E-510 Disttibuted Systems Assignment-2

Pseudocode Overview:

Given below is the pseudocode for totally Ordered Broadcast. Where the block represents a process which internally has a middleware and an application. Each middleware and application has ports from where they can communicate with each other and the network. The application has a port on which it listens for the incoming messages from the middleware for processing. Also, there is a port on which the application emits the messages that the application generates.

The middleware has total of two ports. One port is where it listens to incoming requests from the application and broadcasts that message over the network. Another port is where it listens for incoming messages from the network here the messages are divided in two parts one is a normal message and another is the acknowlegement where acknowledgement messages are just for the messages in the queue. After reciveing the message it is added in the queue and sorted according to the lamport timestamps. Now the clock is adjusted and another port is where the message is transmitted to the application for processing. The queue thread is where the processing happens here it is checked that the message that is at the top of queue has recieved all acknowledgements then it is processed else we deliver the ack messages over the network indicating that the message is at the top of the queue for the current process.

Pseudocode

Application

```
def sendDataToMiddleware(message): # Here the message is generated and sent to this function
    sendMessageToMiddleware(message,port1) # Will connect to the middleware on port1 and send the message

def processMessageFromMiddleware(): # Will listen on a port and recieve the messages sent by the middleware
    while listen(port2)
        if message: # If we get a message we process it otherwise we continously check for incoming messages
            processMessage(message) # This will process the message

def run(): # Each time we call this method to start the application
    Thread(processMessageFromMiddleware).start()
```

Middleware

```
# Here we listen on port1 and we check for any message that is delivered from application to middleware
def recieveMessageFromApplication():
    while listen(port1):
        if message: # If we get a message we broadcast it
            updateClock(blockId) # We update the clock as soon as we recieve the message by the block Id number
            broadcastMessage(message) # We send broadcast for the message we recieved from the application

def broadcastMessage(message):
    for port in allApplicationNetworkRecievePorts: # We loop over all other blocks network receiving port
        sendOnPort(port,message) # We send the message on all blocks network receiving port

def updateClock(number):
    clock += number

def addAcknowledgement(message):
    for msg in queue: # iterate over all messages in the queue
        if message.hashValue == msg.hashValue: # If the hashvalue of the message and the ack matches
            msg.AddAcknowledgement() # Decreases the acknowledgement counter from the message in the queue

def recieveFromNetwork():
    while listen(port3): # Listen on the network recieve port.
        if message:
            # Updating the clock to maximum of current clock or the clock we got from the message
            clock = max(clock+blockId,message.clock)
            # If the message is acknowledgement message then subtract the remaining acknowlegements
            if message.isAckMessage():
                addAcknowledgement(message)
                continue
            queue.add(message)
            # After updating the clock we add the message to queue which would sort
            # the message by message.clock. This queue is processed in the seperate thread

def sendToApplication(message):
    sendOnPort(port2,message)

def processQueue():
    while True: # Continously process the elements in the top of the queue
        # If the message on the top of the queue is having 0 remaning acknoweldgements
        # remaining then send it to the application layer
        if queue.getTopMessage().RemainingAcknowledgements() == 0:
            sendToApplication(queue.getTopMessage())
            queue.removeMessage() # Will remove the top message from the queue
        else:
            # Will give ack message and broadcast ack message indicating that the
            # message is ready to be processed
            broadcastMessage(queue.getTopMessage().getAckMessage())

def run():# Each time we call this method to start the middleware
    Thread(recieveFromNetwork).start()
    Thread(recieveMessageFromApplication).start()
    Thread(processQueue).start()
```

Queue

```
def add(message):
    # This method would compare the time and would keep the message with lowest time at the front
    pass

def getTopMessage():
    return queue[0] # Return the first element of the queue

def removeMessage():
    # Remove the top message and bring the next lowest clock time message on the top
    pass
```

Message

```
# Message class has below properties:
#   1. message
#   2. clock
#   3. hashValue
#   4. blockId
#   5. RemAcks

def computeHash():
    # computes a unique hash value using message clock and blockId which would be used for acks
    pass

def AddAcknowledgement():
    # this would decrement the remacks counter
    pass

def getAckMessage():
    # Returns the ack message for the current message
    return ack
```

Block

```
def spawnApplication():
    Application(portsData).run()

def spawnMiddleware():
    Middleware(portsData).run()
```

```
def run():
    Thread(spawnApplication).start()
    Thread(spawnMiddleware).start()
```