
# FORWARDING DISTRIBUTED LINE MESSAGES IN THE CORRECT ORDER

62130500206 KAEWKET SAELEE

62130500226 WISARUT KITTICHAROENPHONNGAM

62130500254 KAVISARA SRISUWATCHAREE

THIS REPORT IS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR

CSC371 INTRODUCTION TO DISTRIBUTED SYSTEMS
AND PARALLEL COMPUTING

SCHOOL OF INFORMATION TECHNOLOGY

KING MONGKUT'S UNIVERSITY OF TECHNOLOGY THONBURI

SEMESTER 1/2021

## Requirement

D-Line wants to maintain the correct order of the messages from the sender regardless of when the messages actually arrive at the destination.

## Theoretical discussion of the solution

From the requirement, we originate with the idea that we can use causal ordering to order the message in each receiver. But the problem is if we have P1, P2, and P3 while P1 and P2 communicate together and P2 will forward messages to P3 that means P1 and P3 shouldn't able to communicate. So we decide to create two groups of communication. The first is P1 and P2 and the second is P2 and P3. Each group will maintain the order of the message using causal ordering.

## System Architecture

Since we decide to have 3 people P1, P2, and P3 where P1 can send and receive messages from P2, P2 can send messages to P1 and P3 but can receive messages from P1, and P3 can receive messages from P2, the system architecture of communication between each people is shown in Figure 1.

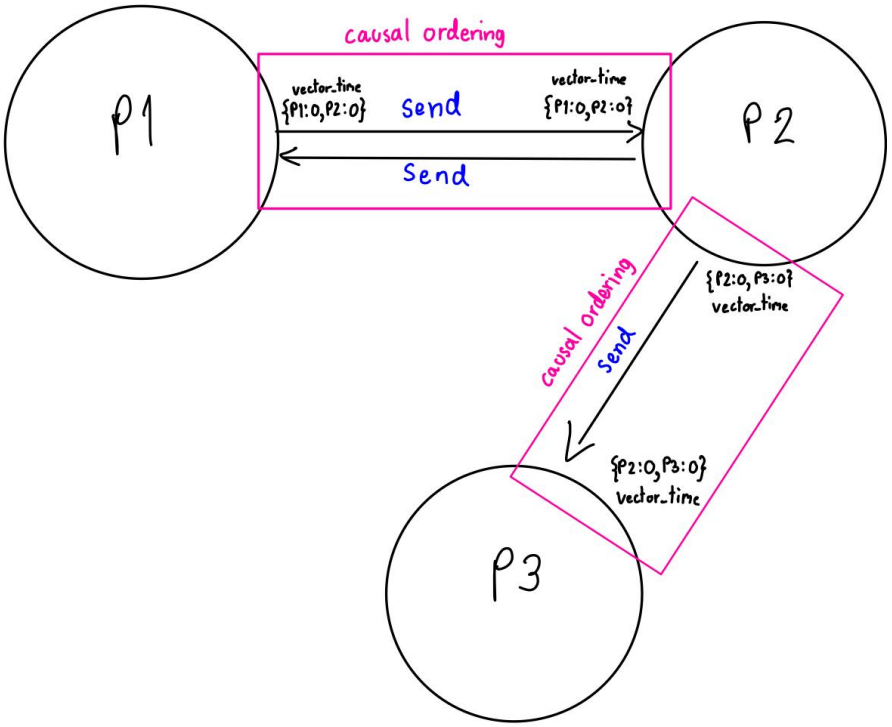



Figure 1 System Architecture

## Implementation

We start with searching for a pre-defined library about causal ordering. Unfortunately, we didn't find the meet our needs so we decide to develop the solution by own.

We decide to develop our simulation using Python with a built-in library such as datetime, multiprocessing, os, socket, time, and threading.

- **datetime** library provides us a function that can manipulate date and time, for this assignment we use datetime.now() to get the current time and datetime.strftime() to format time in the format that we want.
- **multiprocessing** library helps us to manipulate about subprocess in concurrent and allow us to send the data between each process, for this assignment it is the main library that we use to send the message from one process to another process. We use Process to simulate a user, Pipe as the communication way between people, and Array for store shared state.
- **os** library allows us to access the id of the process in the system, we use getpid() to get the process id of the current process.
- **socket** library allows us to transfer data across the network to display in another terminal that we use Netcat to open port for communication.
- **time** library provides us with time-related functions, for this assignment we use time.sleep() to delay between each message sent.
- **threading** library provides us function about managing thread, for this assignment we use threading. Timer to send the message after a specific time for simulating sending different kinds of messages.

**The definition of our simulation**

1) Delays are 0.5 seconds for sending messages and 0 seconds for forwarding a message.
2) Messages are objects that send between the communication of users. There are 3 types of messages.
    a) Text Message uses 1 second for delivery.
    b) Image Message uses 3 seconds for delivery.
    c) Video Message uses 5 seconds for delivery.

**Source Code**

Since we want to use causal ordering which two groups of communication so each people will have its buffer_list and its vector_time to each group. But in the second group, there is only one sender which is P2 so it doesn't need to have buffer_list for that group because it always sends the messages.

P1 will have vector_time and a buffer list of communication with P2.

P2 will have vector_time and a buffer list of communication with P1, and vector_time of communication with P3.

P3 will have vector_time and a buffer list of communication with P2.

We start implementation by using objects for representing the people as you can see in **solution.py.** In this file, Process class has buffer_list as List to store the message that buffered and vector_time as Dict that we decide to make it 2D Dict first dimension is the target group and the second dimension is vector_time in each group.

There are four significant methods of Process class are send_message, send_message_by_type, receive_message, and check_buffer_list.

1) **send_message** is the method that we create for setting the delay of each type of message and we also check that if the sender is P2 it also send to P3.

2) **send_message_by_type** is the method that we implement same as the causal ordering concept that we add the vector time of the sender by 1 and send the message to other people.

3) **receive_message** is the method that we implement same as the causal ordering concept that when we receive the message we need to check two conditions. First is vector time of sender from incoming message must equal our vector time of sender that we plus 1. Second is other vector time of receiver must more than or equal the vector time of sender. Otherwise, the message will be buffered and stored in a buffer list. We also check that if the receiver is P2 it also send to P3 and we also check messages in the buffer list.

4) **check_buffer_list** is the method that we use for checking that we can receive the message in the buffer list or not that we will call when receiving message success.

The code is run in sequence in a single process, works as our expectation, and sent the output to show in another 3 terminals using socket and netcat.

Then we change from use object to use process for representing the people. We bring multiprocessing library in our project for implementation. The code is in **solution_multithread.py**. The code is similar to before, it just changes the method name and refactors to be implemented in multi-process.

As I mentioned before, in multiprocessing we use Process to simulate a user, Pipe as the communication way between people, and Array for store shared state.

There are four significant methods are send_message, receive_message, handle_receive_message, and handle_buffer_message.

1) **send_message** is the method that we use for sending the messages which have different delivery times depending on the type of message to Pipe which is a communication way between two processes (two users).

2) **receive_message** is the method that we use for receiving the messages from Pipe and passing it to handle_receive_message.

3) **handle_receive_message** is the method that we use for handling the message that we can receive or we need to put it into the buffer list. We check two conditions of causal ordering, first is the vector time of sender from incoming message must equal our vector time of sender that we plus 1. Second is other vector time of receiver must more than or equal the vector time of sender. Otherwise, the message will be buffered and stored in a buffer list. After that, we check whether the process is process2 or not because we need to forward the message to process3. Then we call handle_buffer_message method.

4) **handle_buffer_message** is the method that we use for checking that we can receive the message in the buffer list or not by passing the message in the buffer list into handle_receive_message.

We also sent the output to show in another 3 terminals using socket and netcat. But the flow of our solution is changed since when wait to receive a message from another process the process cannot send the message or do anything else. But we think it is acceptable since in the real world when we communicate with people if we talk about the same topic we need to know the message of other people before we reply.

## Test Cases of Simulation

Our test cases provide for sending messages like chat that have only P1 and P2 can send messages. There are 5 test cases for sending the message to test simulation in different contexts as shown in Table 1.

Table 1: D-Line Simulation Testcase

| Test Case Number | Name | Description | Expected Output | Message List |
|---|---|---|---|---|
| 1 | One sender without buffer | In this test case, we want to simulate the message passing when we have only one sender. The receiver can receive the messages sequentially without buffering the early arrival messages, and it can forward those messages to another receiver with the same order of messages as received. | All people have the message in the same order that the sender send | There are four messages sent from P1 to P2.<br>1) Text Message with the content is “Hello Gobgab”<br>2) Text Message with the content is “Good Morning”<br>3) Text Message with the content is “How are you?”<br>4) Text Message with the content is “emoji” |
| 2 | One sender with only one message in the buffer list | In this test case, we want to simulate the message passing when we have only one sender but have only one message in the buffer. The sender can send the messages sequentially. On the other hand, the receiver expects that the early arrival message will be buffered and received from the buffer list when satisfying the message order by other messages in the buffer list. | All people have the message in the same order that the sender send | There are three messages sent from P1 to P2.<br>1) Text Message with the content is “Hello Opal”<br>2) Video Message with the content is "Meme video"<br>3) Text Message with the content is “emoji” |

| | | | | |
|---|---|---|---|---|
| 3 | One sender with multiple messages in the buffer list | In this test case, we want to simulate the message passing when we have only one sender with multiple messages in the buffer. The sender can send the messages sequentially. On the other hand, the receiver expected that the early arrival message will be buffered and received from the buffer list when satisfying the message order by other messages in the buffer list. | All people have the message in the same order that the sender send | There are five messages sent from P1 to P2.<br>1) Text Message with the content is “Hello Gobgab”<br>2) Image Message with the content is “Good morning image”<br>3) Video Message with the content is “video opal”<br>4) Image Message with the content is “image meme gobgab”<br>5) Text Message with the content is “Isn’t it cute” |
| 4 | Two senders with only one message in the buffer list | In this test case, we want to simulate the message passing when we have two senders but have only one message in the buffer. The senders send messages to the receiver. In contrast, another receiver will be the sender to send a message to the receiver. The receiver expects that the early arrival message will be buffered and received from the buffer list when satisfying the message order by other messages in the buffer list. | All people have the message in the same order that the sender send | There are five messages sent from P1 and P2.<br>1) Text Message with the content from P2 is “Hello krub”<br>2) Video Message with the content from P2 is “video chingly”<br>3) Image Message with the content from P2 is “meme image”<br>4) Text Message with the content from P1 is “smile”<br>5) Image Message with content from P1 is “smile image” |

| | | | | |
|---|---|---|---|---|
| 5 | Two senders with multiple messages in the buffer list | In this test case, we want to simulate the message passing when we have two senders with multiple messages in the buffer. The senders send messages to the receiver. In contrast, another receiver will be the sender to send a message to the receiver. The receiver expects that the early arrival message will be buffered and received from the buffer list when satisfying the message order by other messages in the buffer list. | All people have the message in the same order that the sender send | There are seven messages sent from P1 and P2.<br>1) Text Message with the content from P1 is “Hello gobgab”<br>2) Image Message with the content from P1 is “meme image1”<br>3) Text Message with the content from P2 is “Hello Chingly”<br>4) Image Message with the content from P1 is “meme image1”<br>5) Video Message with the content from P2 is “video gobgab”<br>6) Image Message with content from P2 is “image meme gobgab”<br>7) Text Message with the content from P2 is “emoji” |

## Simulation Results

### Test case #1: One sender without buffer

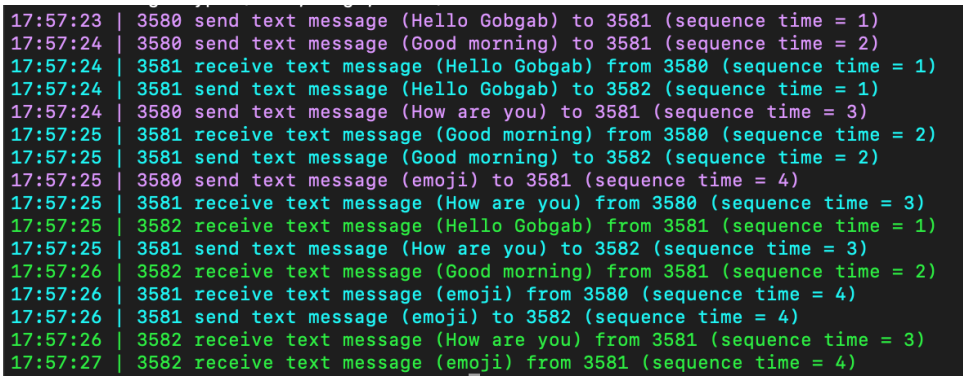



Figure 2: Simulation of Test case #1

From Figure 2, there are 4 messages sent from P1 which are 1. Text Message “Hello Gobgab” 2. Text Message “Good morning” 3. Text Message “How are you” 4. Text Message “emoji”

From our implementation, P1 (3580) will send messages to P2 (3581) which each message has a send delay of 0.5 seconds. Once P2 receives the message successfully, it will forward that message to P3 (3582).

The event that occurs in the system is shown in Table 2. (The timestamp that no event occurs will not be mentioned).

Table 2: Event in the simulation of Test case #1

| **Timestamp from start (second)** | **P1** | **P2** | **P3** |
|---|---|---|---|
| 0 | Send “Hello Gobgab” to P2 | | |
| 0.5 | Send “Good morning” to P2 | | |
| 1 | Send “How are you” to P2 | Receive “Hello Gobgab” and Forward it to P3 | |
| 1.5 | Send “emoji” to P2 | Receive “Good morning” and Forward it to P3 | |
| 2 | | Receive “How are you” and Forward it to P3 | Receive “Hello Gobgab” from P2 |

| Timestamp from start (second) | P1 | P2 | P3 |
|---|---|---|---|
| 2.5 | | Receive “emoji” and Forward it to P3 | Receive “Good morning” from P2 |
| 3 | | | Receive “How are you” from P2 |
| 3.5 | | | Receive “emoji” from P2 |

**Test case #2: One sender with only one message in the buffer list**

```
18:19:14 | 3658 send text message (Hello Opal) to 3659 (sequence time = 1)
18:19:14 | 3658 send video message (Meme video) to 3659 (sequence time = 2)
18:19:15 | 3658 send text message (emoji) to 3659 (sequence time = 3)
18:19:15 | 3659 receive text message (Hello Opal) from 3658 (sequence time = 1)
18:19:15 | 3659 send text message (Hello Opal) to 3660 (sequence time = 1)
18:19:16 | 3660 receive text message (Hello Opal) from 3659 (sequence time = 1)
18:19:19 | 3659 receive video message (Meme video) from 3658 (sequence time = 2)
18:19:19 | 3659 send video message (Meme video) to 3660 (sequence time = 2)
18:19:19 | 3659 receive text message (emoji) from 3658 (sequence time = 3)
18:19:19 | 3659 send text message (emoji) to 3660 (sequence time = 3)
18:19:24 | 3660 receive video message (Meme video) from 3659 (sequence time = 2)
18:19:24 | 3660 receive text message (emoji) from 3659 (sequence time = 3)
```

Figure 3: Simulation of Test case #2

From Figure 3, there are 3 messages sent from P1 which are 1. Text Message “Hello Opal” 2. Video Message “Meme video” 3. Text Message “emoji”

From our implementation, P1 (3658) will send messages to P2 (3659) which each message has a send delay of 0.5 seconds. Once P2 receives the message successfully, it will forward that message to P3 (3660).

The event that occurs in the system is shown in Table 3. (The timestamp that no event occurs will not be mentioned).

Table 3: Event in the simulation of Test case #2

| **Timestamp from start (second)** | **P1** | **P2** | **P3** |
|---|---|---|---|
| 0 | Send “Hello Opal” to P2 | | |
| 0.5 | Send “Meme video” to P2 | | |
| 1 | Send “emoji” to P2 | Receive “Hello Opal” and Forward it to P3 | |
| 2 | | Receive “emoji” but put it in the buffer list. | Receive “Hello Opal” from P2 |
| 3 | | Receive “emoji” and Forward it to P3 | |

| Timestamp from start (second) | P1 | P2 | P3 |
|---|---|---|---|
| 5.5 | | Receive “Meme video” and Forward it to P3”<br>Receive “emoji” from the buffer list and Forward it to P3 | |
| 6.5 | | | Receive “emoji” from P2 but put it in the buffer list. |
| 10.5 | | | Receive “Meme video” from P2<br>Receive “emoji” from the buffer list |

**Test case #3: One sender with multiple messages in the buffer list**

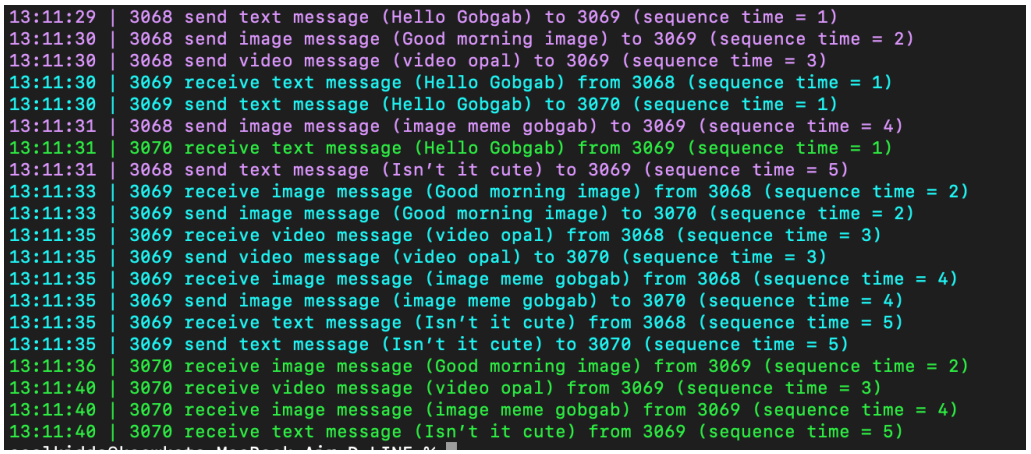



Figure 4: Simulation of Test case #3

From Figure 4, there are 6 messages sent from P1 which are 1. Text Message "Hello Gobgab" 2. Image Message "Good morning image" 3. Video Message "Video opal" 4. Image Message "Image meme gobgab" 5. Text Message "Isn't it cute?"

From our implementation, P1(3068) will send messages to P2 (3069) which each message has a send delay of 0.5 seconds. Once P2 receives the message successfully, it will forward that message to P3 (3070).

The event that occurs in the system is shown in Table 4. (The timestamp that no event occurs will not be mentioned).

Table 4: Event in the simulation of Test case #3

| **Timestamp from start (second)** | **P1** | **P2** | **P3** |
|---|---|---|---|
| 0 | Send "Hello Gobgab" to P2 | | |
| 0.5 | Send "Good morning Image" to P2 | | |
| 1 | Send "Video opal" to P2 | Receive "Hello Gobgab" and Forward it to P3 | |
| 1.5 | Send "Image meme gobgab" to P2 | | |

| **Timestamp from start (second)** | **P1** | **P2** | **P3** |
|---|---|---|---|
| 2 | Send “Isn’t it cute?” to P2 | | Receive “Hello Gobgab” from P2 |
| 3 | | Receive “Isn’t it cute?” but put it in the buffer list. | |
| 3.5 | | Receive “Good morning Image” and Forward it to P3 | |
| 4.5 | | Receive “Image meme gobgab” but put it in the buffer list. | |
| 6 | | Receive “Video opal” and Forward it to P3”<br>Receive “Image meme gobgab” from the buffer list and Forward it to P3<br>Receive “Isn’t it cute?” from the buffer list and Forward it to P3 | |
| 6.5 | | | Receive “Good morning Image” from P2 |
| 7 | | | Receive “Isn’t it cute?” from P2 but put it in the buffer list. |

| Timestamp from start (second) | P1 | P2 | P3 |
|---|---|---|---|
| 9 | | | Receive “Image meme gobgab” from P2 but put it in the buffer list. |
| 11 | | | Receive “Video opal” from P2<br>Receive “Image meme gobgab” from the buffer list<br>Receive “Isn’t it cute?” from the buffer list |

**Test case #4: Two senders with only one message in the buffer list**

```
20:34:40 | 69624 send text message (Hello krub) to 69623 (sequence time = 1)
20:34:40 | 69624 send text message (Hello krub) to 69625 (sequence time = 1)
20:34:40 | 69624 send video message (video chingly) to 69623 (sequence time = 2)
20:34:40 | 69624 send video message (video chingly) to 69625 (sequence time = 2)
20:34:41 | 69623 receive text message (Hello krub) from 69624 (sequence time = 1)
20:34:41 | 69625 receive text message (Hello krub) from 69624 (sequence time = 1)
20:34:41 | 69624 send image message (meme image 1) to 69623 (sequence time = 3)
20:34:41 | 69624 send image message (meme image 1) to 69625 (sequence time = 3)
20:34:45 | 69625 receive video message (video chingly) from 69624 (sequence time = 2)
20:34:45 | 69625 receive image message (meme image 1) from 69624 (sequence time = 3)
20:34:45 | 69623 receive video message (video chingly) from 69624 (sequence time = 2)
20:34:45 | 69623 receive image message (meme image 1) from 69624 (sequence time = 3)
20:34:45 | 69623 send text message (smile) to 69624 (sequence time = 1)
20:34:46 | 69623 send image message (smile img) to 69624 (sequence time = 2)
20:34:46 | 69624 receive text message (smile) from 69623 (sequence time = 1)
20:34:46 | 69624 send text message (smile) to 69625 (sequence time = 4)
20:34:47 | 69625 receive text message (smile) from 69624 (sequence time = 4)
20:34:49 | 69624 receive image message (smile img) from 69623 (sequence time = 2)
20:34:49 | 69624 send image message (smile img) to 69625 (sequence time = 5)
20:34:52 | 69625 receive image message (smile img) from 69624 (sequence time = 5)
```

Figure 5: Simulation of Test case #4

From Figure 5, there are 5 messages sent from P1 and P2 which are 1. Text Message “Hello grub” 2. Video Message “Video chingly” 3. Image Message “meme image” 4. Text message “smile” 5. Image message “smile image”

From our implementation, P2 (69624) will send messages to P1 (69623) and P3 (69625) at the same time. And in some parts, P1 also sends a message to P2 which each message has a send delay of 0.5 seconds. Once P2 receives the message successfully, it will forward that message to P3 (69625).

The event that occurs in the system is shown in Table 5. (The timestamp that no event occurs will not be mentioned).

Table 5: Event in the simulation of Test case #4

| **Timestamp from start (second)** | **P1** | **P2** | **P3** |
|---|---|---|---|
| 0 | | Send “Hello krub” to P1 and P3 | |
| 0.5 | | Send “Video chingly” to P1 and P3 | |
| 1 | Receive “Hello krub” from P2 | Send “Meme image” to P1 and P3 | Receive “Hello krub” from P2 |

| Timestamp from start (second) | P1 | P2 | P3 |
|---|---|---|---|
| 4 | Receive “Meme image” from P2 but put it in the buffer list | | Receive “Meme image” from P2 but put it in the buffer list |
| 5.5 | Receive “Video chingly” from P2<br>Receive “Meme image” from the buffer list<br>Send “Smile text” to P2 | | Receive “Video chingly” from P2<br>Receive “Meme image” from the buffer list |
| 6 | Send “Smile image” to P2 | | |
| 6.5 | | Receive “Smile text” from P2 and Forward it to P3 | |
| 7.5 | | | Receive “Smile text” from P2 |
| 9 | | Receive “Smile image” from P1 and Forward it to P3 | |
| 12 | | | Receive “Smile image” from P2 |

**Test case #5: 2 senders with multiple messages in buffer**

```
09:06:18 | 1589 send text message (Hello gobgab) to 1590 (sequence time = 1)
09:06:18 | 1589 send image message (Meme image 1) to 1590 (sequence time = 2)
09:06:19 | 1590 receive text message (Hello gobgab) from 1589 (sequence time = 1)
09:06:19 | 1590 send text message (Hello gobgab) to 1591 (sequence time = 1)
09:06:20 | 1591 receive text message (Hello gobgab) from 1590 (sequence time = 1)
09:06:21 | 1590 receive image message (Meme image 1) from 1589 (sequence time = 2)
09:06:21 | 1590 send image message (Meme image 1) to 1591 (sequence time = 2)
09:06:21 | 1590 send text message (Hello chingly) to 1589 (sequence time = 1)
09:06:21 | 1590 send text message (Hello chingly) to 1591 (sequence time = 3)
09:06:22 | 1590 send image message (Meme image 2) to 1589 (sequence time = 2)
09:06:22 | 1590 send image message (Meme image 2) to 1591 (sequence time = 4)
09:06:22 | 1589 receive text message (Hello chingly) from 1590 (sequence time = 1)
09:06:22 | 1590 send video message (Video gobgab) to 1589 (sequence time = 3)
09:06:22 | 1590 send video message (Video gobgab) to 1591 (sequence time = 5)
09:06:23 | 1590 send image message (Image meme gobgab) to 1589 (sequence time = 4)
09:06:23 | 1590 send image message (Image meme gobgab) to 1591 (sequence time = 6)
09:06:23 | 1590 send text message (emoji) to 1589 (sequence time = 5)
09:06:23 | 1590 send text message (emoji) to 1591 (sequence time = 7)
09:06:24 | 1591 receive image message (Meme image 1) from 1590 (sequence time = 2)
09:06:24 | 1591 receive text message (Hello chingly) from 1590 (sequence time = 3)
09:06:25 | 1589 receive image message (Meme image 2) from 1590 (sequence time = 2)
09:06:25 | 1591 receive image message (Meme image 2) from 1590 (sequence time = 4)
09:06:27 | 1591 receive video message (Video gobgab) from 1590 (sequence time = 5)
09:06:27 | 1589 receive video message (Video gobgab) from 1590 (sequence time = 3)
09:06:27 | 1591 receive image message (Image meme gobgab) from 1590 (sequence time = 6
09:06:27 | 1589 receive image message (Image meme gobgab) from 1590 (sequence time = 4
09:06:27 | 1591 receive text message (emoji) from 1590 (sequence time = 7)
09:06:27 | 1589 receive text message (emoji) from 1590 (sequence time = 5)
```

Figure 6: Simulation of Test case #5

From Figure 6, There are 7 messages sent from P1 and P2 which are 1. Text Message “Hello gobgab” 2. Image Message “Meme image 1” 3. Text Message “Hello Chingly” 4. Image message “Meme image 2” 5. Video message “Video gobgab” 6. Image message “Image meme gobgab” 7. Text message “emoji”

From our implementation, P1 (1589) will send messages to P2 (1590) which each message has a send delay of 0.5 seconds. Once P2 receives the message successfully, it will forward that message to P3 (1591). And in some parts, P2 also sends messages to P1 and P3 at the same time.

The event that occurs in the system is shown in Table 6. (The timestamp that no event occurs will not be mentioned).

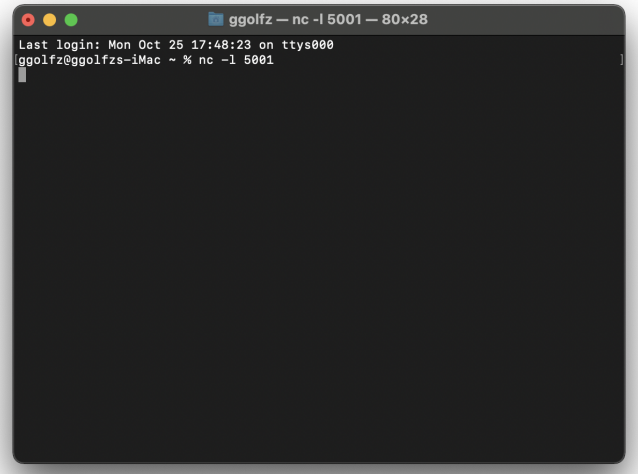
Table 6: Event in the simulation of Test case #5

| **Timestamp from start (second)** | **P1** | **P2** | **P3** |
|---|---|---|---|
| 0 | Send “Hello gobgab” to P2 | | |
| 0.5 | Send “Meme image 1 to P2” | | |
| 1 | | Receive “Hello gobgab” and Forward it to P3 | |
| 2 | | | Receive “Hello gobgab” from P2 |
| 3.5 | | Receive “Meme image 1” and Forward it to P3 Send “Hello Chingly” to P1 and P3 | |
| 4 | | Send “Meme image 2” to P1 and P3 | |
| 4.5 | Receive “Hello Chingly” from P2 | Send “video gobgab” to P1 and P3 | Receive “Hello Chingly” from P2 but put it in the buffer list |
| 5 | | Send “Image meme gobgab” to P1 and P3 | |
| 5.5 | | Send “emoji” to P1 and P3 | |

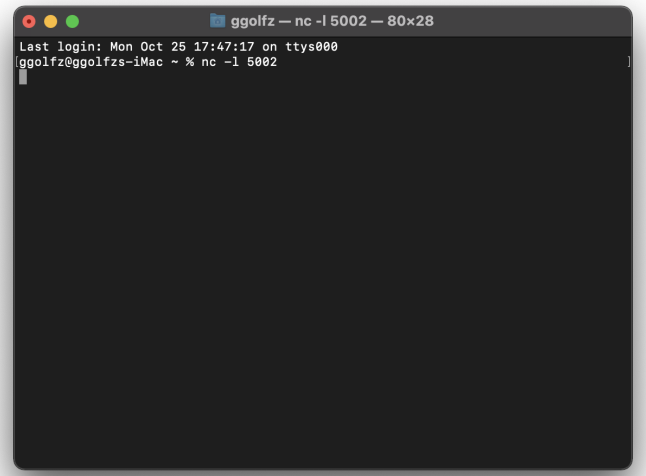
| Timestamp from start (second) | P1 | P2 | P3 |
|---|---|---|---|
| 6.5 | Receive “emoji” From P2 but put it in the buffer list | | Receive “Meme image 1” from P2<br>Receive “Hello Chingly” from the buffer list<br>Receive “emoji” From P2 but put it in the buffer list |
| 7 | Receive “Meme image 2” From P2 | | Receive “Meme image 2 From P2” |
| 8 | Receive “Image meme gobgab” from P2 but put it in the buffer list | | Receive “Image meme gobgab” from P2 but put it in the buffer list |
| 9.5 | Receive “video gobgab” from P2<br>Receive “Image meme gobgab” and “emoji” sequentially from the buffer list | | Receive “video gobgab” from P2<br>Receive “Image meme gobgab” and “emoji” sequentially from the buffer list |

# User Manual

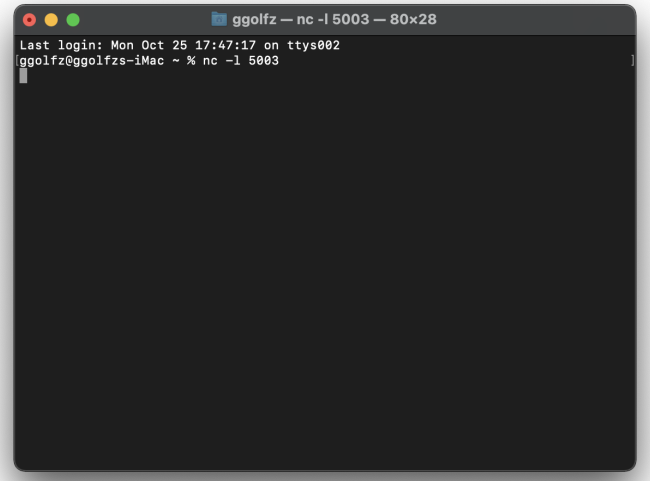
1) Open 4 terminals
2) Run 1st terminal using nc -l 5001



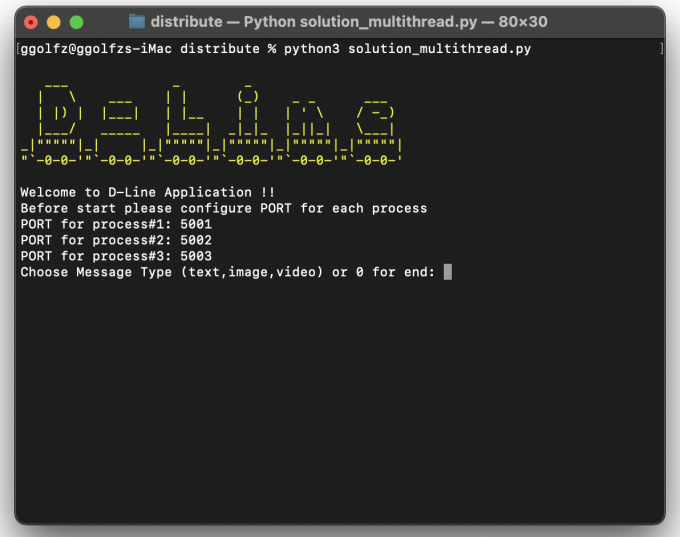

3) Run 2nd terminal using nc -l 5002



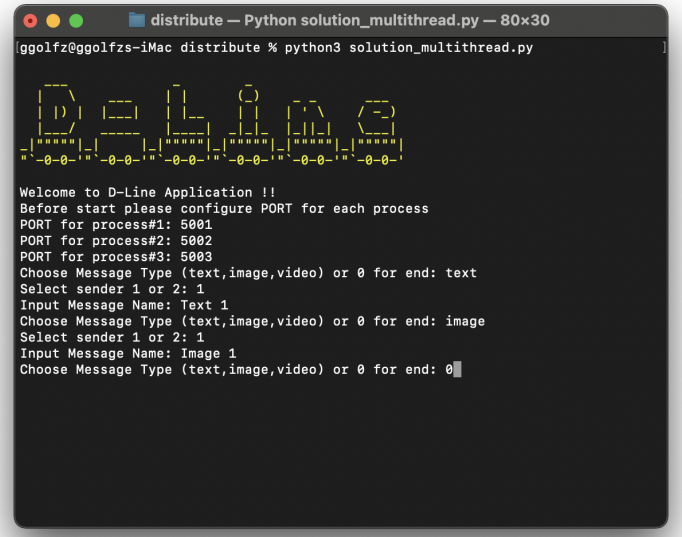
4) Run 3rd terminal using nc -l 5003




5) Run python3 solution_multithread.py in the last terminal or python3 solution_multithread.py < testcase-n.in (n is from 1 to 5) if you want to use a predefined test case (predefined test case will skip step 6-12 for you).
6) Configure port of process#1 to be 5001
7) Configure port of process#2 to be 5002
8) Configure port of process#3 to be 5003



9) Select type of messages (text, image, video)
10) Select sender (1 or 2)
11) Type the message content
12) Repeat steps 9-11 until you want to stop (press 0 for stop)




13) The simulation will show in all terminals.