# CSE536 Advanced Operating Systems

Nagarjuna Myla
Professor Alan Skousen
School of Computing, Informatics, and Decision Systems Engineering
Arizona State University
Tempe, AZ 85287

**Abstract**

Project3 extends the capability of the project2 deliverable by impose an ordering on messages using a logical clock over the distributed systems. In a distributed system, it is not possible in practice to synchronize time across entities (typically thought of as processes) within the system; hence, the entities can use the concept of a logical clock based on the events through which they communicate.

Keywords:
Linux, kernel, module, character device, network, logical, clock, event, ordering, distributed, environment

# 1 Introduction

## 1.1 Terminology

RecordId - ACK=0 Event=1
Final Clock: To store the global time across multiple systems to impose ordering
Orig Clock: Local clock to the single system to store its events
Source: IP address of source machine
Destination: IP address of destination machine
TM: target machine (which is server)
DM: debug machine (which is client)

## 1.2 Goal Description

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

When sending a message from user i to user j over distributed environment we want to have an ordering to determine among several messages which came first.
• Where to store the clock
• How to handle multiple apps on the same host
• How to make the communication reliable and not duplicated
• Transaction format – Store IP numbers in standard format (Big Endian in_aton() at kernel level, inet_aton() at the user level)

## 1.3 Assumptions

cse5361 Character device buffer can hold a max of 256 characters.

# 2 Proposed Solution

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

We decided to store clock in the .ko. 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. To make the communication reliable and not duplicated o 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. We will use the following format to communicate over the network for Acknowledgements and Events.

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

In order to use the monitor all transmissions will need to be made in a uniform manner. Using the record layout specified above app will submit a buffer to the kernel module using the write function. The app will initialize the buffer to zeros and then filled in the record ID with a 1 (for an event), the final and original clock with 0, the source IP with the local IP, and the destination IP with the destination address. The string can be any printable string terminated with a 0. When the write function returns, the original clock field in the buffer is updated by kernel module. The kernel module also have sends the event to its destination. Upon return the app will send a copy of the event, as returned by the module in the buffer, to the CSE536monitor using the udpclient code.
Upon receipt of an event by kernel module the clock is updated and the value stored in the final clock field and the event stored for the local app to read. The kernel module then change the record ID to 0 in a copy of the event and send the copy as an acknowledgement back to the source where it will be stored until the app reads it. When the app reads the acknowledgement it sends a copy to the CSE536monitor using the udpclient code.
Apps will be reading events from the module and acknowledgements for transmitted events as well. Only acknowledgements are sent to the CSE536monitor from the read function.

## 3 Execution

### 3.1 Project3

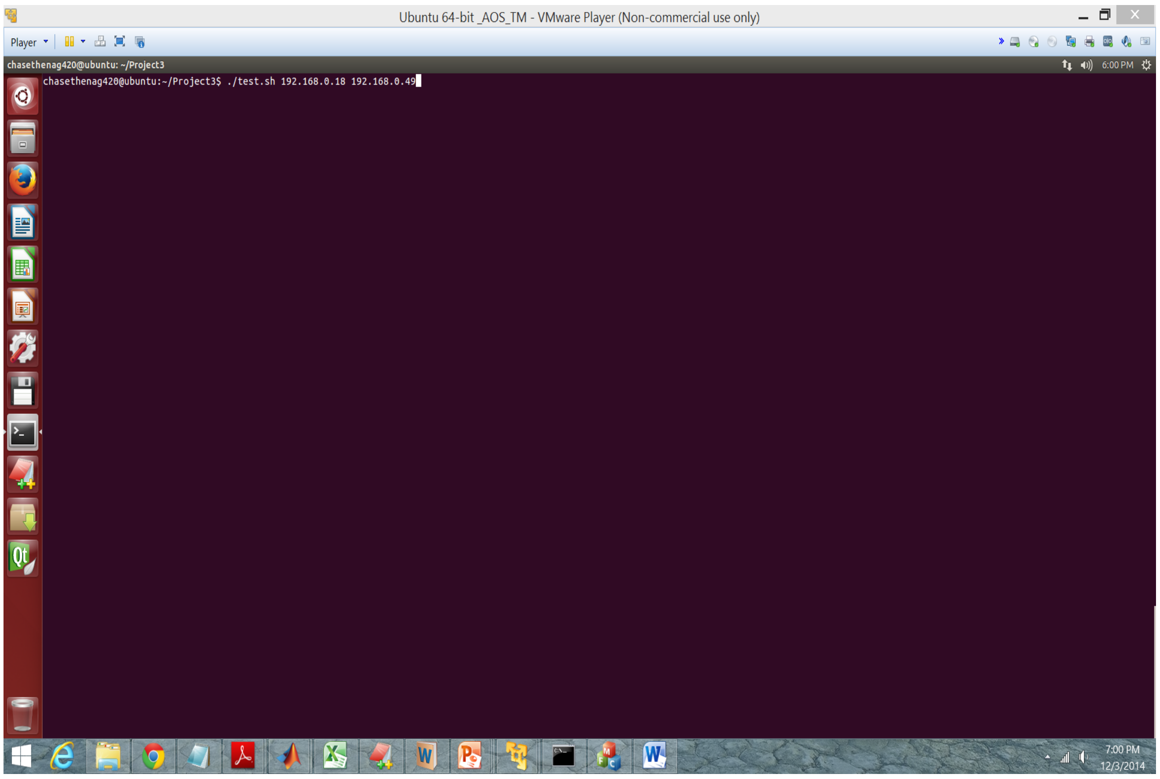
To run project3 you need to complete all the steps in project2 and project1. Once you are ready execution of Project is as simple as running a shell script file. Extract the Project3.zip package and run test.sh by passing two parameters: 1. cse536monitor ip address and 2. remote machine IP address.

Usage: ./test.sh 'cse536monitor ip' 'remote machine ip'
Example: ./test.sh 192.168.0.14 192.168.0.15

Do the same process on the remote machine as well to connect to other machine.

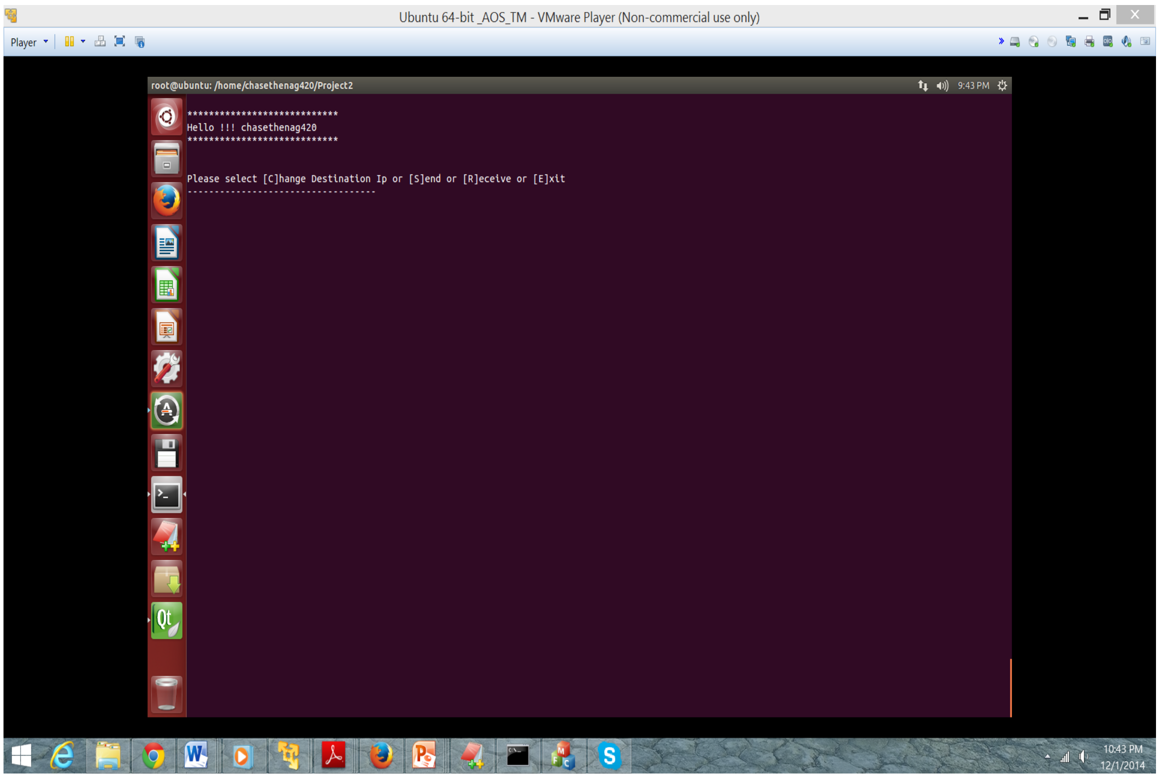
Above script copies required files from Project3 directory to cse536 and linux-3.13.0/drivers/char/cse536 and builds the required modules and make your program to run which you can directly start communicating with other party once the other party is also ready with above steps. If you are running as normal user it will ask for root password.

The sample execution results are given below as screenshots:


Ubuntu 64-bit _AOS_TM - VMware Player (Non-commercial use only)
Player
chasethenag420@ubuntu: ~/Project3
6:00 PM
7:00 PM
12/3/2014

```
chasethenag420@ubuntu:~/Project3$ ./test.sh 192.168.0.18 192.168.0.49
```

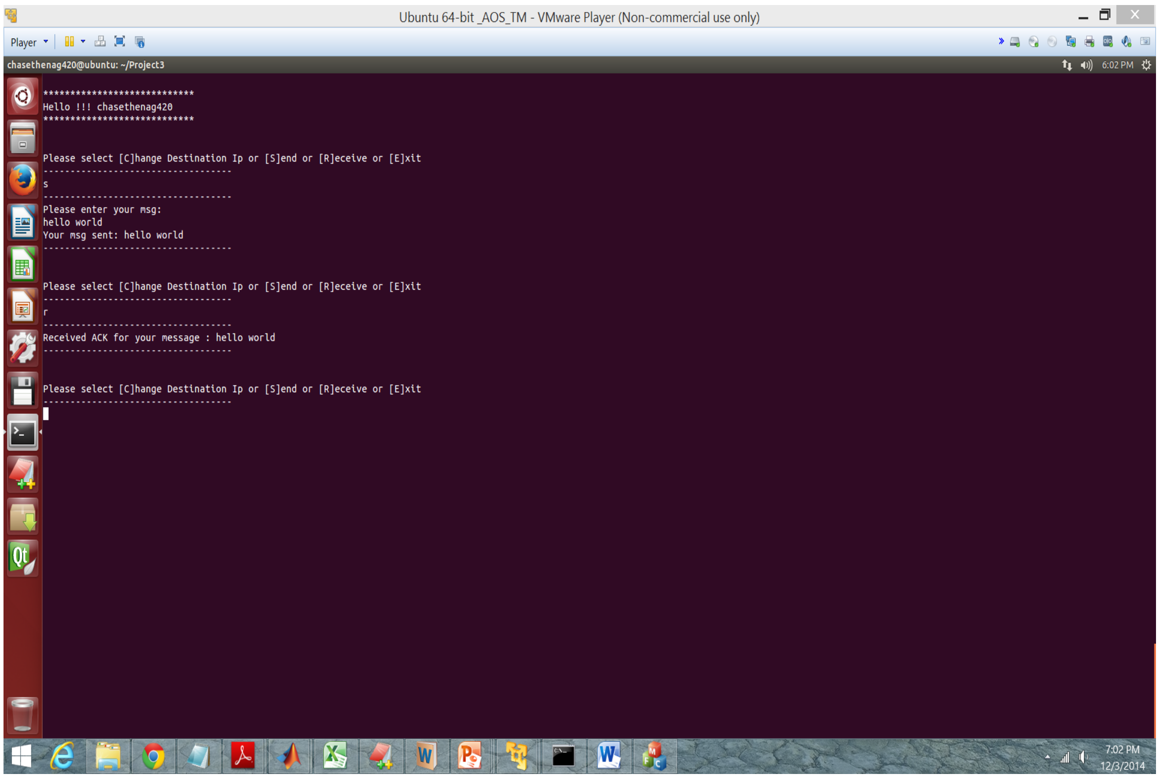
Figure 1: Project3 AOS Start


Ubuntu 64-bit _AOS_TM - VMware Player (Non-commercial use only)
Player
root@ubuntu: /home/chasethenag420/Project2
9:43 PM
10:43 PM
12/1/2014

```
****************************
Hello !!! chasethenag420
****************************


Please select [C]hange Destination Ip or [S]end or [R]eceive or [E]xit
-----------------------------------
```

Figure 2: Project3 AOS Choose option


Ubuntu 64-bit _AOS_TM - VMware Player (Non-commercial use only)
Player
chasethenag420@ubuntu: ~/Project3
6:02 PM
7:02 PM
12/3/2014

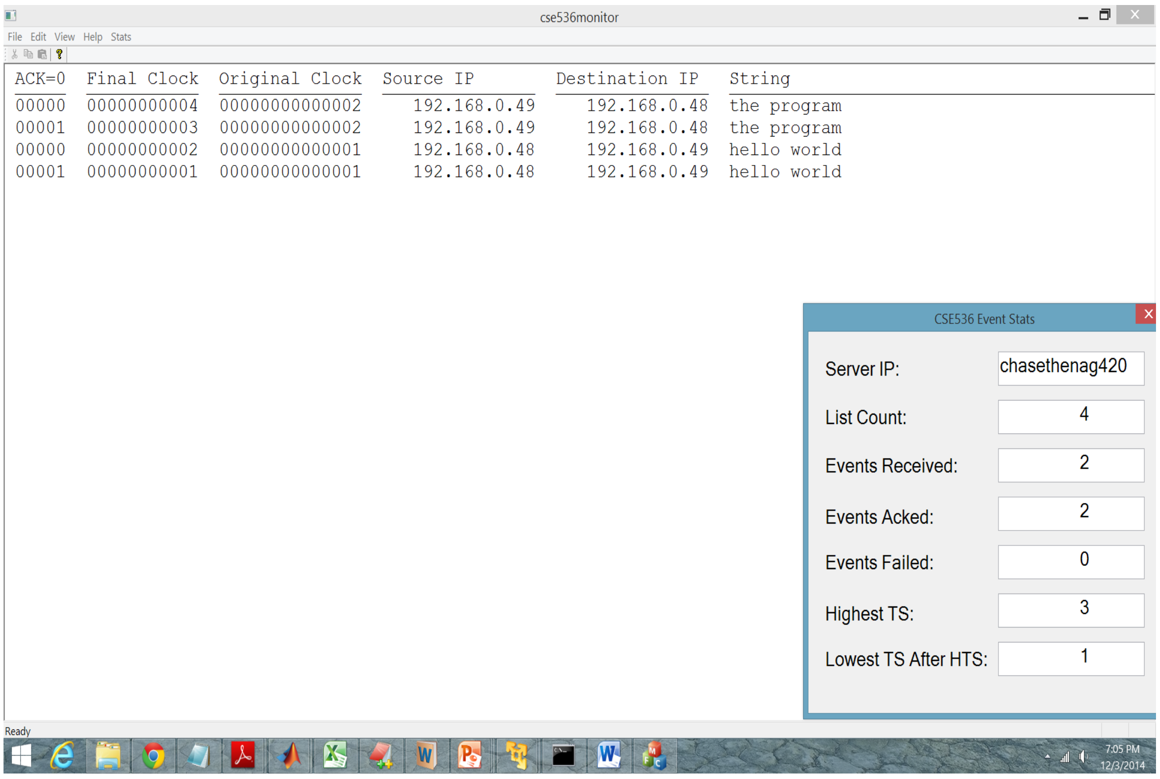
```
****************************
Hello !!! chasethenag420
****************************


Please select [C]hange Destination Ip or [S]end or [R]eceive or [E]xit
-----------------------------------
s
-----------------------------------
Please enter your msg:
hello world
Your msg sent: hello world
-----------------------------------


Please select [C]hange Destination Ip or [S]end or [R]eceive or [E]xit
-----------------------------------
r
-----------------------------------
Received ACK for your message : hello world
-----------------------------------


Please select [C]hange Destination Ip or [S]end or [R]eceive or [E]xit
-----------------------------------
```

Figure 3: Project3 AOS Send Message & Read Message

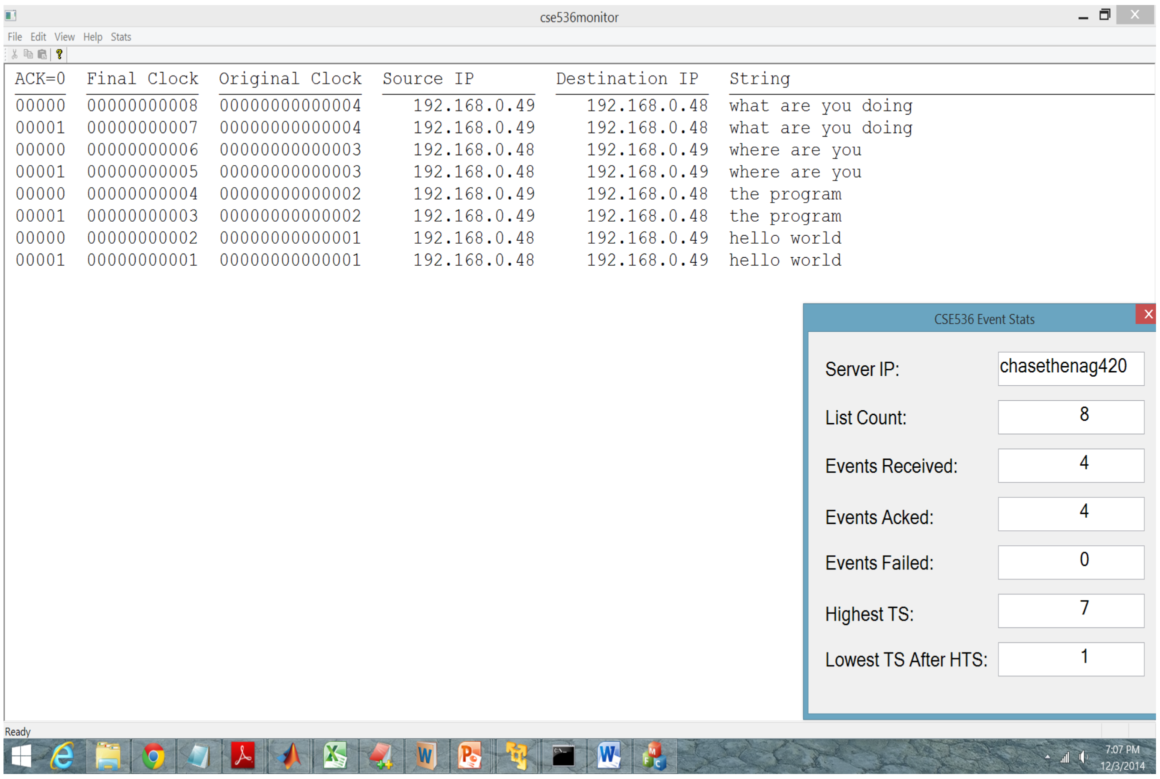


```
ACK=0  Final Clock  Original Clock  Source IP     Destination IP  String
00000  00000000004  00000000000002    192.168.0.49    192.168.0.48  the program
00001  00000000003  00000000000002    192.168.0.49    192.168.0.48  the program
00000  00000000002  00000000000001    192.168.0.48    192.168.0.49  hello world
00001  00000000001  00000000000001    192.168.0.48    192.168.0.49  hello world
```

CSE536 Event Stats

| | |
|---|---|
| Server IP: | chasethenag420 |
| List Count: | 4 |
| Events Received: | 2 |
| Events Acked: | 2 |
| Events Failed: | 0 |
| Highest TS: | 3 |
| Lowest TS After HTS: | 1 |

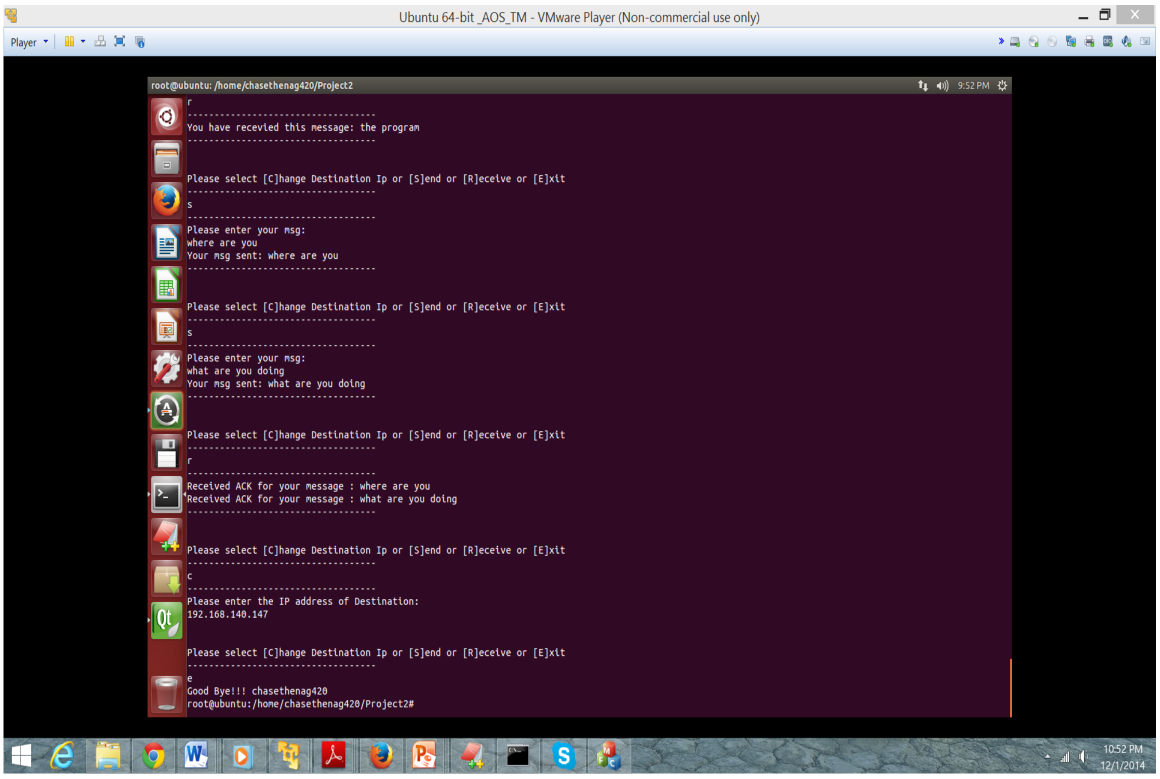
Figure 4: Project3 AOS cse536monitor



```
ACK=0  Final Clock  Original Clock  Source IP     Destination IP  String
00000  00000000008  00000000000004   192.168.0.49   192.168.0.48  what are you doing
00001  00000000007  00000000000004   192.168.0.49   192.168.0.48  what are you doing
00000  00000000006  00000000000003   192.168.0.48   192.168.0.49  where are you
00001  00000000005  00000000000003   192.168.0.48   192.168.0.49  where are you
00000  00000000004  00000000000002   192.168.0.49   192.168.0.48  the program
00001  00000000003  00000000000002   192.168.0.49   192.168.0.48  the program
00000  00000000002  00000000000001   192.168.0.48   192.168.0.49  hello world
00001  00000000001  00000000000001   192.168.0.48   192.168.0.49  hello world
```

| | |
|---|---|
| Server IP: | chasethenag420 |
| List Count: | 8 |
| Events Received: | 4 |
| Events Acked: | 4 |
| Events Failed: | 0 |
| Highest TS: | 7 |
| Lowest TS After HTS: | 1 |

Figure 5: Project3 AOS cse536monitor



```
r
-----------------------------------
You have recevied this message: the program
-----------------------------------


Please select [C]hange Destination Ip or [S]end or [R]eceive or [E]xit
-----------------------------------
s
-----------------------------------
Please enter your msg:
where are you
Your msg sent: where are you
-----------------------------------


Please select [C]hange Destination Ip or [S]end or [R]eceive or [E]xit
-----------------------------------
s
-----------------------------------
Please enter your msg:
what are you doing
Your msg sent: what are you doing
-----------------------------------


Please select [C]hange Destination Ip or [S]end or [R]eceive or [E]xit
-----------------------------------
r
-----------------------------------
Received ACK for your message : where are you
Received ACK for your message : what are you doing
-----------------------------------


Please select [C]hange Destination Ip or [S]end or [R]eceive or [E]xit
-----------------------------------
c
-----------------------------------
Please enter the IP address of Destination:
192.168.140.147


Please select [C]hange Destination Ip or [S]end or [R]eceive or [E]xit
-----------------------------------
e
Good Bye!!! chasethenag420
root@ubuntu:/home/chasethenag420/Project2#
```

Ubuntu 64-bit _AOS_TM - VMware Player (Non-commercial use only)
Player
root@ubuntu: /home/chasethenag420/Project2
9:52 PM
10:52 PM
12/1/2014

Figure 6: Project3 AOS change destination ip & exit

## 4 Conclusion

In this project we have worked on ordering of messages in distributed enviroment using logical clock.

## 5 References

1. Lamport timestamps
   http://en.wikipedia.org/wiki/Lamport_timestamps

2. Happened-before
   http://en.wikipedia.org/wiki/Happened-before