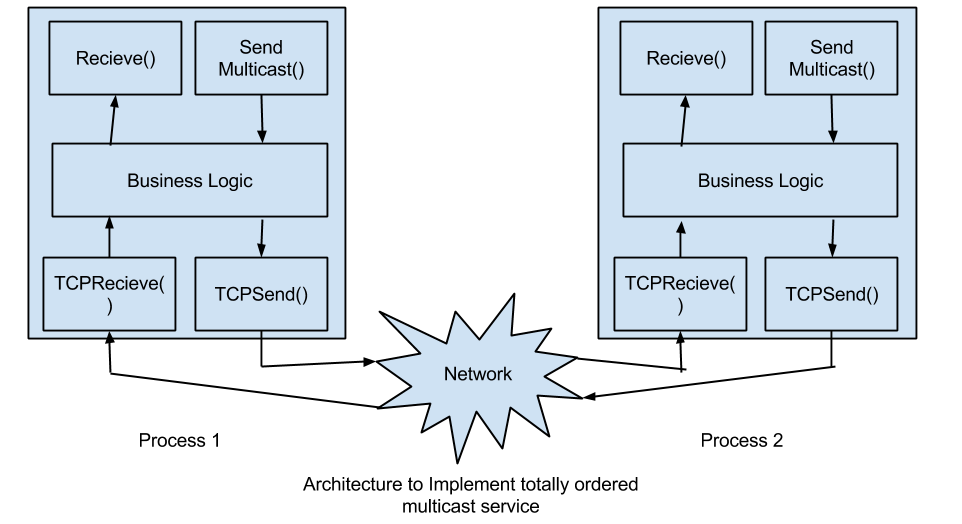
# Totally Ordered Multicast Service

## Design Document

1. **Implementation Details**: Each Program is divided into two modules, first to implement the application and other to implement totally ordered multicast service. The overall architecture is shown here:

\*In picture I have shown TCP IP but in code it has been changed to SCTP.



Each process creates a new thread, which continuously keeps listening to the messages received and the current thread is used to send messages and display GUI.

In order to achieve total ordered multicast, we have identified 3 types of messages:

* + 1. **Data Message** (Msg type -0): This is the multicast data message, which sender multicasts to receivers.
    2. **Control Message 1**(Msg type-1): This is the clock reply from receiver to sender, having being received Data Message.
    3. **Control Message 2**(Msg type -2): This is the multicast message from sender and contains the maximum clock value.

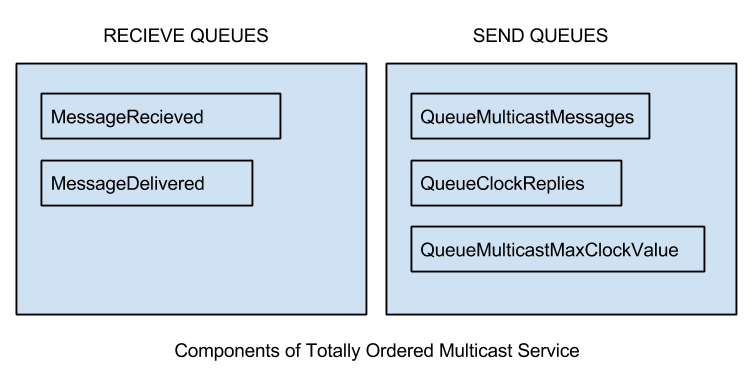
Now to manage these messages, we have following queues in our architecture:

* + 1. Send Queues:
       1. QueueMulticastMessages
       2. QueueClockReplies
       3. QueueMulticastMaxClockValue

From these send queues, Sender picks the elements from queues based on priority. Highest priority is to QueueClockReplies then QueueMulticastMaxClockValue and then QueueMulticastMessages.

So the send function is continuously monitoring these queues to send messages.

* + 1. Receive Queues:
       1. MessageRecieved (Priority Queue)
       2. MessageDelivered



Once the message is received, then following actions may be performed:

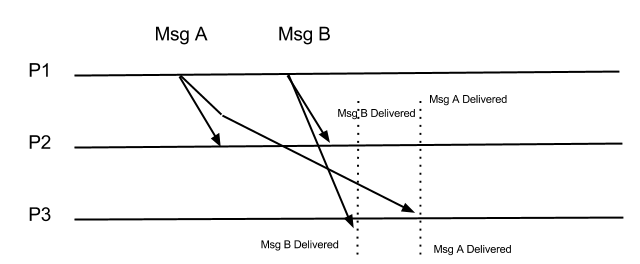
* + - 1. If(Msg type =0), then
         * Message is pushed into MessageRecieved.
         * A new Control message with current Clock Value is pushed into QueueClockReplies.
      2. If(Msg type =1 && received clock replies from all recipient processes of multicast) , then
         * A new Control message with maximum Clock Value is pushed into QueueClockReplies.
      3. If(Msg type =2), then
         * Remove all messages from MessageRecieved whose clock value < clock value recieved.
         * Push all these messages into messagesDelivered.

1. **Testing**: Python script that compares the log files of messages received and look for reverse links or cycles in messages

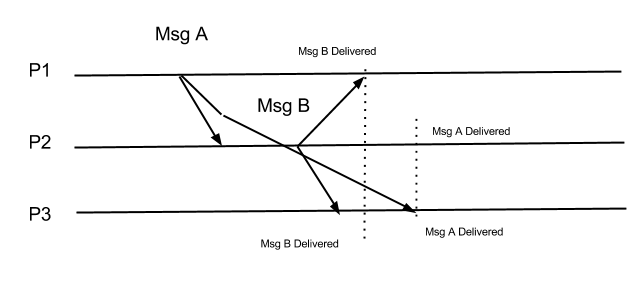
Hence this Testing instance makes sure that , If process Pi sends messages x, y to processes Pj, Pk , then all the processes Pj, Pk … receive the messages in the same order (x,y or y,x).

For example:

* 1. Instance 1: In the picture shown below, Msg B is send after Msg A from same process, but our protocol ensures that both messages are delivered in same order at P2 and P3.



* 1. Instance 2: In the picture shown below, Msg B is send after Msg A but from different processes, but our protocol ensures that both messages are delivered in same order at P1, P2 and P3.



Hence in both the cases our testing instance has to ensure that order in which the messages are received is same.

This way of testing increases the message complexity, but it eliminates the task of verifying the logs by a script making the testing fully automated.