CSE536 Project 3 – Use a logical clock to impose an ordering on messages

Goal:

Project 3 uses project 2 to communicate between machines to implement a logical clock. When sending a message from user i to user j we want to have an ordering to determine among several messages which came first.

Other details:

- Where to store the clock
  - We decided to store it in the .ko
- How to handle multiple apps on the same host
  - We decided to use synchronization to control access from multiple users. Note the need to copy data from the user to the kernel before allowing a tasklet thread to access the data.
- How to make the communication reliable and not duplicated
  - We decided to limit our reliability to waiting for an ack before retransmitting after 5 seconds. If a second timeout occurs discard the event as unobtainable. Keep no state on the receiver ignoring possible duplicates.
- Protocol Number 234
- Transaction format – Store IP numbers in standard format (Big Endian in_aton() at kernel level, inet_aton() at the user level)
  - Ack format
    - 4 Record ID – ack=0 or event =1
    - 4 final clock
    - 4 original clock
    - 4 source IP
    - 4 destination IP
    - 236 string
  - Event format
    - 4 Record ID – ack=0 or event=1
    - 4 final clock
    - 4 original clock
    - 4 source IP
    - 4 destination IP
    - 236 string

Step by step working:

Sender side,

1. We create a data structure of 256 bytes as described in the problem statement.
   struct packetformat {
   uint32_t record_id;
   uint32_t final_clock;
   uint32_t original_clock;
   __be32 source_ip;
   __be32 destination_ip;
   uint8_t data[236];
   }
2. Then we set destination address and monitor IP address.
3. Next we take input from user from our menu W option and assign it to 'data' field of packet.
4. Initialize other members such as record ID =1, final and original clock to 0.
5. In the cse5361 kernel module we assign clock value to original clock and send the event to destination.

At the same time Pass the event to cse536app again and send the copy to CSE536monitor using the udpclient code.

6. When acknowledgement is not received in 5 seconds, resend the message.
7. Increment the clock after each event sent.

On receiver side,

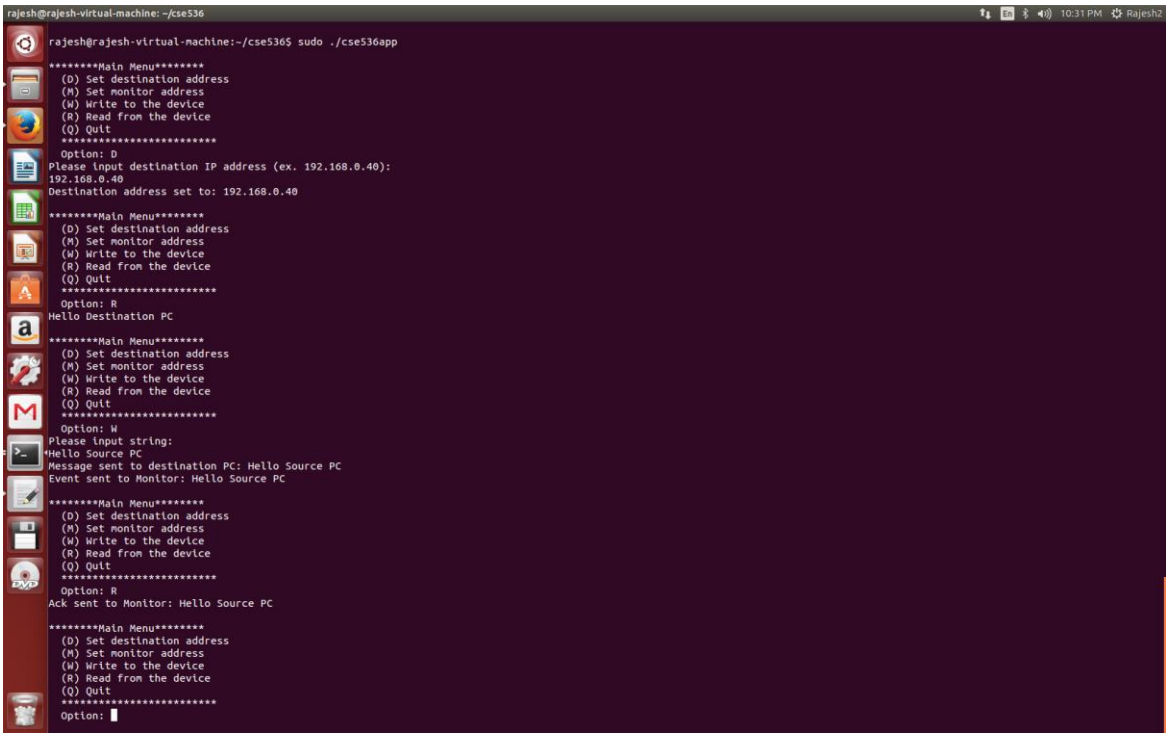
8. Check the record ID, if it is 1 then change it to 0 and assign clock value to final clock value field. Resend the message to sender with rest of the field unchanged.
9. Also, this is the time to update clock. If packet's original clock is greater than local clock then
   Clock = received original clock +1;
   Final clock = clock;

On sender side again,

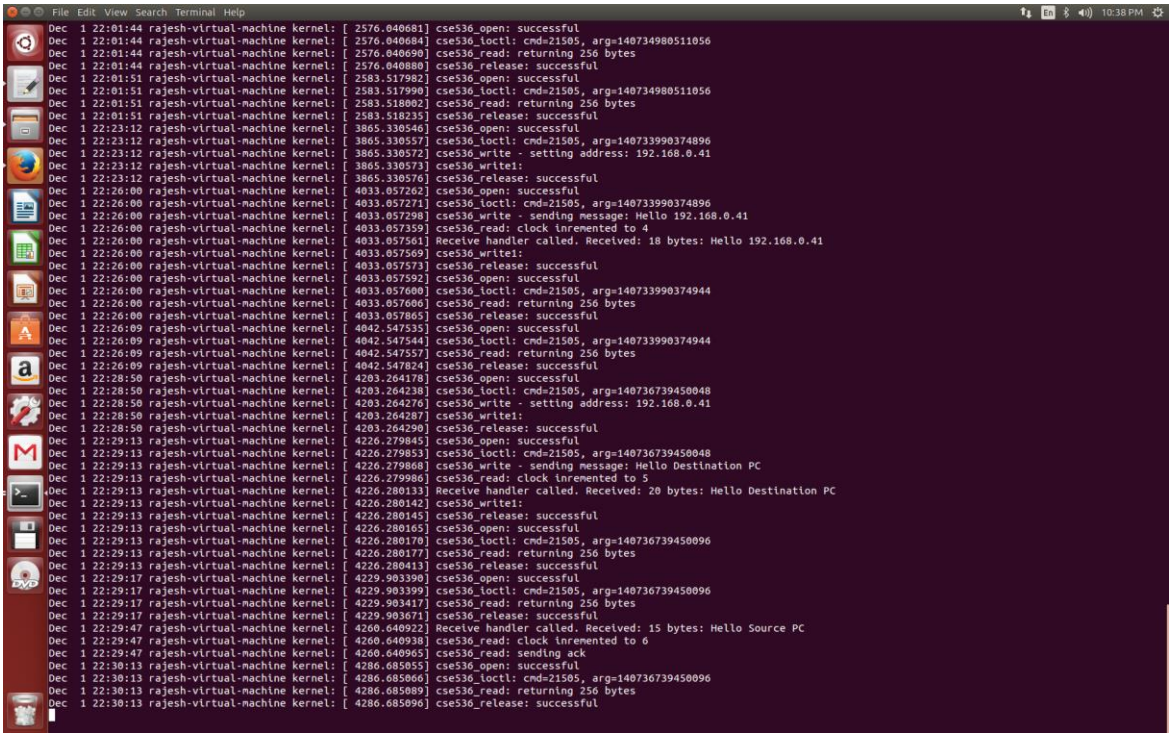
10. When acknowledgement is received (record ID is 0) read it in cse536app.
11. If acknowledgement matches with event sent then send the copy to CSE536monitor using the udpclient code.

Screenshots:

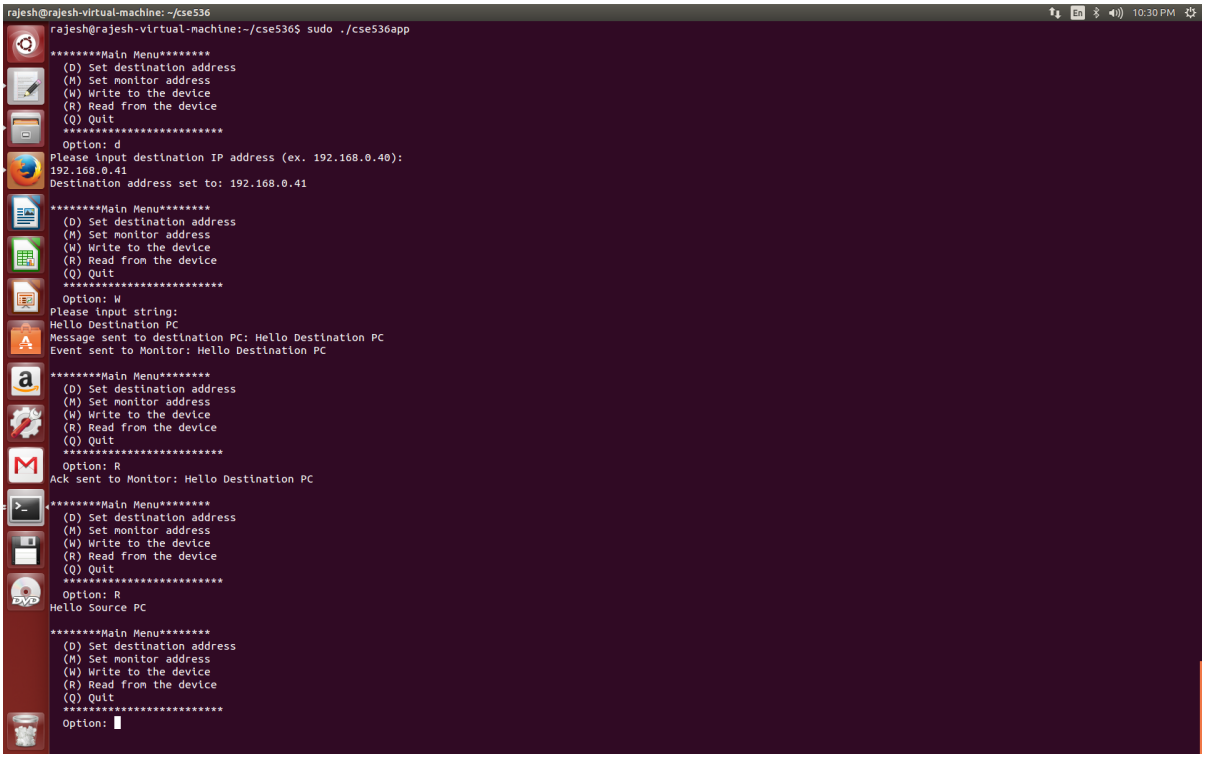
1. Sender (IP address = 192.168.0.40)



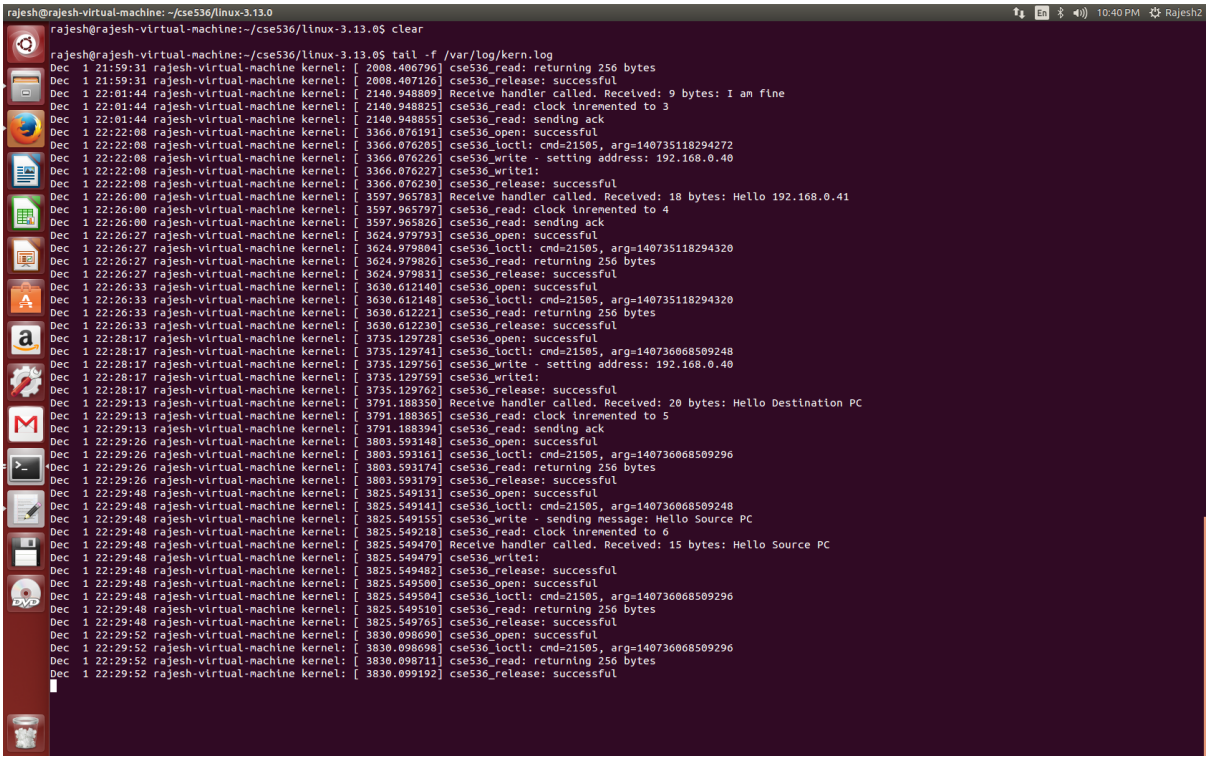
2. Kernel Traces on sender side



3. Receiver (IP address = 192.168.0.41)



4. Kernel Traces on receiver side



5. Monitor after the events and acknowledgements