Experiment No. 4 Grouped Communication

Aim: Develop a program for Grouped Communication

Tools: VSCode, Python, MSWord

Theory: In **distributed computing**, **grouped communication types** refer to different ways multiple processes or nodes communicate in a distributed system. The main types include:

1. Unicast Communication

- One-to-one communication between two processes.
- Example: A client sending a request to a specific server.

2. Multicast Communication

- One-to-many communication where a message is sent to a specific group of nodes.
- Example: A streaming service distributing video data to subscribed users.

3. Broadcast Communication

- One-to-all communication where a message is sent to every node in the system.
- Example: A network router sending an ARP request to all devices.

4. Anycast Communication

- One-to-one-of-many communication, where a message is sent to the nearest or best-available node from a group.
- Example: A DNS request being routed to the closest DNS server.

5. Gossip Communication

- A probabilistic, peer-to-peer message-passing approach where nodes communicate with random neighbors to spread information gradually.
- Example: Blockchain networks propagating transaction data.

6. Reliable Group Communication

- Ensures message delivery guarantees in distributed systems using techniques like acknowledgment and consensus.

Server (`server.py`)

```python
import asyncio
import websockets

clients = set()

async def handler(websocket):
    clients.add(websocket)
    print(f"Client connected: {websocket.remote_address}")

    try:
        async for message in websocket:
            print(f"Received from client: {message}")  # Show received
            for client in clients:
                await client.send(f"Broadcast: {message}")  # Send to a
    finally:
        clients.remove(websocket)
        print(f"Client disconnected: {websocket.remote_address}")

async def main():
    async with websockets.serve(handler, "localhost", 8765):
        print("Server started on ws://localhost:8765")
        await asyncio.Future()  # Keeps the server running

asyncio.run(main())
```

Client (`client.py`)

```python
import asyncio
import websockets

async def listen():
    async with websockets.connect("ws://localhost:8765") as websocket:
        print("Connected to server!")

        # Send a test message
        await websocket.send("Hello, this is a test message!")

        while True:
            message = await websocket.recv()
            print(f"Received: {message}")

asyncio.run(listen())
```

Server Output

```
Server started on ws://localhost:8765
```

Client Output

```
Connected to server!
Received: Broadcast: Hello, this is a test message!
```

Conclusion:

This WebSocket-based broadcast system enables real-time one-to-many communication, where messages from any client are instantly sent to all connected clients. It's a simple yet effective way to implement live updates in distributed applications.

For Faculty Use:

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