Chandy-Lamport Algorithm Simulation: Assignment-1.a

Submitted by : Group no. 04

28 - Ciya Sithara K P

53 - Pooja Vijith

54 - Richma Kabeer

69 - Meeraja P

Date : 23/02/2025

1.Introduction :

In distributed systems, ensuring a consistent global state is essential for debugging, checkpointing, and fault tolerance. The Chandy-Lamport Global Snapshot Algorithm is a non-intrusive method for recording a consistent snapshot of a distributed system without disrupting its normal operation.

This report presents a simulation of the Chandy-Lamport algorithm in a distributed system with three or four processes exchanging messages over FIFO channels. The implementation follows the algorithm's principles to capture local states of processes and channel states at the time of the snapshot.

2. Design of the Simulation :

System Assumptions :

- The system consists of three or four processes communicating asynchronously.
- FIFO channels ensure messages are received in the order sent.
- A process can initiate a snapshot after sending a message.
- The snapshot records local states and messages in transit without disrupting message exchanges.

System Components :

- Processes (P1, P2, P3, P4): Each process maintains a balance and exchanges messages.
- Channels (C1→2, C2→3, etc.): Enable unidirectional message transfer.
- Snapshot Mechanism: Uses marker messages to record local and channel states.

3. Steps Followed in the Chandy-Lamport Algorithm :

Test Case 1 (3 Processes: P1, P2, P3)

1. Initial Message Exchanges:
    - P1 transfers 5 units to P2.
    - P2 receives 5 units, updated balance: 25.
    - P2 transfers 10 units to P3.
    - P3 receives 10 units, updated balance: 40.
2. Snapshot Initiation by P1:
    - P1 records its state and sends marker messages to P2.
3. Handling Marker Messages:
    - P2 receives the marker, records its state as 25, and sends a marker to P3.
    - P3 receives the marker, records its state as 40.
4. Post-Snapshot Message Exchanges:
    - P1 transfers 3 units to P2.
    - P2 receives 3 units, updated balance: 28.
    - P2 transfers 7 units to P3.
    - P3 receives 7 units, updated balance: 47.

Test Case 2 (4 Processes: P1, P2, P3, P4)

1. Initial Message Exchanges:
    - P1 transfers 15 units to P2.
    - P2 receives 15 units, updated balance: 215.
    - P2 transfers 25 units to P3.

    - P3 receives 25 units, updated balance: 325.
    - P3 transfers 35 units to P4.
    - P4 receives 35 units, updated balance: 435.
    - P4 transfers 45 units to P1.
    - P1 receives 45 units, updated balance: 145.
2. Snapshot Initiation by P3:
    - P3 records its local state and sends marker messages.
3. Marker Message Handling:
    - P4, P1, and P2 receive markers and record their states.

4. Global Snapshot Results :

Test Case 1 (3 Processes)

- P1: Local State = 10, Channel State = {2: []}
- P2: Local State = 25, Channel State = {3: []}
- P3: Local State = 40, Channel State = {}

Test Case 2 (4 Processes)

- P1: Local State = 145, Channel State = {2: []}
- P2: Local State = 215, Channel State = {3: []}
- P3: Local State = 325, Channel State = {4: []}
- P4: Local State = 435, Channel State = {1: []}

The snapshot correctly captures the system's state when the first marker is sent. No in-transit messages were found at the time of the snapshot.

5.Simulation Details :

The simulation was implemented in Python to model the Chandy-Lamport Global Snapshot Algorithm in a distributed system.

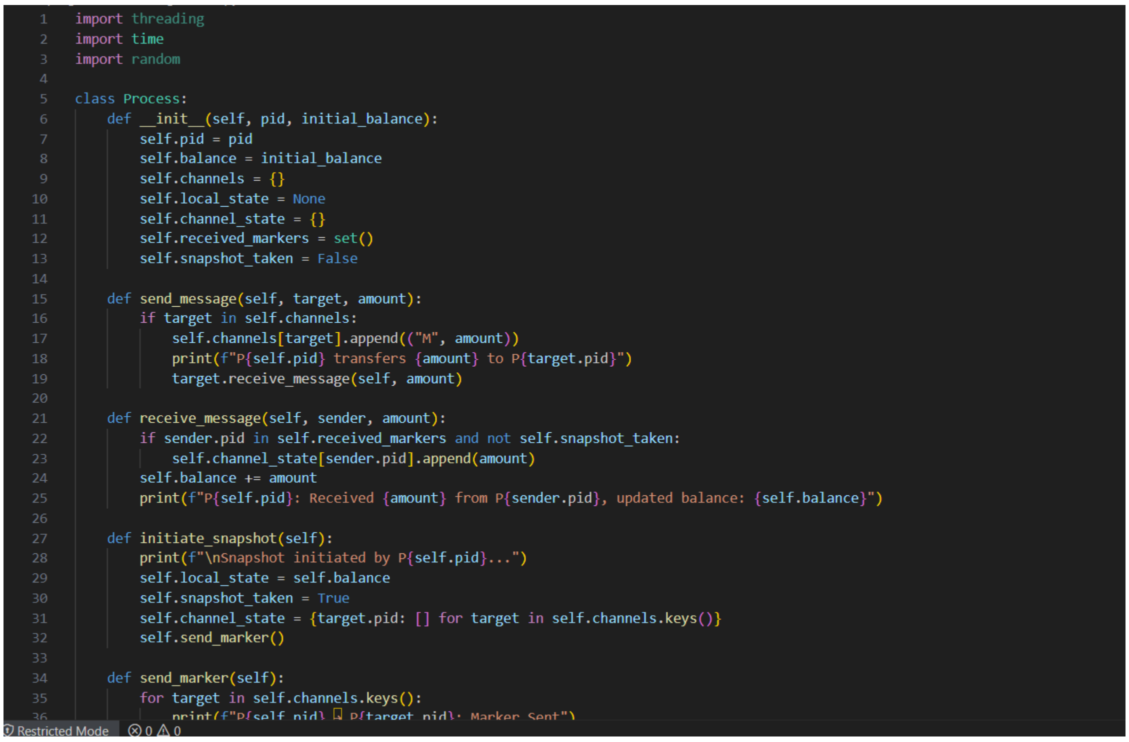
Implementation Steps

1. Process Representation

- Each process is an object with a balance and communication channels.
- Processes send and receive messages, updating balances accordingly.

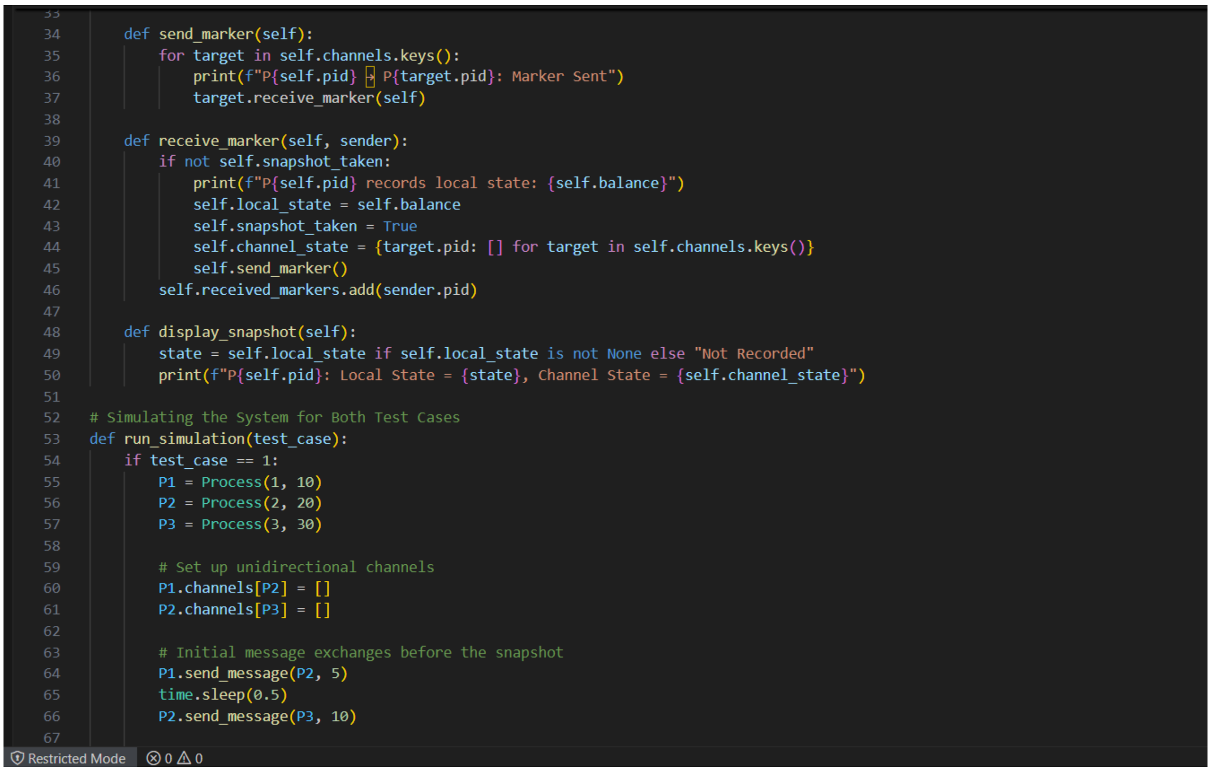
2. Message Passing
   - Transactions simulate real-world distributed messaging.
   - Processes exchange funds before and after snapshot initiation.
3. Snapshot Initiation
   - A process starts the snapshot by sending a marker to connected processes.
   - The marker triggers state recording in receiving processes.
4. State Recording
   - Each process records its balance and channel state.
   - Any in-transit messages before receiving the marker are logged.
5. Final Snapshot Results
   - The global state is displayed, ensuring consistency.
   - The implementation adheres to the Chandy-Lamport algorithm principles.

Code implementation



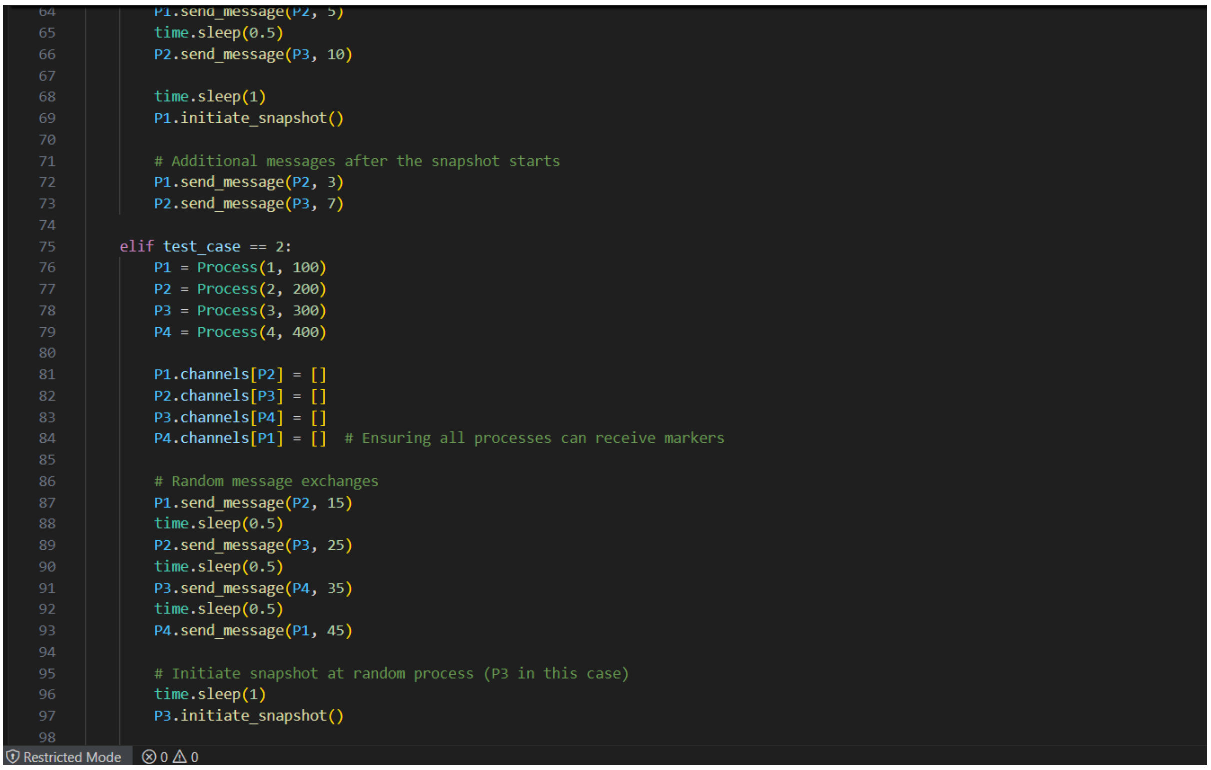
```python
import threading
import time
import random

class Process:
    def __init__(self, pid, initial_balance):
        self.pid = pid
        self.balance = initial_balance
        self.channels = {}
        self.local_state = None
        self.channel_state = {}
        self.received_markers = set()
        self.snapshot_taken = False

    def send_message(self, target, amount):
        if target in self.channels:
            self.channels[target].append(("M", amount))
            print(f"P{self.pid} transfers {amount} to P{target.pid}")
            target.receive_message(self, amount)

    def receive_message(self, sender, amount):
        if sender.pid in self.received_markers and not self.snapshot_taken:
            self.channel_state[sender.pid].append(amount)
        self.balance += amount
        print(f"P{self.pid}: Received {amount} from P{sender.pid}, updated balance: {self.balance}")

    def initiate_snapshot(self):
        print(f"\nSnapshot initiated by P{self.pid}...")
        self.local_state = self.balance
        self.snapshot_taken = True
        self.channel_state = {target.pid: [] for target in self.channels.keys()}
        self.send_marker()

    def send_marker(self):
        for target in self.channels.keys():
            print(f"P{self.pid} → P{target.pid}: Marker Sent")
```

Restricted Mode ⊗ 0 ⚠ 0



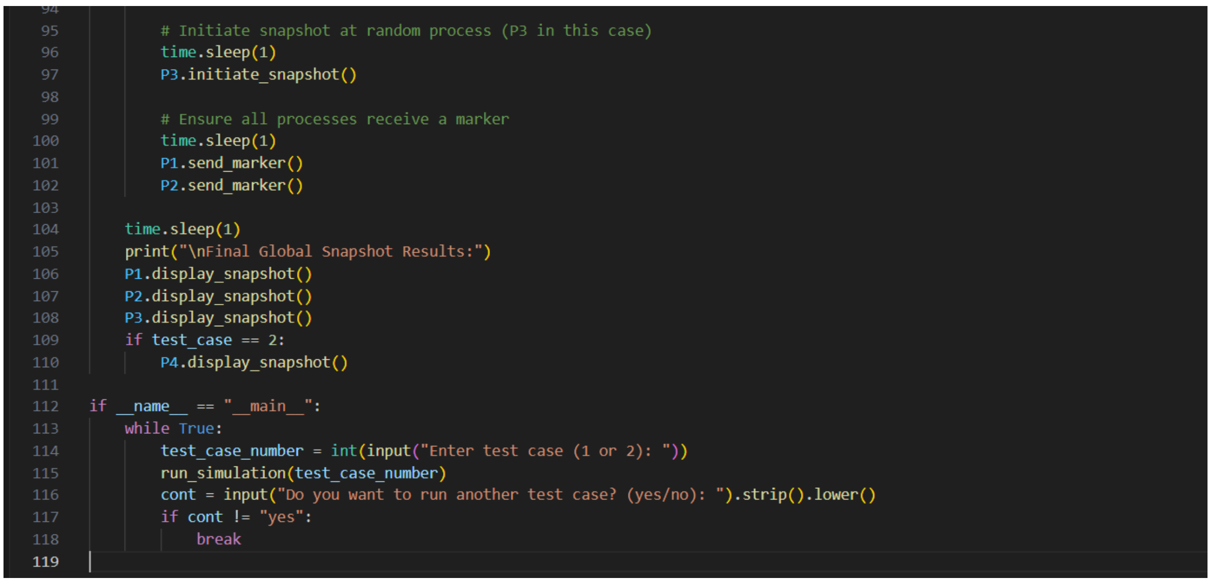
```
    def send_marker(self):
        for target in self.channels.keys():
            print(f"P{self.pid} → P{target.pid}: Marker Sent")
            target.receive_marker(self)

    def receive_marker(self, sender):
        if not self.snapshot_taken:
            print(f"P{self.pid} records local state: {self.balance}")
            self.local_state = self.balance
            self.snapshot_taken = True
            self.channel_state = {target.pid: [] for target in self.channels.keys()}
            self.send_marker()
        self.received_markers.add(sender.pid)

    def display_snapshot(self):
        state = self.local_state if self.local_state is not None else "Not Recorded"
        print(f"P{self.pid}: Local State = {state}, Channel State = {self.channel_state}")

# Simulating the System for Both Test Cases
def run_simulation(test_case):
    if test_case == 1:
        P1 = Process(1, 10)
        P2 = Process(2, 20)
        P3 = Process(3, 30)

        # Set up unidirectional channels
        P1.channels[P2] = []
        P2.channels[P3] = []

        # Initial message exchanges before the snapshot
        P1.send_message(P2, 5)
        time.sleep(0.5)
        P2.send_message(P3, 10)

```

Restricted Mode ⊗ 0 ⚠ 0



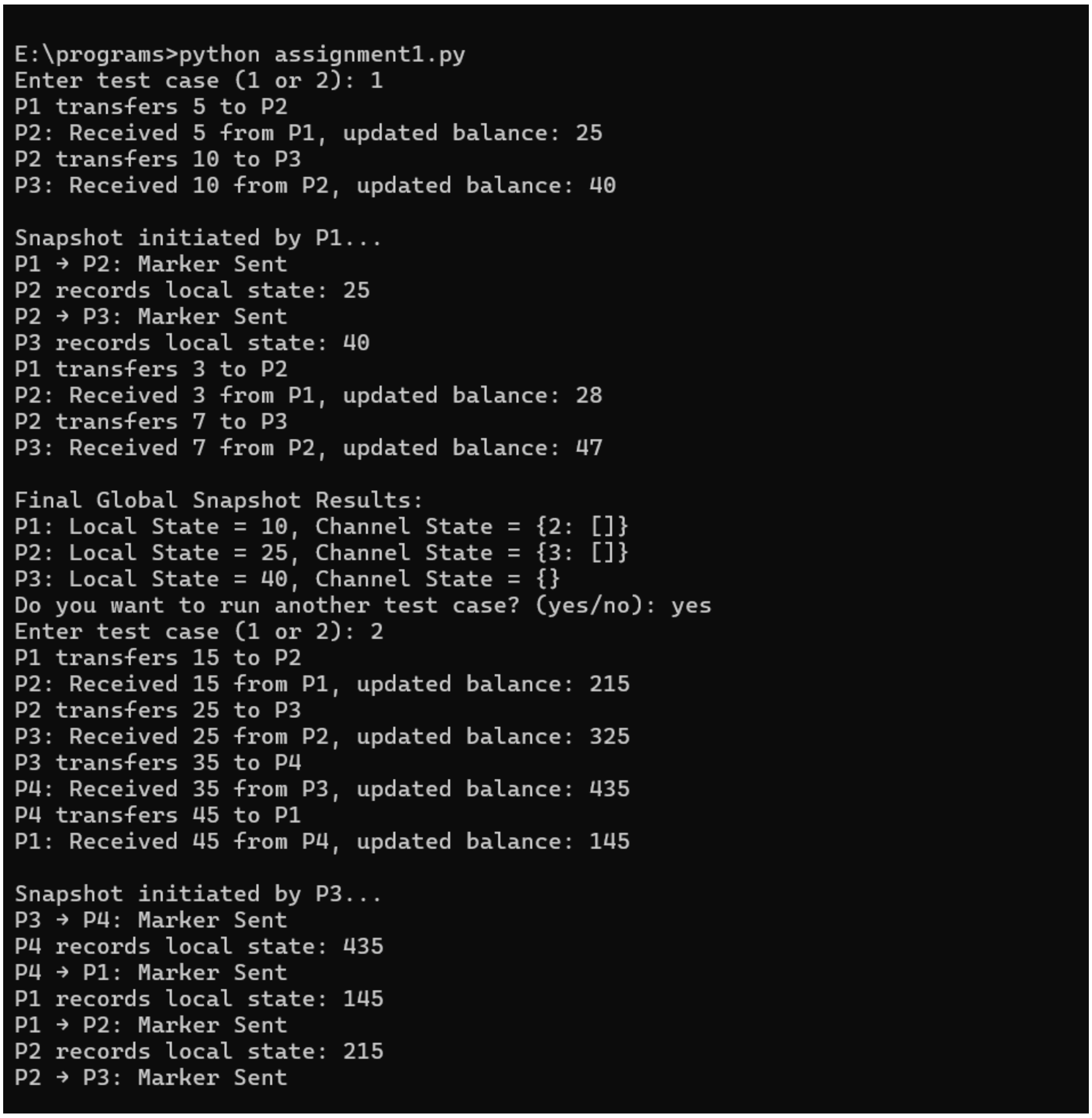
```
        P1.send_message(P2, 5)
        time.sleep(0.5)
        P2.send_message(P3, 10)

        time.sleep(1)
        P1.initiate_snapshot()

        # Additional messages after the snapshot starts
        P1.send_message(P2, 3)
        P2.send_message(P3, 7)

    elif test_case == 2:
        P1 = Process(1, 100)
        P2 = Process(2, 200)
        P3 = Process(3, 300)
        P4 = Process(4, 400)

        P1.channels[P2] = []
        P2.channels[P3] = []
        P3.channels[P4] = []
        P4.channels[P1] = []  # Ensuring all processes can receive markers

        # Random message exchanges
        P1.send_message(P2, 15)
        time.sleep(0.5)
        P2.send_message(P3, 25)
        time.sleep(0.5)
        P3.send_message(P4, 35)
        time.sleep(0.5)
        P4.send_message(P1, 45)

        # Initiate snapshot at random process (P3 in this case)
        time.sleep(1)
        P3.initiate_snapshot()

```

Restricted Mode ⊗ 0 ⚠ 0

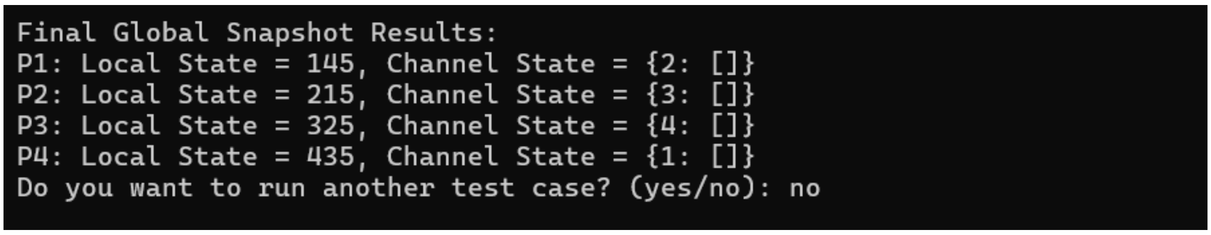


```python
        # Initiate snapshot at random process (P3 in this case)
        time.sleep(1)
        P3.initiate_snapshot()

        # Ensure all processes receive a marker
        time.sleep(1)
        P1.send_marker()
        P2.send_marker()

    time.sleep(1)
    print("\nFinal Global Snapshot Results:")
    P1.display_snapshot()
    P2.display_snapshot()
    P3.display_snapshot()
    if test_case == 2:
        P4.display_snapshot()

if __name__ == "__main__":
    while True:
        test_case_number = int(input("Enter test case (1 or 2): "))
        run_simulation(test_case_number)
        cont = input("Do you want to run another test case? (yes/no): ").strip().lower()
        if cont != "yes":
            break

```

6. Screenshots or Logs of the Simulation Output :



```
E:\programs>python assignment1.py
Enter test case (1 or 2): 1
P1 transfers 5 to P2
P2: Received 5 from P1, updated balance: 25
P2 transfers 10 to P3
P3: Received 10 from P2, updated balance: 40

Snapshot initiated by P1...
P1 → P2: Marker Sent
P2 records local state: 25
P2 → P3: Marker Sent
P3 records local state: 40
P1 transfers 3 to P2
P2: Received 3 from P1, updated balance: 28
P2 transfers 7 to P3
P3: Received 7 from P2, updated balance: 47

Final Global Snapshot Results:
P1: Local State = 10, Channel State = {2: []}
P2: Local State = 25, Channel State = {3: []}
P3: Local State = 40, Channel State = {}
Do you want to run another test case? (yes/no): yes
Enter test case (1 or 2): 2
P1 transfers 15 to P2
P2: Received 15 from P1, updated balance: 215
P2 transfers 25 to P3
P3: Received 25 from P2, updated balance: 325
P3 transfers 35 to P4
P4: Received 35 from P3, updated balance: 435
P4 transfers 45 to P1
P1: Received 45 from P4, updated balance: 145

Snapshot initiated by P3...
P3 → P4: Marker Sent
P4 records local state: 435
P4 → P1: Marker Sent
P1 records local state: 145
P1 → P2: Marker Sent
P2 records local state: 215
P2 → P3: Marker Sent
```



```
Final Global Snapshot Results:
P1: Local State = 145, Channel State = {2: []}
P2: Local State = 215, Channel State = {3: []}
P3: Local State = 325, Channel State = {4: []}
P4: Local State = 435, Channel State = {1: []}
Do you want to run another test case? (yes/no): no
```

7. Conclusion :

The Chandy-Lamport algorithm was successfully implemented and tested. The global snapshot results correctly capture the state of the system, ensuring a consistent and accurate representation of process balances and in-transit messages.