Experiment 04: Write a program to implement group communication in distributed computing

Learning Objective: Student should be able to implement group communication in distributed computing
Tools: Python
Theory: Group communication refers to the exchange of information and ideas among members of a group. It can take place through various channels and involve multiple participants. In the context of programming, group communication often involves coordinating and exchanging data among different components or processes.

Types of Group Communication:

1. Broadcast Communication:
Definition: In broadcast communication, messages are sent to all members of the group.
Programming Context: In programming, broadcast communication often involves sending information or signals to all components or processes within a system. This is particularly useful for scenarios where global updates or notifications need to be disseminated to all participants.
Example: In a distributed system, a server might broadcast a message to all connected clients to notify them of a system-wide event, such as server maintenance or an important update.

2. Multicast Communication:
Definition: In multicast communication, messages are sent to a specific subset of the group.
Programming Context: Multicast is beneficial when you want to target a specific group of participants rather than the entire set. This is useful for scenarios where certain components or processes share common interests or responsibilities.
Example: In a multiplayer online game, multicast communication could be used to send updates only to players in a specific region or those involved in a particular in-game event.

3. Unicast Communication:
Definition: In unicast communication, messages are sent between two specific members of the group.
Programming Context: Unicast is similar to traditional one-to-one communication. It is commonly used for direct communication between two components or processes within a group.
Example: In a peer-to-peer network, unicast communication might occur between two nodes exchanging specific data, such as file transfers or real-time chat messages.

Additional Considerations:

Message Queues: Many group communication implementations involve the use of message queues. Processes or components can publish messages to a queue, and subscribers receive messages from the queue based on their interest or topic.

Reliability: Depending on the application, you might need to consider the reliability of communication. For instance, using acknowledgment mechanisms or ensuring message delivery order may be crucial in certain scenarios.

Scalability: Group communication mechanisms should be scalable to accommodate a growing number of participants. This involves considerations of system architecture and the chosen communication patterns.

Security: When designing group communication, security measures such as encryption and authentication should be considered, especially if sensitive data is being exchanged.

Code:

```
import multiprocessing

def worker_function(worker_id, shared_data):
    print(f"Worker {worker_id} received: {shared_data.value}")

if __name__ == "__main__":
    # Shared data among processes
    shared_data = multiprocessing.Value('i', 10)

    # Creating multiple processes
    num_workers = 3
    processes = []

    for i in range(num_workers):
        process = multiprocessing.Process(target=worker_function, args=(i, shared_data))
        processes.append(process)
        process.start()

    # Broadcasting data to all processes
    # shared_data.value = 42

    # Waiting for all processes to finish
    for process in processes:
        process.join()
```

Output:

```
Worker 0 received: 10
Worker 1 received: 10
Worker 2 received: 10
```

For Faculty Use

Correction Parameters	Formative Assessment [40%]	Timely completion of Practical [40%]	Attendance / Learning Attitude [20%]	
Marks Obtained				