

Adwait Purao - 2021300101 TE Comps B - Batch B

DC LAB

**DISTRIBUTED COMPUTING EXPERIMENT 6**

**Aim:**

Implementation of Election Algorithms

**Theory:**

* Distributed computing involves the coordination and management of multiple processes running on various processors in a networked environment. In many distributed systems, it is essential to have a single process act as a coordinator or leader to ensure organized and efficient operations.
* The challenge lies in determining which process should assume this role, and this is where election algorithms come into play.
* Election algorithms are designed to elect a coordinator process from the currently running processes in a way that ensures there is always a single coordinator at any given time.

The primary goal of an election algorithm is to guarantee that when an election is initiated, it concludes with all the processes reaching a consensus on which process should be the coordinator. This theory section explores the concept of election algorithms and focuses on two specific algorithms: the Bully Algorithm and the Ring Algorithm.

**Election Algorithms**

Coordinator Election Problem

The coordinator election problem is defined as the task of selecting one process from a set of processes running on various processors in a distributed system to serve as the central coordinator. The fundamental requirement is that this process should be elected in a distributed and coordinated manner. In this context, the following assumptions are made:

* Multiple processes are running on distinct machines within the distributed system.
* Peer-to-peer communication: Each process has the capability to communicate with all other processes.
* Each process has a unique ID, with the highest ID being associated with the highest priority.
* A process Pi is denoted with priority i.

**Bully Algorithm**

The Bully Algorithm is one of the approaches designed to solve the coordinator election problem. It operates as follows:

Election Process:

1. When a process Pi initiates communication with the current coordinator, it attempts to elect itself as the new leader if it receives no response from the current coordinator within a specified time (T units).
2. If Pi perceives that the coordinator is absent (no response is received), Pi notifies all processes with higher priorities of an "Election."
3. If no other process responds to the election message, Pi initiates the coordinator code and notifies all processes with lower priorities that "Elected Pi" has occurred.
4. If another process does respond, Pi gracefully steps down, allowing the higher-priority process to take on the role of coordinator. The coordinator change is communicated to all processes with lower priorities.
5. If Pi is not elected as the coordinator, and it receives a message from another process Pj indicating that it has been elected, Pi updates its internal state to acknowledge Pj as the new coordinator.
6. If Pi receives an "election" message from Pj (where i < j), Pi responds to Pj to confirm that it is still alive. Pi then initiates a vote and waits for the new coordinator.

The Bully Algorithm ensures that the process with the highest priority, among those aware of the election, assumes the coordinator role. If a higher-priority process comes into play, it will take over as the coordinator, ensuring the highest priority process is always the coordinator in the system.

**Ring Algorithm**

The Ring Algorithm is another approach for solving the coordinator election problem and operates in a fundamentally different manner. In the Ring Algorithm, processes are arranged in a logical ring, and the coordinator role is passed sequentially from one process to the next in the ring. The process with the highest priority is initially the coordinator.

The election process in the Ring Algorithm proceeds as follows:

1. When an election is initiated, a message is sent along the ring, and each process checks its own priority against the incoming message. If a process with higher priority than the sender exists, it takes over as the new coordinator.
2. If no process with higher priority is found, the message circulates the ring until it reaches the sender again, and that process assumes the coordinator role.

In the Ring Algorithm, the coordinator changes in a cyclic manner, ensuring that the highest-priority process becomes the coordinator and maintains this role until another higher-priority process enters the system.

**1. Bully Algorithm:**

**Code:**

import time

class InventoryManagement:

def \_\_init\_\_(self):

self.inventory = {}

def add\_product(self, id, product, quantity):

if id in self.inventory:

self.inventory[id]['quantity'] += quantity

else:

self.inventory[id] = {'product': product, 'quantity': quantity}

print(f"\nAdded {quantity} {product} to the inventory.")

def subtract\_product(self, id, quantity):

if id in self.inventory and self.inventory[id]['quantity'] >= quantity:

self.inventory[id]['quantity'] -= quantity

print(f"\nSubtracted {quantity} {self.inventory[id]['product']} from the inventory.")

else:

print(f"Unable to subtract {quantity} {id} from the inventory.")

def view\_products(self):

if not self.inventory:

print("The inventory is empty.")

else:

print("ID\tName\tQuantity")

for id in self.inventory:

print(f"{id}\t{self.inventory[id]['product']}\t{self.inventory[id]['quantity']}\n")

class BullyElectionAlgorithm:

def \_\_init\_\_(self, process\_id, all\_processes):

self.process\_id = process\_id

self.coordinator = max(all\_processes)

self.all\_processes = all\_processes

self.election\_in\_progress = False

def send\_election\_message(self, target\_process\_id):

print(f"Process {self.process\_id} sends election message to Process {target\_process\_id}")

def send\_coordinator\_message(self, target\_process\_id):

print(f"Process {self.process\_id} sends coordinator message to Process {target\_process\_id}")

def initiate\_election(self):

if not self.election\_in\_progress:

self.election\_in\_progress = True

for process in self.all\_processes:

if process > self.process\_id:

self.send\_election\_message(process)

def handle\_election\_message(self, sender\_process\_id):

if sender\_process\_id > self.process\_id:

print(f"Process {self.process\_id} acknowledges Process {sender\_process\_id}")

self.initiate\_election()

else:

self.send\_coordinator\_message(sender\_process\_id)

self.coordinator = sender\_process\_id

print(f"Process {self.process\_id} becomes the coordinator.")

self.election\_in\_progress = False

def election\_result(self):

return self.coordinator

if \_\_name\_\_ == "\_\_main\_\_":

all\_processes = [1, 2, 3, 4, 5]

default\_coordinator = max(all\_processes)

print("\n-------------------------------------------------\n")

print("Process 1 started")

print("Process 2 started")

print("Process 3 started")

print("Process 4 started")

print("Process 5 started")

print("\n-------------------------------------------------\n")

print(f"Process {default\_coordinator} chosen as Coordinator")

election\_algorithm = BullyElectionAlgorithm(default\_coordinator, all\_processes)

inventory\_management = InventoryManagement()

while True:

print("\n-------------------------------------------------\n")

print("1. Get Election Result")

print("2. Simulate Leader Failure and Election")

print("3. Add Product")

print("4. Subtract Product")

print("5. View All Products")

print("6. Quit\n")

choice = int(input("Enter your choice: "))

print("-------------------------------------------------")

if choice == 1:

current\_coordinator = election\_algorithm.election\_result()

print(f"Current Coordinator: Process {current\_coordinator}")

elif choice == 2:

current\_coordinator = election\_algorithm.election\_result()

print(f"Current Coordinator: Process {current\_coordinator}")

leader\_failure = int(input("Enter the process ID to simulate leader failure:"))

print(f"Process {current\_coordinator} has failed.")

for i in range(leader\_failure+1, current\_coordinator+1):

print(f"Process {leader\_failure} sends election message to Process {i}")

newProcessList = []

for i in range(leader\_failure+1, current\_coordinator):

print(f"Process {i} acknowledges Process {leader\_failure}")

newProcessList.append(i)

new\_coordinator = max(newProcessList)

for i in range(current\_coordinator+1):

if i != new\_coordinator:

print(f"Process {new\_coordinator} sends co-ordinator message to Process {i}")

election\_algorithm.coordinator = new\_coordinator

print("New leader elected")

elif choice == 3:

id = input("Enter the product id: ")

product = input("Enter the product name: ")

quantity = int(input("Enter the quantity: "))

inventory\_management.add\_product(id, product, quantity)

elif choice == 4:

id = input("Enter the product id: ")

quantity = int(input("Enter the quantity: "))

inventory\_management.subtract\_product(id, quantity)

elif choice == 5:

inventory\_management.view\_products()

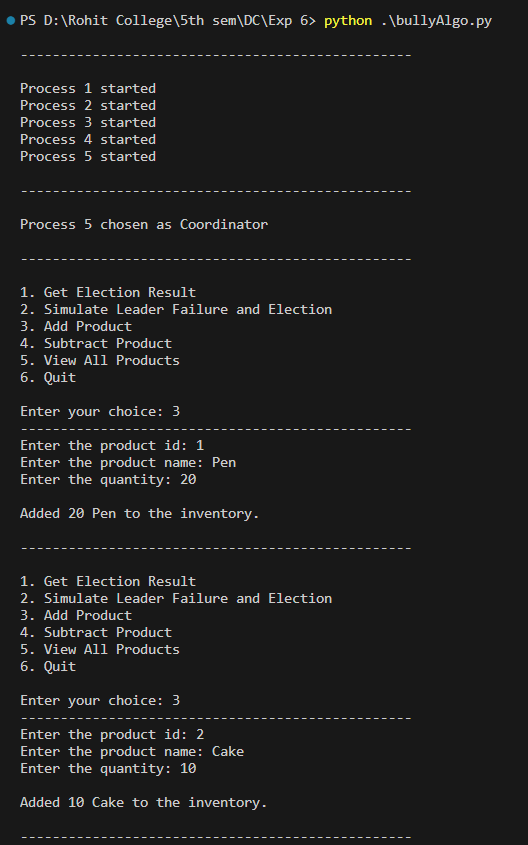
elif choice == 6:

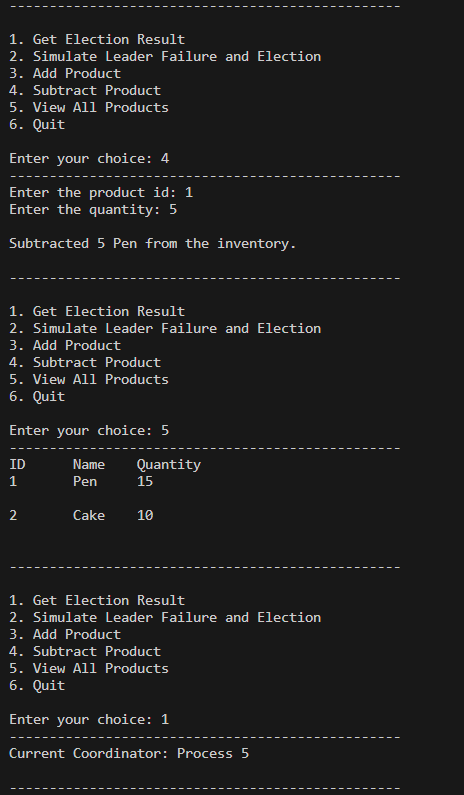
break

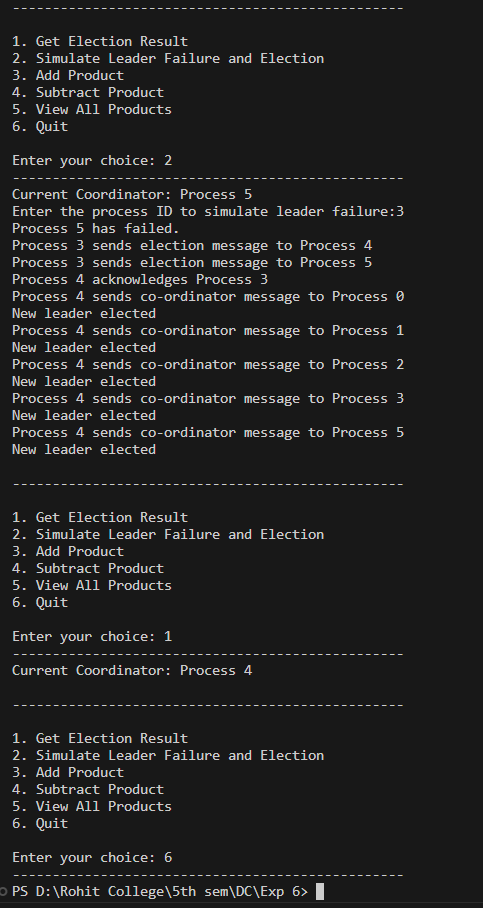
else:

print("Invalid choice. Please select a valid option.")

**Output**







**2. Ring Algorithm:**

**Code:**

class InventoryManagement:

def \_\_init\_\_(self):

self.inventory = {}

def add\_product(self, id, product, quantity):

if id in self.inventory:

self.inventory[id]['quantity'] += quantity

else:

self.inventory[id] = {'product': product, 'quantity': quantity}

print(f"\nAdded {quantity} {product} to the inventory.")

def subtract\_product(self, id, quantity):

if id in self.inventory and self.inventory[id]['quantity'] >= quantity:

self.inventory[id]['quantity'] -= quantity

print(f"\nSubtracted {quantity} {self.inventory[id]['product']} from the inventory.")

else:

print(f"Unable to subtract {quantity} {id} from the inventory.")

def view\_products(self):

if not self.inventory:

print("The inventory is empty.")

else:

print("ID\tName\tQuantity")

for id in self.inventory:

print(f"{id}\t{self.inventory[id]['product']}\t{self.inventory[id]['quantity']}\n")

class RingElectionAlgorithm:

def \_\_init\_\_(self, process\_id, all\_processes):

self.process\_id = process\_id

self.coordinator = max(all\_processes)

self.all\_processes = all\_processes

self.election\_in\_progress = False

self.next\_process\_id = (process\_id + 1) % len(all\_processes)

def send\_election\_message(self):

print(f"Process {self.process\_id} sends election message to Process {self.next\_process\_id}")

def receive\_election\_message(self, sender\_process\_id):

if sender\_process\_id > self.process\_id:

print(f"Process {self.process\_id} acknowledges Process {sender\_process\_id}")

self.coordinator = sender\_process\_id

self.election\_in\_progress = False

print(f"Process {self.process\_id} becomes the coordinator.")

else:

print(f"Process {self.process\_id} forwards election message to Process {self.next\_process\_id}")

self.send\_election\_message()

self.election\_in\_progress = True

def initiate\_election(self):

if not self.election\_in\_progress:

self.election\_in\_progress = True

self.send\_election\_message()

def election\_result(self):

return self.coordinator

if \_\_name\_\_ == "\_\_main\_\_":

all\_processes = [1, 2, 3, 4, 5]

default\_coordinator = max(all\_processes)

print("\n-------------------------------------------------\n")

print("Process 1 started")

print("Process 2 started")

print("Process 3 started")

print("Process 4 started")

print("Process 5 started")

print("\n-------------------------------------------------\n")

print(f"Process {default\_coordinator} chosen as Coordinator")

election\_algorithm = RingElectionAlgorithm(default\_coordinator, all\_processes)

inventory\_management = InventoryManagement()

def ring\_algorithm(election\_algorithm, leader\_failure, all\_processes):

current\_coordinator = election\_algorithm.election\_result()

if current\_coordinator==leader\_failure:

print("\n-------------------------------------------------\n")

print("Same process ID of leader entered")

print("\n-------------------------------------------------\n")

return

print(f"Current Coordinator: Process {current\_coordinator}")

print(f"Process {current\_coordinator} has failed.")

coordinator\_list = []

j=leader\_failure

for i in range(leader\_failure+1,current\_coordinator):

print(f"Message sent from {j} to {i}")

coordinator\_list.append(j)

j=i

# if j<all\_processes[len(all\_processes)-1]:

# coordinator\_list.append(j)

if j<all\_processes[len(all\_processes)-1]:

for i in range(current\_coordinator+1,len(all\_processes)):

print(f"Message sent from {j} to {i}")

coordinator\_list.append(j)

j=i

if all\_processes[0]==leader\_failure:

coordinator\_list.append(j)

print(f"Message sent from {j} to {leader\_failure}")

else:

for i in range(all\_processes[0],leader\_failure+1):

print(f"Message sent from {j} to {i}")

coordinator\_list.append(j)

j=i

print(f"Candidates for leader : ",coordinator\_list)

new\_coo = max(coordinator\_list)

print("New leader elected")

print(f"Process {new\_coo} elected as new leader.")

for i in range(all\_processes[0], all\_processes[len(all\_processes)-1]):

if i != leader\_failure and i != current\_coordinator:

print(f"{leader\_failure} send message to {i}, New leader elected")

election\_algorithm.coordinator = new\_coo

while True:

print("\n-------------------------------------------------\n")

print("1. Get Election Result")

print("2. Simulate Leader Failure and Election")

print("3. Add Product")

print("4. Subtract Product")

print("5. View All Products")

print("6. Quit\n")

choice = int(input("Enter your choice: "))

print("-------------------------------------------------")

if choice == 1:

current\_coordinator = election\_algorithm.election\_result()

print(f"Current Coordinator: Process {current\_coordinator}")

elif choice == 2:

leader\_failure = int(input("Enter the process ID to simulate leader failure:"))

ring\_algorithm(election\_algorithm, leader\_failure, all\_processes)

elif choice == 3:

id = input("Enter the product id: ")

product = input("Enter the product name: ")

quantity = int(input("Enter the quantity: "))

inventory\_management.add\_product(id, product, quantity)

elif choice == 4:

id = input("Enter the product id: ")

quantity = int(input("Enter the quantity: "))

inventory\_management.subtract\_product(id, quantity)

elif choice == 5:

inventory\_management.view\_products()

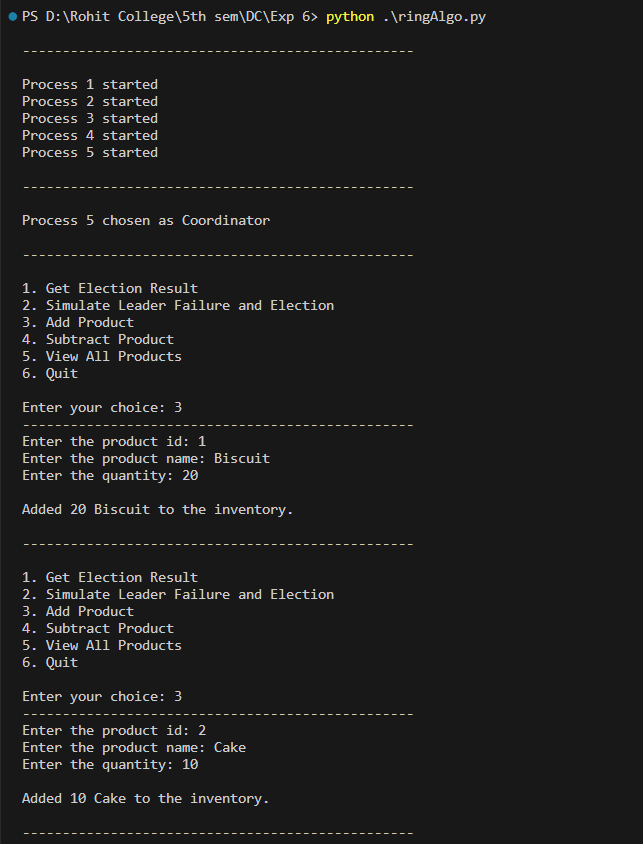
elif choice == 6:

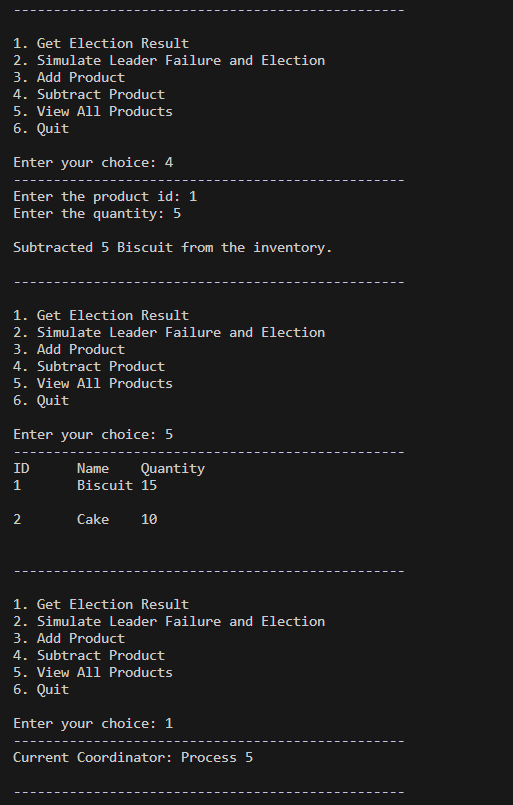
break

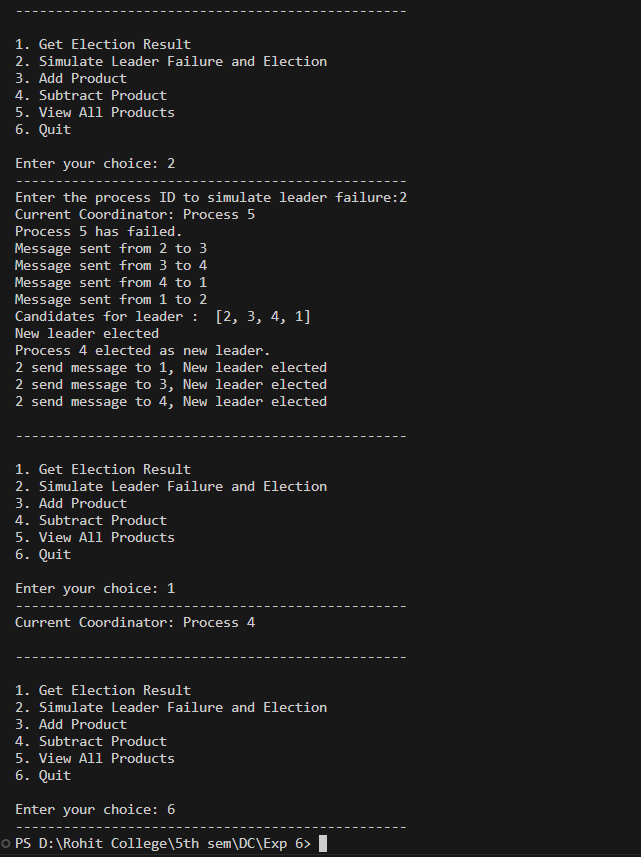
else:

print("Invalid choice. Please select a valid option.")

**Output:**

****

****

****

**Conclusion:**

In summary, our study has provided valuable insights into the significance of election algorithms in the context of distributed systems. Through our examination of the Bully Algorithm and the Ring Algorithm, we've gained an understanding of how these algorithms enable processes to independently select a coordinator in a distributed environment. As demonstrated, these algorithms guarantee the presence of a sole coordinator at all times, ultimately improving the efficiency of operational management in the distributed system.