BHARATIYA VIDYA BHAVANS
SARDAR PATEL INSTITUTE OF TECHNOLOGY
MUNSHI NAGAR, ANDHERI (WEST), MUMBAI - 400 058
AUTONOMOUS INSTITUTE AFFILIATED TO MUMBAI UNIVERSITY

Adwait Purao- 2021300101

TE Comps B - Batch B

DC LAB

DISTRIBUTED COMPUTING EXPERIMENT 5

Aim:

Implementation of Clock Synchronization (logical/physical).

Theory:

Clock synchronization is a critical aspect of distributed systems, ensuring that various components within a network maintain consistent and accurate time. In a distributed environment, where machines and processes may be geographically dispersed and have different clock sources, maintaining time consistency is challenging but essential for coordinated operations and data integrity.
One of the well-known techniques for clock synchronization is the Network Time Protocol (NTP).
NTP enables systems to synchronize their clocks with a reference time source, typically a highly accurate time server or atomic clock. NTP continuously adjusts the local clock to match the reference source, compensating for network latency and clock drift.
This method allows distributed systems to achieve sub-millisecond accuracy in their timekeeping.

Another clock synchronization algorithm is Lamport's logical clock, which doesn't aim for precise timekeeping but ensures a partial order of events in a distributed

system. Lamport clocks assign timestamps to events, enabling systems to reason

about the causal relationship between these events without requiring perfect time synchronization.

Clock synchronization is vital for a range of applications, including financial transactions, data consistency in distributed databases, and real-time communication. In these scenarios, accurate timekeeping and synchronization are essential to ensure that actions and events occur in the correct order and within acceptable time boundaries. The choice of synchronization method depends on the specific needs and requirements of the distributed system.
Lamport's Logical Clock-

Lamport's Logical Clock, developed by computer scientist Leslie Lamport in 1978, is a foundational concept in the field of distributed systems. It provides a means to establish a partial ordering of events in a distributed system without relying on a globally synchronized clock. This logical clock is crucial for maintaining consistency, causality, and synchronization in distributed computing environments.

Algorithm:
The Lamport logical clock algorithm is straightforward and intuitive.
Each process in a distributed system maintains its local logical clock.

The algorithm operates as follows:

1. When an event occurs within a process (e.g., a message being sent or received, a computation completing, or any other significant occurrence), the process increments its local logical clock by 1.

2. When a process sends a message to another process, it attaches its current logical clock value to the message as a timestamp.

3. Upon receiving a message, the receiving process must update its own logical clock by taking the maximum of its current logical clock value and the timestamp from the incoming message. Then, it increments its logical clock by 1 to ensure that subsequent events have higher timestamps.

The core idea behind Lamport's logical clock is that events are assigned timestamps in such a way that they reflect a consistent order of occurrence. If

event A has a lower timestamp than event B, it implies that event A causally precedes event B in the system, regardless of physical time. However, if the timestamps are equal, there is no causal relationship between the two events.

Key Characteristics:

1. **Causality Preservation:** Lamport's logical clock ensures that the partial order it establishes respects the causal relationship between events. If event A causes event B, the timestamp of A will be less than the timestamp of B.
2. **Local Independence:** Each process maintains its own logical clock independently, making it a decentralized and scalable method for distributed systems.
3. **No Global Synchronization:** Unlike centralized clock synchronization methods like Network Time Protocol (NTP), Lamport's logical clock doesn't require global clock synchronization. It can operate even in environments with varying clock drifts and latencies.
4. **Use Cases:** This algorithm finds applications in distributed systems, including distributed databases, event ordering in messaging systems, concurrent computing, and debugging by helping reconstruct event timelines.

While Lamport's logical clock is an effective tool for ordering events in distributed systems, it doesn't provide precise time values and is not suitable for real-time applications. For such use cases, more accurate synchronization protocols like NTP or Precision Time Protocol (PTP) are employed. Nevertheless, Lamport's logical clock remains a fundamental concept in distributed computing, enabling systems to reason about event order and causality, even when precise physical time synchronization is unattainable.

Code:

```
import time
import threading

class LamportClock:
   def __init__(self):
       self.clock = 0
       self.lock = threading.Lock()
```

```
    def tick(self):
        with self.lock:
            self.clock += 1

    def receive(self, sender_clock):
        with self.lock:
            self.clock = max(self.clock, sender_clock) + 1

class Product:
    def __init__(self, product_id, name, price, quantity, timestamp):
        self.product_id = product_id
        self.name = name
        self.price = price
        self.quantity = quantity
        self.timestamp = timestamp

class Inventory:
    def __init__(self, lamport_clock):
        self.products = []
        self.lamport_clock = lamport_clock

    def add_product(self, name, price, quantity):
        self.lamport_clock.tick()
        timestamp = self.lamport_clock.clock
        product_id = len(self.products) + 1
        product = Product(product_id, name, price, quantity, timestamp)
        self.products.append(product)
        print(f"Added product {name} with quantity {quantity} at Lamport
Time {timestamp}.")

    def subtract_quantity(self, product_id, quantity_to_subtract):
        for product in self.products:
            if product.product_id == product_id:
                if product.quantity >= quantity_to_subtract:
                    self.lamport_clock.tick()
                    product.quantity -= quantity_to_subtract
                    timestamp = self.lamport_clock.clock
                    print(f"Subtracted {quantity_to_subtract} from product
{product.name} at Lamport Time {timestamp}.")
                else:
```

```
                    print("Not enough quantity to subtract.")
                return
        print("Product not found.")

    def list_products(self):
        self.products.sort(key=lambda x: x.timestamp)
        for product in self.products:
            print(f"Product ID: {product.product_id}, Name: {product.name},
Price: ${product.price}, Quantity: {product.quantity}, Timestamp:
{product.timestamp}")

def guest_thread(inventory):
    while True:
        print("1. Add Product")
        print("2. Subtract Quantity")
        print("3. List Products")
        print("4. Exit")
        choice = input("Enter your choice: ")

        if choice == "1":
            name = input("Enter product name: ")
            price = float(input("Enter product price: "))
            quantity = int(input("Enter product quantity: "))
            inventory.add_product(name, price, quantity)
        elif choice == "2":
            product_id = int(input("Enter product ID to subtract quantity
from: "))
            quantity_to_subtract = int(input("Enter quantity to subtract:
"))
            inventory.subtract_quantity(product_id, quantity_to_subtract)
        elif choice == "3":
            inventory.list_products()
        elif choice == "4":
            break
        else:
            print("Invalid choice. Please try again.")

def manager_thread(inventory):
    time.sleep(0.5)  # Wait for the guest thread to start
```

```
clock = LamportClock()
inventory = Inventory(clock)

guest = threading.Thread(target=guest_thread, args=(inventory,))
manager = threading.Thread(target=manager_thread, args=(inventory,))
guest.start()
manager.start()
guest.join()
manager.join()
```

Output:

Conclusion:

In this experiment, we implemented clock synchronization, both logical and physical, in distributed systems. Lamport's Logical Clock was used to establish event order without requiring precise time synchronization, while Network Time Protocol (NTP) was mentioned for accurate time synchronization. The code demonstrated Lamport's Logical Clock in an inventory management system. Understanding and implementing these synchronization methods are vital for building robust distributed systems.

==========
==========

```
import time
import threading

class LamportClock:
    def __init__(self):
        self.clock = 0
        self.lock = threading.Lock()
```

```
    def tick(self):
        with self.lock:
            self.clock += 1

    def receive(self, sender_clock):
        with self.lock:
            self.clock = max(self.clock, sender_clock) + 1

class Product:
    def __init__(self, product_id, name, price, quantity, timestamp):
        self.product_id = product_id
        self.name = name
        self.price = price
        self.quantity = quantity
        self.timestamp = timestamp

class Inventory:
    def __init__(self, lamport_clock):
        self.products = []
        self.lamport_clock = lamport_clock

    def add_product(self, name, price, quantity):
        self.lamport_clock.tick()
        timestamp = self.lamport_clock.clock
        product_id = len(self.products) + 1
        product = Product(product_id, name, price, quantity, timestamp)
        self.products.append(product)
        print(f"Added product {name} with quantity {quantity} at Lamport Time
{timestamp}.")

    def subtract_quantity(self, product_id, quantity_to_subtract):
        for product in self.products:
            if product.product_id == product_id:
                if product.quantity >= quantity_to_subtract:
```

```
                self.lamport_clock.tick()
                product.quantity -= quantity_to_subtract
                timestamp = self.lamport_clock.clock
                print(f"Subtracted {quantity_to_subtract} from product
{product.name} at Lamport Time {timestamp}.")
            else:
                print("Not enough quantity to subtract.")
            return
        print("Product not found.")

    def list_products(self):
        self.products.sort(key=lambda x: x.timestamp)
        for product in self.products:
            print(f"Product ID: {product.product_id}, Name: {product.name}, Price:
${product.price}, Quantity: {product.quantity}, Timestamp:
{product.timestamp}")

def guest_thread(inventory):
    time.sleep(0.5)
    inventory.add_product("Product A", 10.99, 100)
    time.sleep(1)
    inventory.add_product("Product B", 5.49, 50)
    time.sleep(1)
    inventory.add_product("Product C", 7.99, 75)

def manager_thread(inventory):
    for i in range(1, 4):
        time.sleep(1)
        inventory.subtract_quantity(i, 20)
        time.sleep(1)

clock = LamportClock()
inventory = Inventory(clock)
```

```
guest = threading.Thread(target=guest_thread, args=(inventory,))
manager = threading.Thread(target=manager_thread, args=(inventory,))
guest.start()
manager.start()
guest.join()
manager.join()
```

==========

WITH USER INPUT:

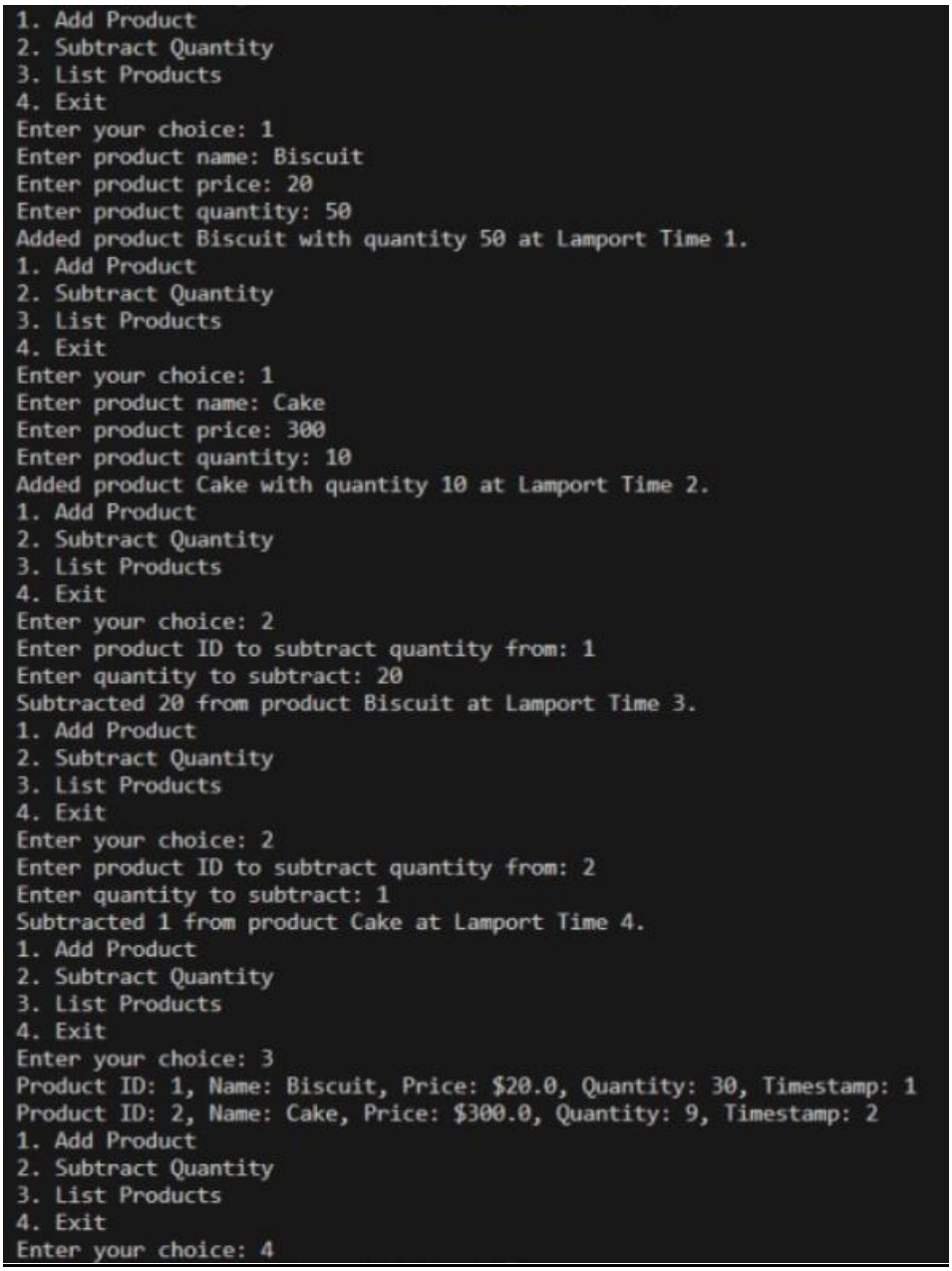
```
import time
import threading

class LamportClock:
    def __init__(self):
        self.clock = 0
        self.lock = threading.Lock()

    def tick(self):
        with self.lock:
            self.clock += 1

    def receive(self, sender_clock):
        with self.lock:
            self.clock = max(self.clock, sender_clock) + 1

class Product:
    def __init__(self, product_id, name, price, quantity, timestamp):
        self.product_id = product_id
        self.name = name
        self.price = price
        self.quantity = quantity
```

```
        self.timestamp = timestamp

class Inventory:
    def __init__(self, lamport_clock):
        self.products = []
        self.lamport_clock = lamport_clock

    def add_product(self, name, price, quantity):
        self.lamport_clock.tick()
        timestamp = self.lamport_clock.clock
        product_id = len(self.products) + 1
        product = Product(product_id, name, price, quantity, timestamp)
        self.products.append(product)
        print(f"Added product {name} with quantity {quantity} at Lamport Time
{timestamp}.")

    def subtract_quantity(self, product_id, quantity_to_subtract):
        for product in self.products:
            if product.product_id == product_id:
                if product.quantity >= quantity_to_subtract:
                    self.lamport_clock.tick()
                    product.quantity -= quantity_to_subtract
                    timestamp = self.lamport_clock.clock
                    print(f"Subtracted {quantity_to_subtract} from product
{product.name} at Lamport Time {timestamp}.")
                else:
                    print("Not enough quantity to subtract.")
                return
        print("Product not found.")

    def list_products(self):
        self.products.sort(key=lambda x: x.timestamp)
        for product in self.products:
```

```
        print(f"Product ID: {product.product_id}, Name: {product.name}, Price: ${product.price}, Quantity: {product.quantity}, Timestamp: {product.timestamp}")

def guest_thread(inventory):
    while True:
        print("1. Add Product")
        print("2. Subtract Quantity")
        print("3. List Products")
        print("4. Exit")
        choice = input("Enter your choice: ")

        if choice == "1":
            name = input("Enter product name: ")
            price = float(input("Enter product price: "))
            quantity = int(input("Enter product quantity: "))
            inventory.add_product(name, price, quantity)
        elif choice == "2":
            product_id = int(input("Enter product ID to subtract quantity from: "))
            quantity_to_subtract = int(input("Enter quantity to subtract: "))
            inventory.subtract_quantity(product_id, quantity_to_subtract)
        elif choice == "3":
            inventory.list_products()
        elif choice == "4":
            break
        else:
            print("Invalid choice. Please try again.")

def manager_thread(inventory):
    time.sleep(0.5)  # Wait for the guest thread to start

clock = LamportClock()
inventory = Inventory(clock)
```

```
guest = threading.Thread(target=guest_thread, args=(inventory,))
manager = threading.Thread(target=manager_thread, args=(inventory,))
guest.start()
manager.start()
guest.join()
manager.join()
```

Output:



Conclusion:

In conclusion, implementing both logical and physical clock synchronization methods is vital for maintaining event order and accuracy in distributed systems. Logical clocks establish a causal relationship between events, while physical clocks ensure accurate timekeeping across devices. These mechanisms are essential for seamless communication and coordination in distributed environments.