Chandy–Lamport Algorithm for Consistent State Capture in a Distributed System

1. Introduction

In a distributed system, multiple processes execute concurrently and communicate through message passing. Due to the absence of a global clock, capturing the global state of such a system at a single instant is non-trivial. The Chandy–Lamport algorithm provides a method to record a consistent global snapshot without stopping the execution of the system. This snapshot consists of the local state of each process and the state of communication channels (messages in transit).

2. System Requirements

- A distributed system consisting of multiple processes (nodes).
- Processes communicate only through message passing.
- Communication channels are reliable and follow FIFO order.
- Each process can record its local state.
- A special control message called MARKER is available.
- The system continues normal execution during snapshot recording.

3. Algorithm Process

- Any one process can initiate the snapshot.
- The initiator records its local state and sends a MARKER message on all outgoing channels.
- When a process receives a MARKER for the first time, it records its local state and forwards MARKERs on its outgoing channels.
- Messages received on a channel after recording local state but before receiving a MARKER on that channel are recorded as channel state.
- When MARKERs have been received on all incoming channels, the snapshot for that process is complete.

4. Python Implementation

```
import threading
import time
from queue import Queue
from copy import deepcopy

NORMAL = "NORMAL"
MARKER = "MARKER"

class Message:
    def __init__(self, msg_type, content=None):
        self.type = msg_type
        self.content = content

class Process:
    def __init__(self, pid, initial_state):
        self.pid = pid
        self.state = initial_state
        self.in_channels = {}
        self.out_channels = {}
        self.snapshot_taken = False
        self.local_snapshot = None
        self.channel_snapshots = {}

    def add_out_channel(self, target, channel):
        self.out_channels[target] = channel

    def add_in_channel(self, source, channel):
        self.in_channels[source] = channel
        self.channel_snapshots[source] = []

    def send_message(self, target, value):
        msg = Message(NORMAL, value)
        self.out_channels[target].put(msg)
        print(f"P{self.pid} sends {value} to P{target}")

    def start_snapshot(self):
        print(f"\nP{self.pid} initiates SNAPSHOT")
        self.record_local_state()
        for channel in self.out_channels.values():
            channel.put(Message(MARKER))

    def record_local_state(self):
        self.snapshot_taken = True
        self.local_snapshot = deepcopy(self.state)
        print(f"P{self.pid} records local state: {self.local_snapshot}")

    def receive(self, source, message):
        if message.type == MARKER:
            self.handle_marker(source)
        else:
            self.handle_normal(source, message)

    def handle_marker(self, source):
        print(f"P{self.pid} received MARKER from P{source}")
        if not self.snapshot_taken:
            self.record_local_state()
            self.channel_snapshots[source] = []
            for channel in self.out_channels.values():
                channel.put(Message(MARKER))

    def handle_normal(self, source, message):
        if self.snapshot_taken:
            self.channel_snapshots[source].append(message.content)
        self.state += message.content
        print(f"P{self.pid} received {message.content} from P{source} | New state: {self.state}")

def channel_listener(source, target, channel):
    while True:
        if not channel.empty():
            msg = channel.get()
            target.receive(source, msg)
        time.sleep(0.1)

if __name__ == "__main__":
    n = int(input("Enter number of processes: "))
    processes = {}

    for i in range(1, n + 1):
        state = int(input(f"Enter initial state of P{i}: "))
        processes[i] = Process(i, state)

    for i in range(1, n + 1):
        for j in range(1, n + 1):
            if i != j:
                ch = Queue()
                processes[i].add_out_channel(j, ch)
                processes[j].add_in_channel(i, ch)
                threading.Thread(
                    target=channel_listener,
                    args=(i, processes[j], ch),
                    daemon=True
                ).start()

    m = int(input("Enter number of messages to send before snapshot: "))
    for _ in range(m):
        src = int(input("Sender process ID: "))
        dst = int(input("Receiver process ID: "))
        val = int(input("Message value (+/-): "))
        processes[src].send_message(dst, val)
        time.sleep(0.5)

    initiator = int(input("Enter snapshot initiator process ID: "))
    processes[initiator].start_snapshot()

    time.sleep(3)

    print("\n===== GLOBAL SNAPSHOT RESULT =====")
    for p in processes.values():
        print(f"Process P{p.pid}")
        print(f"Local State: {p.local_snapshot}")
        for src, msgs in p.channel_snapshots.items():
            print(f"Channel from P{src}: {msgs}")
```

5. Sample Input and Output

```
Sample Input:
Enter number of processes: 3
Enter initial state of P1: 500
Enter initial state of P2: 300
Enter initial state of P3: 700
Enter number of messages to send before snapshot: 2
Sender process ID: 1
Receiver process ID: 2
Message value (+/-): 50
Sender process ID: 2
Receiver process ID: 3
Message value (+/-): -30
Enter snapshot initiator process ID: 1

Sample Output:
P1 sends 50 to P2
P2 received 50 from P1 | New state: 350
P2 sends -30 to P3
P3 received -30 from P2 | New state: 670
P1 initiates SNAPSHOT
P1 records local state: 500
P2 records local state: 350
P3 records local state: 670

===== GLOBAL SNAPSHOT RESULT =====
Process P1
Local State: 500
Process P2
Local State: 350
Process P3
Local State: 670
```