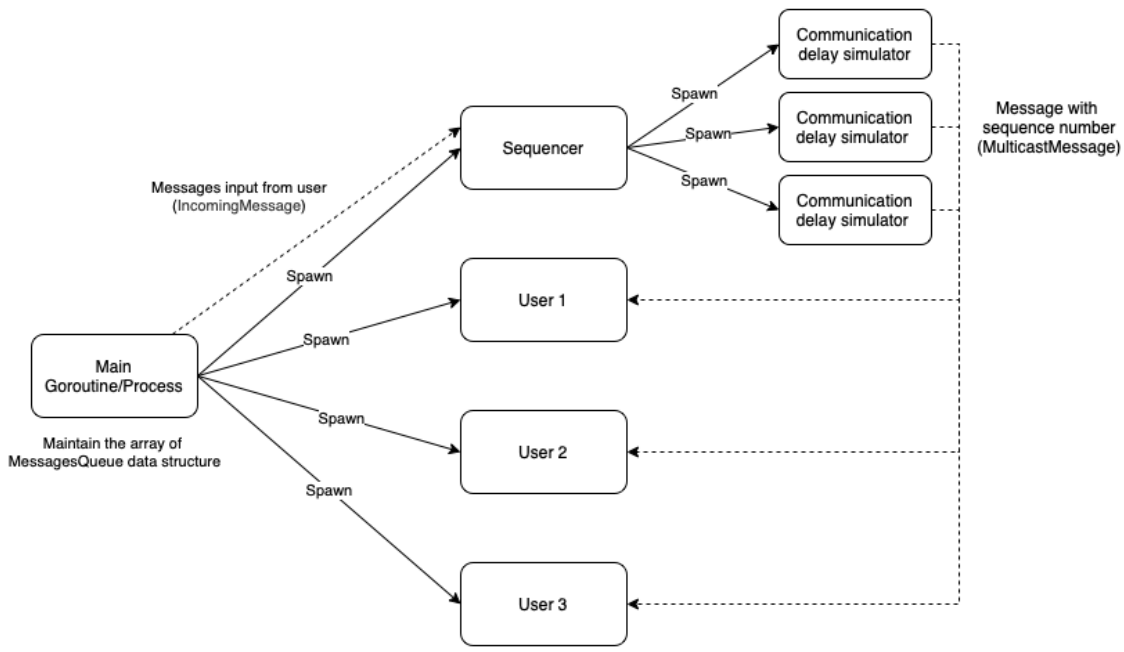
The proposed solution

In order to ensure that every message will be displayed to the users in the same order regardless of the delay in communication. Our solution implements the concept of total ordering in multicast ordering. Therefore, we must have a sequencer that assigns the sequence number of each message before sending multicast to the users. Each user will 2 things to keep track of. First, the local sequence number initially starts at zero. Second, the messages buffer to store the messages that are not received in the order. They must follow the same protocol such that the message will display if and only if the local sequence number + 1 is the sequence number of the incoming message. Otherwise, the message will be put in the messages buffer. Once the message has been displayed, they will check in the messages buffer again for the message that satisfies the displaying condition (local sequence number + 1 is equal to the sequence number of the message). For this solution implementation, the message will be coming from only one user and sent to the sequencer. In reality, the messages can be coming from every user but all of them have to be sent to the sequencer first. Thus, we decided to let the user input the messages first before sending those messages to the sequencer to start the simulation.

The implement of the solution

Our solution is implemented in Golang due to the following reasons.

1. Golang support concurrency with Goroutine which is the higher abstraction level of thread and process
2. It has many tools to control the concurrency when entering critical sections such as Mutex and WaitGroup.
3. The communication between Goroutine can be done easily using Channel.



Data Structures

```
type MulticastMessage struct {
    seq         uint
    message     string
    messageType string
}
```

The **MulticastMessage** struct defined the data structure of the data that the sequencer sent to each user/process. It contains the message, message type, and sequence number assigned by the sequencer.

```
type IncomingMessage struct {
    message     string
    messageType string
}
```

IncomingMessage struct represents the message sent from the main goroutine to the sequencer.

```
type MessagesQueue struct {
    incomingMessage IncomingMessage
    waitDuration    time.Duration
    timestamp       time.Time
}
```

MessagesQueue defined the data structure of the recorded input message from the user given thorough console input. It has `waitDuration` to tell the main goroutine the interval time between this message and the previous message.

Main Goroutine

The main function is the entry point where the Go application is compiled and run. It responsible for the following tasks

1. Printing the instruction message to the console
2. Create communication channels from the main Goroutine to the sequencer and sequencer to all of the users. The total number of Channels is 4 (3 Users).
3. Receiving message and type of message from the console
4. Spawning n number of user/process (3 is the default) and a sequencer in another Goroutine with appropriate parameters.
5. Sending all the messages to the sequencer regarding the order and time interval between each message.
6. Wait for every message are delivered and displayed at each user
7. Terminate the execution once done.

Sequencer

The purpose of the sequencer is to assign the message with a running sequence number and send them out to each user. It has only one variable which is the global sequence number initially starting at 0. The sequencer is running in an infinite loop that waiting for a new message (IncomingMessage data structure) to come in through Channel. Once the message comes, the global sequence variable is increased by 1, and **IncomingMessage** is transformed into **MulticastMessage**. The **MulticastMessage** is sent to each user through Channels. For simulating the communication time, we spawn a `sendMulticastMessage` function another Goroutine for

every message sending process. It is responsible for determining communication time to the user and using `time.Sleep()` function to simulate the communication time delay. We decided to spawn another Goroutine to do this job because of using `time.Sleep()` function would block the **IncomingMessage** from Main Goroutine to coming into the sequencer and also block the message sending process from sequencer to another user as well.

User

Each user is the same function that is running in the difference Goroutine. It maintains the local sequence number in the integer variable and the array of **MulticastMessage** for the messages buffer. The user is running in an infinite loop. It waiting for an incoming message from the Channel and processes that message. When the message comes in, the message sequence number is compared with the local sequence number. If the message sequence number equals to local sequence number + 1, then the message will be displayed to the console and the local sequence number is increased by 1. After that, the buffer is checked if there is more message that satisfies the displaying condition. On the other hand, if the message sequence number is not equal to the local sequence number + 1, that message is put in the buffer. After that, the whole buffer is sorted by message sequence number for easier comparison.

Test scenarios

According to our solution, there are 3 users (user1, user2, and user3), has only one sender that send message to sender, and there are 3 types of message (text, image, video).

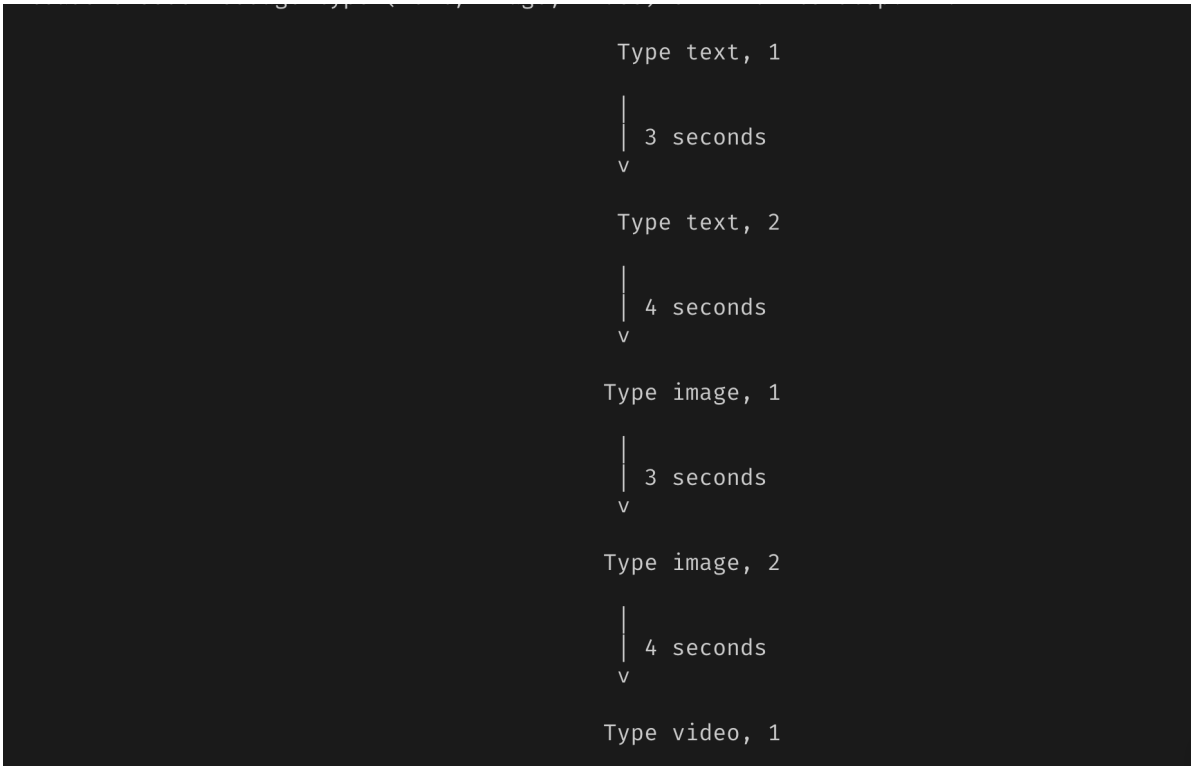
Total communication time of each message = communication time of message type + random communicate time (range between 0-5 seconds). Note that communication time is randomed when sequencer send message to each user (it might be different)

1. Text (1 second + random communication time)
2. Image (5 seconds + random communication time)
3. Video (10 seconds + random communication time)

We come up with 3 test cases.

1. There is no message in buffer

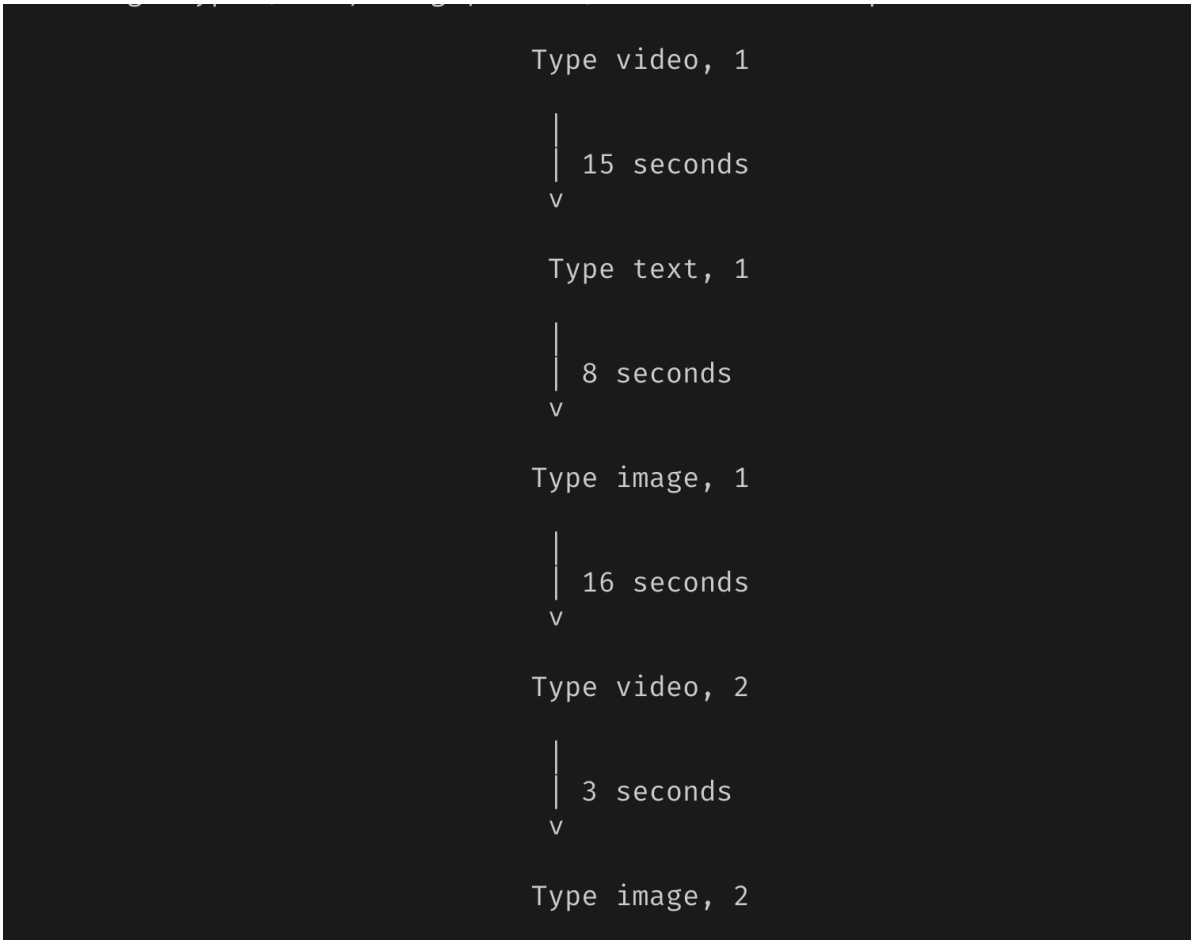
```
Please choose message type (Text, Image, Video) or 'End' to stop: Text
Please enter text message: 1

Please choose message type (Text, Image, Video) or 'End' to stop: Text
Please enter text message: 2

Please choose message type (Text, Image, Video) or 'End' to stop: Image
Please enter image title: 1

Please choose message type (Text, Image, Video) or 'End' to stop: Image
Please enter image title: 2

Please choose message type (Text, Image, Video) or 'End' to stop: Video
Please enter video title: 1

Please choose message type (Text, Image, Video) or 'End' to stop: End
```



Type	Message	Waiting time before sending next message	Communication Time
Text1	1	3 Second	1 second + random time
Text	2	4 Second	1 second + random time
Image	1	3 Second	5 second + random time
Image	2	4 Second	5 second + random time
Video	1		10 second + random time

2. There is one message in buffer

```
Please choose message type (Text, Image, Video) or 'End' to stop: Video
Please enter video title: 1

Please choose message type (Text, Image, Video) or 'End' to stop: Text
Please enter text message: 1

Please choose message type (Text, Image, Video) or 'End' to stop: Image
Please enter image title: 1

Please choose message type (Text, Image, Video) or 'End' to stop: Video
Please enter video title: 2

Please choose message type (Text, Image, Video) or 'End' to stop: Image
Please enter image title: 2

Please choose message type (Text, Image, Video) or 'End' to stop: End
```

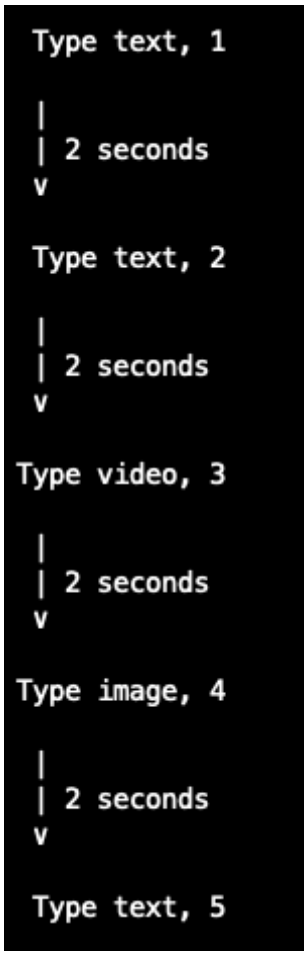


Type	Message	Waiting time before sending next message	Communication Time
Video	1	15 Second	10 second + random time
Text	1	8 Second	1 second + random time
Image	1	16 Second	5 second + random time
Video	2	3 Second	10 second + random time
Image	2		5 second + random time

3. There are many messages in buffer

- Input: sender send five messgaes in the following order

```
Please choose message type (Text, Image, Video) or 'End' to stop: text
Please enter text message: 1

Please choose message type (Text, Image, Video) or 'End' to stop: text
Please enter text message: 2

Please choose message type (Text, Image, Video) or 'End' to stop: video
Please enter video title: 3

Please choose message type (Text, Image, Video) or 'End' to stop: image
Please enter image title: 4

Please choose message type (Text, Image, Video) or 'End' to stop: text
Please enter text message: 5

Please choose message type (Text, Image, Video) or 'End' to stop: end
```



Type	Message	Waiting time before sending next message / end	Communication Time
Text	1	2 seconds	1 second + random time
Text	2	2 seconds	1 second + random time
Video	3	2 seconds	10 seconds + random time
Image	4	2 seconds	5 seconds + random time
Text	5	2 seconds	1 second + random time

Simulation results

These are the result of the test scenarios

1. There is no message in buffer

```
# Main

 SUCCESS  SENT MESSAGE, Timestamp 1635063336, Type text, 1
 SUCCESS  SENT MESSAGE, Timestamp 1635063339, Type text, 2
 SUCCESS  SENT MESSAGE, Timestamp 1635063343, Type image, 1
 SUCCESS  SENT MESSAGE, Timestamp 1635063346, Type image, 2
 SUCCESS  SENT MESSAGE, Timestamp 1635063350, Type video, 1

## Sequencer
```

Main Thread send the message following by the order to the sequencer.

```
## Sequencer

 INFO  RECEIVED MESSAGE, Timestamp 1635063336, Type text, 1
 SUCCESS  SENT MESSAGE TO USER 1, Timestamp 1635063336, Seq 1, Type text, 1
 SUCCESS  SENT MESSAGE TO USER 2, Timestamp 1635063336, Seq 1, Type text, 1
 SUCCESS  SENT MESSAGE TO USER 3, Timestamp 1635063336, Seq 1, Type text, 1
 INFO  RECEIVED MESSAGE, Timestamp 1635063339, Type text, 2
 SUCCESS  SENT MESSAGE TO USER 1, Timestamp 1635063339, Seq 2, Type text, 2
 SUCCESS  SENT MESSAGE TO USER 2, Timestamp 1635063339, Seq 2, Type text, 2
 SUCCESS  SENT MESSAGE TO USER 3, Timestamp 1635063339, Seq 2, Type text, 2
 INFO  RECEIVED MESSAGE, Timestamp 1635063343, Type image, 1
 SUCCESS  SENT MESSAGE TO USER 1, Timestamp 1635063343, Seq 3, Type image, 1
 SUCCESS  SENT MESSAGE TO USER 2, Timestamp 1635063343, Seq 3, Type image, 1
 SUCCESS  SENT MESSAGE TO USER 3, Timestamp 1635063343, Seq 3, Type image, 1
 INFO  RECEIVED MESSAGE, Timestamp 1635063346, Type image, 2
 SUCCESS  SENT MESSAGE TO USER 1, Timestamp 1635063346, Seq 4, Type image, 2
 SUCCESS  SENT MESSAGE TO USER 2, Timestamp 1635063346, Seq 4, Type image, 2
 SUCCESS  SENT MESSAGE TO USER 3, Timestamp 1635063346, Seq 4, Type image, 2
 INFO  RECEIVED MESSAGE, Timestamp 1635063350, Type video, 1
 SUCCESS  SENT MESSAGE TO USER 1, Timestamp 1635063350, Seq 5, Type video, 1
 SUCCESS  SENT MESSAGE TO USER 2, Timestamp 1635063350, Seq 5, Type video, 1
 SUCCESS  SENT MESSAGE TO USER 3, Timestamp 1635063350, Seq 5, Type video, 1
```

Sequencer will receiver message from main thread in the order. Each time of sequencer receive the message, it will sent the message to all user with delay in communication time.

```
### User  1

 SUCCESS  RECEIVED MESSAGE AND DISPLAY, Timestamp 1635063341, Seq 1, Type text, 1
 SUCCESS  RECEIVED MESSAGE AND DISPLAY, Timestamp 1635063342, Seq 2, Type text, 2
 SUCCESS  RECEIVED MESSAGE AND DISPLAY, Timestamp 1635063349, Seq 3, Type image, 1
 SUCCESS  RECEIVED MESSAGE AND DISPLAY, Timestamp 1635063352, Seq 4, Type image, 2
 SUCCESS  RECEIVED MESSAGE AND DISPLAY, Timestamp 1635063364, Seq 5, Type video, 1

#### User  2

 SUCCESS  RECEIVED MESSAGE AND DISPLAY, Timestamp 1635063337, Seq 1, Type text, 1
 SUCCESS  RECEIVED MESSAGE AND DISPLAY, Timestamp 1635063343, Seq 2, Type text, 2
 SUCCESS  RECEIVED MESSAGE AND DISPLAY, Timestamp 1635063349, Seq 3, Type image, 1
 SUCCESS  RECEIVED MESSAGE AND DISPLAY, Timestamp 1635063351, Seq 4, Type image, 2
 SUCCESS  RECEIVED MESSAGE AND DISPLAY, Timestamp 1635063363, Seq 5, Type video, 1

##### User  3

 SUCCESS  RECEIVED MESSAGE AND DISPLAY, Timestamp 1635063340, Seq 1, Type text, 1
 SUCCESS  RECEIVED MESSAGE AND DISPLAY, Timestamp 1635063341, Seq 2, Type text, 2
 SUCCESS  RECEIVED MESSAGE AND DISPLAY, Timestamp 1635063349, Seq 3, Type image, 1
 SUCCESS  RECEIVED MESSAGE AND DISPLAY, Timestamp 1635063354, Seq 4, Type image, 2
 SUCCESS  RECEIVED MESSAGE AND DISPLAY, Timestamp 1635063363, Seq 5, Type video, 1
```

From the result in this scenario, they didn't have any message in buffer. So, it will display all the message in the order that they received.

2. There is one message in buffer

```
# Main

 SUCCESS  SENT MESSAGE, Timestamp 1635063867, Type video, 1
 SUCCESS  SENT MESSAGE, Timestamp 1635063882, Type text, 1
 SUCCESS  SENT MESSAGE, Timestamp 1635063890, Type image, 1
 SUCCESS  SENT MESSAGE, Timestamp 1635063905, Type video, 2
 SUCCESS  SENT MESSAGE, Timestamp 1635063908, Type image, 2
```

Main Thread send the message following by the order to the sequencer.

```
## Sequencer

 INFO  RECEIVED MESSAGE, Timestamp 1635063867, Type video, 1
 SUCCESS  SENT MESSAGE TO USER 1, Timestamp 1635063867, Seq 1, Type video, 1
 SUCCESS  SENT MESSAGE TO USER 2, Timestamp 1635063867, Seq 1, Type video, 1
 SUCCESS  SENT MESSAGE TO USER 3, Timestamp 1635063867, Seq 1, Type video, 1
 INFO  RECEIVED MESSAGE, Timestamp 1635063882, Type text, 1
 SUCCESS  SENT MESSAGE TO USER 1, Timestamp 1635063882, Seq 2, Type text, 1
 SUCCESS  SENT MESSAGE TO USER 2, Timestamp 1635063882, Seq 2, Type text, 1
 SUCCESS  SENT MESSAGE TO USER 3, Timestamp 1635063882, Seq 2, Type text, 1
 INFO  RECEIVED MESSAGE, Timestamp 1635063890, Type image, 1
 SUCCESS  SENT MESSAGE TO USER 1, Timestamp 1635063890, Seq 3, Type image, 1
 SUCCESS  SENT MESSAGE TO USER 2, Timestamp 1635063890, Seq 3, Type image, 1
 SUCCESS  SENT MESSAGE TO USER 3, Timestamp 1635063890, Seq 3, Type image, 1
 INFO  RECEIVED MESSAGE, Timestamp 1635063905, Type video, 2
 SUCCESS  SENT MESSAGE TO USER 1, Timestamp 1635063905, Seq 4, Type video, 2
 SUCCESS  SENT MESSAGE TO USER 2, Timestamp 1635063905, Seq 4, Type video, 2
 SUCCESS  SENT MESSAGE TO USER 3, Timestamp 1635063905, Seq 4, Type video, 2
 INFO  RECEIVED MESSAGE, Timestamp 1635063908, Type image, 2
 SUCCESS  SENT MESSAGE TO USER 1, Timestamp 1635063908, Seq 5, Type image, 2
 SUCCESS  SENT MESSAGE TO USER 2, Timestamp 1635063908, Seq 5, Type image, 2
 SUCCESS  SENT MESSAGE TO USER 3, Timestamp 1635063908, Seq 5, Type image, 2
```

Sequencer will receiver message from main thread in the order. Each time of sequencer receive the message, it will sent the message to all user with delay in communication time.

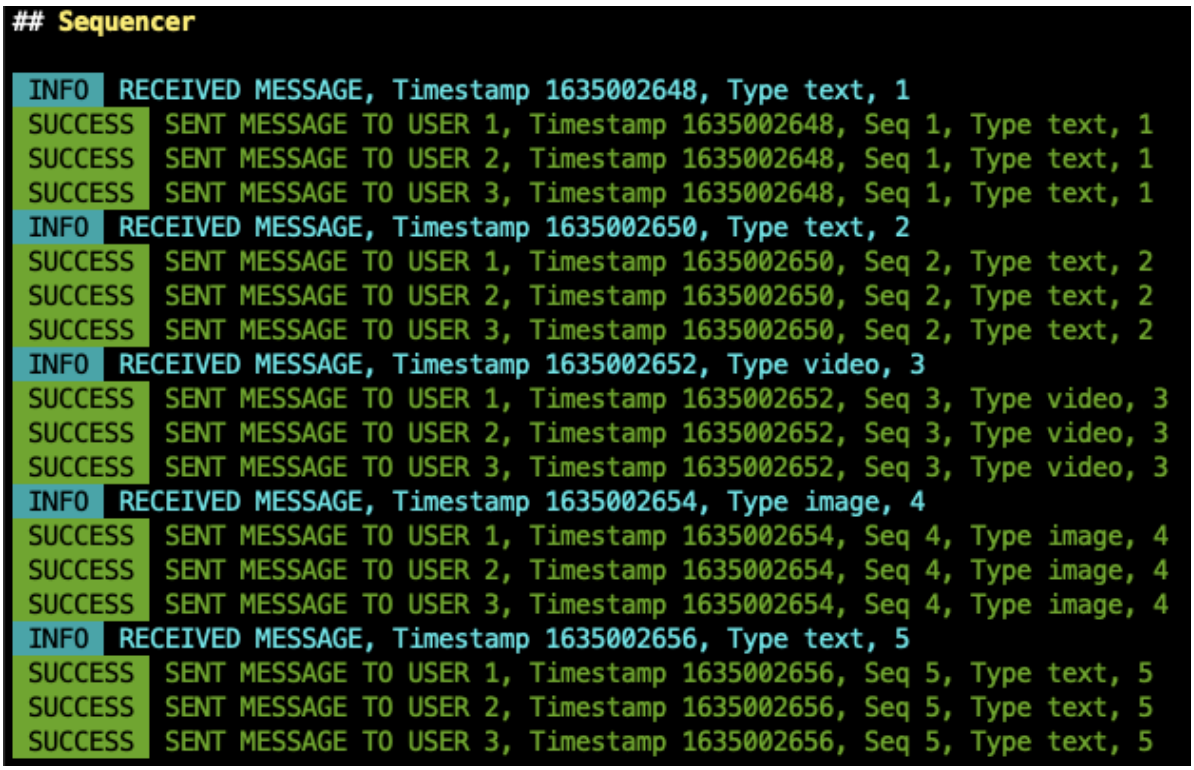
```
### User  1

 SUCCESS  RECEIVED MESSAGE AND DISPLAY, Timestamp 1635055036, Seq 1, Type video, 1
 SUCCESS  RECEIVED MESSAGE AND DISPLAY, Timestamp 1635055043, Seq 2, Type text, 1
 SUCCESS  RECEIVED MESSAGE AND DISPLAY, Timestamp 1635055051, Seq 3, Type image, 1
    INFO  RECEIVED MESSAGE AND ADDED TO BUFFER, Timestamp 1635055062, Seq 5, Type image, 2
 SUCCESS  RECEIVED MESSAGE AND DISPLAY, Timestamp 1635055065, Seq 4, Type video, 2
 SUCCESS  DISPLAY FROM BUFFER, Timestamp 1635055065, Seq 5, Type image, 2

#### User  2

 SUCCESS  RECEIVED MESSAGE AND DISPLAY, Timestamp 1635055035, Seq 1, Type video, 1
 SUCCESS  RECEIVED MESSAGE AND DISPLAY, Timestamp 1635055041, Seq 2, Type text, 1
 SUCCESS  RECEIVED MESSAGE AND DISPLAY, Timestamp 1635055052, Seq 3, Type image, 1
    INFO  RECEIVED MESSAGE AND ADDED TO BUFFER, Timestamp 1635055062, Seq 5, Type image, 2
 SUCCESS  RECEIVED MESSAGE AND DISPLAY, Timestamp 1635055067, Seq 4, Type video, 2
 SUCCESS  DISPLAY FROM BUFFER, Timestamp 1635055067, Seq 5, Type image, 2

##### User  3

 SUCCESS  RECEIVED MESSAGE AND DISPLAY, Timestamp 1635055037, Seq 1, Type video, 1
 SUCCESS  RECEIVED MESSAGE AND DISPLAY, Timestamp 1635055043, Seq 2, Type text, 1
 SUCCESS  RECEIVED MESSAGE AND DISPLAY, Timestamp 1635055050, Seq 3, Type image, 1
    INFO  RECEIVED MESSAGE AND ADDED TO BUFFER, Timestamp 1635055062, Seq 5, Type image, 2
 SUCCESS  RECEIVED MESSAGE AND DISPLAY, Timestamp 1635055065, Seq 4, Type video, 2
 SUCCESS  DISPLAY FROM BUFFER, Timestamp 1635055065, Seq 5, Type image, 2
```

From the result in this scenarios, User1 receive the Video1 and Text 1 in order with no buffer. This situation happened by the text message (15 second) had duration more than Video (12 second) to sent the message in order. Then User1 will receive image1 and image2 but image2 was wrong sequence number, so User1 will waiting to get Video2 and add the image 2 to buffer. Then, User1 will receive Video2 and go into the buffer to display the image 2.

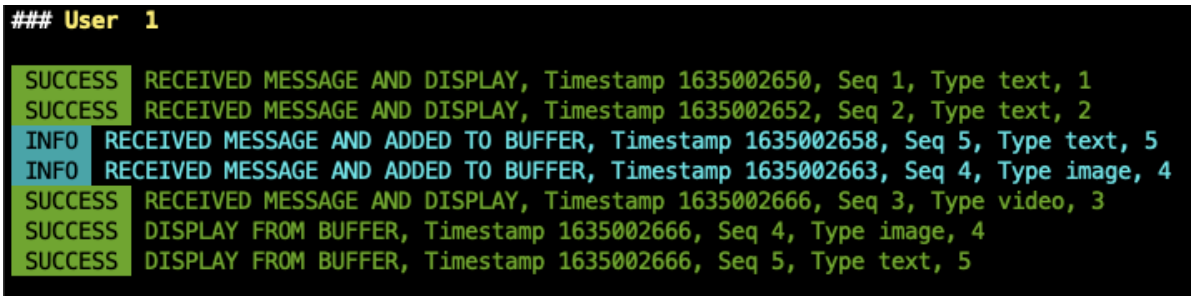
User2 have same order the message with User1.

User3 have same order the message with User1.

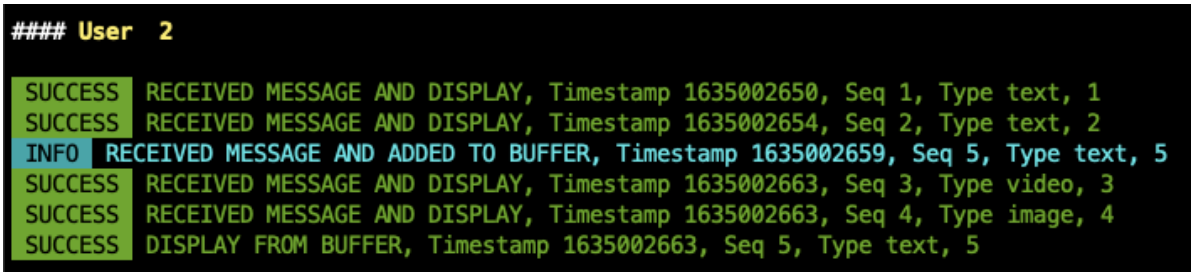
There are many messages in buffer



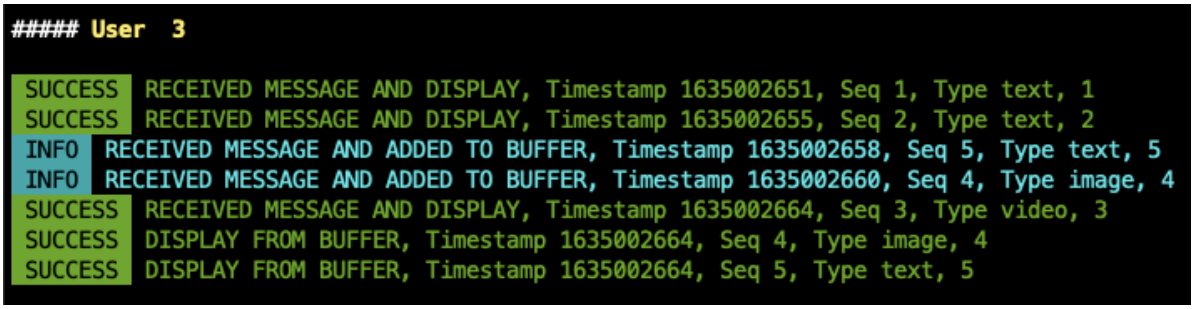
Sequencer receives messages from main thread in the order that message was sent. Each time sequence receive the message, it will send message to all users (one message might reach users in the different time due to the random communication time).



User1 receive and display message 1, and 2 respectively. After that it receive message 4, 5 but the sequence number that user1 is waiting for is 3. So message 4, 5 must be stored in the buffer. Then user1 receive message3 that they are waiting for, so it can display this message. After that they go into buffer and display message 4, and 5 respectively.



User2 receive an display message 1, and 2 respectively. After that it receives message 5 which sequence number is 5 but the sequence of user2 is 2. So message5 will be put int buffer. Then user2 receive message3 (sequence = 3), and sequence of user2 is 2. So message3 can be displayed. After that user2 iterate all messages in buffer which contains message5 (seq = 5) but it cannot be displayed because seq of user2 is 3 and 3+1 != 5. After that user2 receive and display message4 because 3+1 = 4. Finally, user2 iterate all messages in buffer and can display message5 because 4+1 = 5.



The order of messages that user3 receive is same as user1.

User manuals

All you need to do is just install Go by accessing link below

https://golang.org/doc/install

After complete the installation, change directory to the project directory and use go run command to run the simulator

```
go run main.go
```

Team Members

- 62130500209 Thanakorn Aungunchuchod
- 62130500212 Thanaphon Sombunkaeo
- 62130500230 Sethanant Pipatpakorn