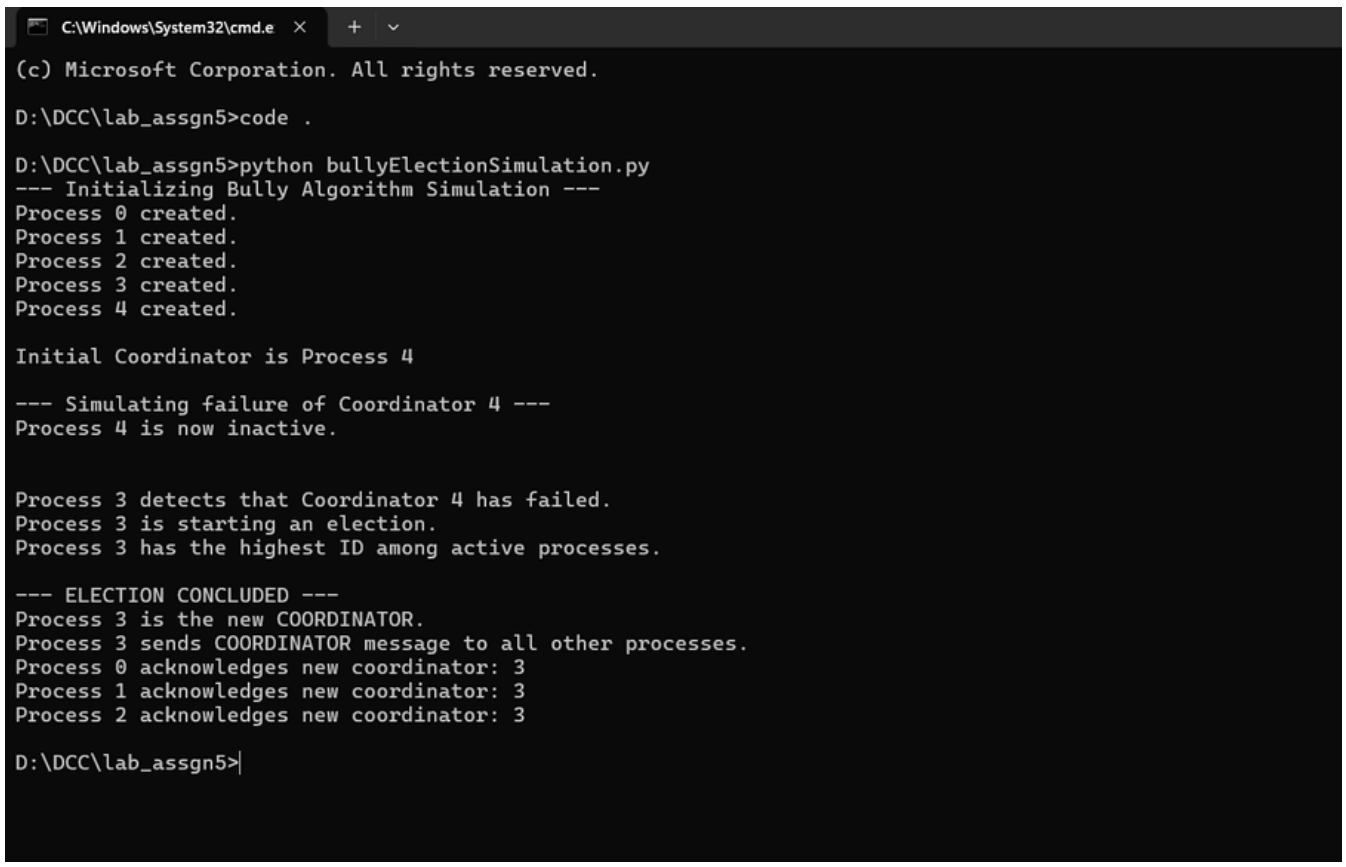
Code

bullyElectionSimulation.py

```python
import time
import random

class Process:
    """
    Represents a single process in the distributed system.
    """
    def __init__(self, process_id, num_processes):
        self.id = process_id
        self.is_active = True
        self.coordinator_id = num_processes - 1 # Assume highest ID is coordinator initially
        self.num_processes = num_processes
        print(f"Process {self.id} created.")

    def set_processes(self, processes_list):
        """Inject the list of all processes for communication."""
        self.processes = processes_list

    def election(self):
        """
        Initiates the election process.
        """
        print(f"Process {self.id} is starting an election.")

        # Send election message to all processes with higher IDs
        higher_processes = [p for p in self.processes if p.id > self.id and p.is_active]

        if not higher_processes:
            # This process has the highest active ID, so it becomes coordinator
            print(f"Process {self.id} has the highest ID among active processes.")
            self.announce_victory()
            return

        print(f"Process {self.id} sends ELECTION messages to: {[p.id for p in higher_processes]}")

        # Simulate waiting for a response
        time.sleep(1)

        responded = False
        for p in higher_processes:
            # Simulate message passing
            if p.receive_election_message(self.id):
                responded = True

        if not responded:
            # No higher process responded, so this process wins
            print(f"Process {self.id} received no responses. It is the new coordinator.")
            self.announce_victory()

    def receive_election_message(self, sender_id):
        """
        Handles an incoming ELECTION message from a lower-ID process.
        """
        if not self.is_active:
            return False

        print(f"Process {self.id} receives ELECTION from Process {sender_id}.")
        # Respond to the sender to let it know a "bully" is active
        print(f"Process {self.id} sends RESPONSE back to Process {sender_id}.")

        # Start its own election
        self.election()
        return True

    def announce_victory(self):
        """
        Announces that this process is the new coordinator.
        """
        print(f"\n--- ELECTION CONCLUDED ---")
        print(f"Process {self.id} is the new COORDINATOR.")

        # Send COORDINATOR message to all other processes
```

```
        print(f"Process {self.id} sends COORDINATOR message to all other processes.")
        for p in self.processes:
            if p.is_active:
                p.receive_coordinator_message(self.id)

        # Update its own coordinator
        self.coordinator_id = self.id

    def receive_coordinator_message(self, new_coordinator_id):
        """
        Handles an incoming COORDINATOR message.
        """
        if self.id != new_coordinator_id:
            print(f"Process {self.id} acknowledges new coordinator: {new_coordinator_id}")
        self.coordinator_id = new_coordinator_id

    def check_coordinator(self):
        """
        Checks if the coordinator is active. If not, starts an election.
        """
        if not self.is_active:
            return

        coordinator_process = self.processes[self.coordinator_id]
        if not coordinator_process.is_active:
            print(f"\nProcess {self.id} detects that Coordinator {self.coordinator_id} has failed.")
            self.election()
        else:
            print(f"Process {self.id} confirms Coordinator {self.coordinator_id} is active.")


def simulate_bully_algorithm(num_processes):
    """
    Main simulation function.
    """
    print("--- Initializing Bully Algorithm Simulation ---")

    # 1. Create all processes
    processes = [Process(i, num_processes) for i in range(num_processes)]
    for p in processes:
        p.set_processes(processes)

    print(f"\nInitial Coordinator is Process {processes[-1].id}\n")

    # 2. Simulate failure of the coordinator
    coordinator_id = num_processes - 1
    print(f"--- Simulating failure of Coordinator {coordinator_id} ---")
    processes[coordinator_id].is_active = False
    print(f"Process {coordinator_id} is now inactive.\n")

    # 3. An active process detects the failure and starts an election
    time.sleep(1)

    # Pick a random active process to start the election
    initiator = None
    while not initiator:
        potential_initiator = random.choice(processes)
        if potential_initiator.is_active:
            initiator = potential_initiator

    initiator.check_coordinator()


if __name__ == "__main__":
    NUM_PROCESSES = 5
    simulate_bully_algorithm(NUM_PROCESSES)
```

Output


C:\Windows\System32\cmd.e
(c) Microsoft Corporation. All rights reserved.
D:\DCC\lab_assgn5>code .
D:\DCC\lab_assgn5>python bullyElectionSimulation.py
--- Initializing Bully Algorithm Simulation ---
Process 0 created.
Process 1 created.
Process 2 created.
Process 3 created.
Process 4 created.
Initial Coordinator is Process 4
--- Simulating failure of Coordinator 4 ---
Process 4 is now inactive.
Process 3 detects that Coordinator 4 has failed.
Process 3 is starting an election.
Process 3 has the highest ID among active processes.
--- ELECTION CONCLUDED ---
Process 3 is the new COORDINATOR.
Process 3 sends COORDINATOR message to all other processes.
Process 0 acknowledges new coordinator: 3
Process 1 acknowledges new coordinator: 3
Process 2 acknowledges new coordinator: 3
D:\DCC\lab_assgn5>