Network Architecture I

Project Report

Sri Harsha Chennavajjala(16210893)

Teja Garidepally(16183523)

Raj Kiran Reddy Munnangi(16210167)

**Introduction:**

In this project, we have developed a simple TCP client and server programs using GENI for simple message exchanges and simple file transfers.

**Initial setup:**

* Geni account creation
* Slice creation
* Resource allocation.

**Part I: GENI/Socket programming Warm‐up**

* Client and server java code
* Initial lookup for part 2.

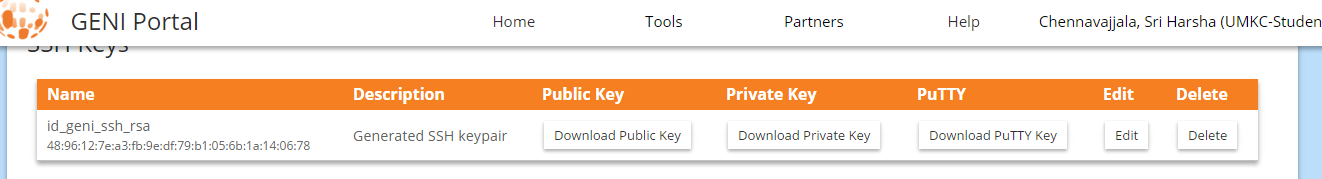
**Requirements:**

* Geni account and a slice to add resources where we can work on.
* SSH Keypairs for logging in to nodes and running scripts.
* Java scripts for communication between client and server.

**Steps involved:**

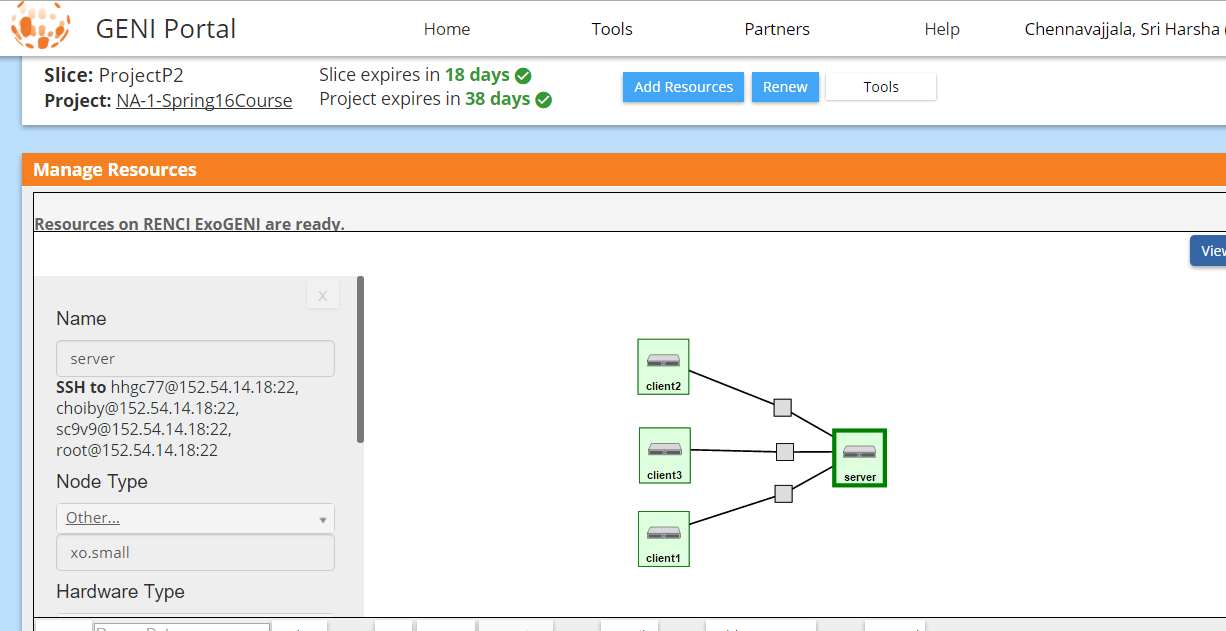
**Step 1:**

After logging in to GENI Portal, we have downloaded the SSH keypairs for authentication purposes.



**Step 2:**

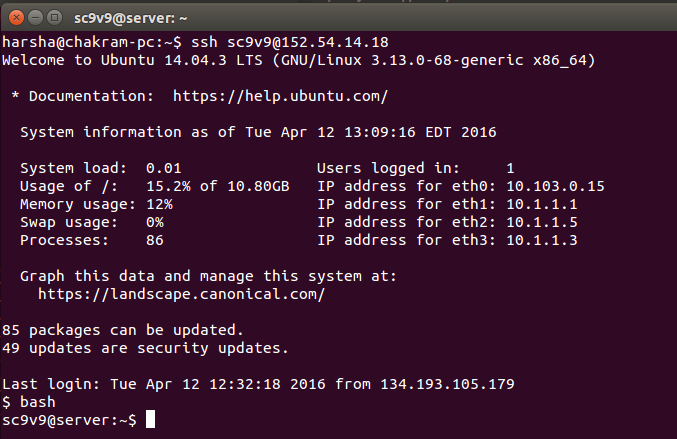
Created a slice “ProjectP2” and added four resources i,'e four VM’s. One of the VMs acts as server and the remaining three will act as clients. In the next step, I’ve established connection between these four resources.



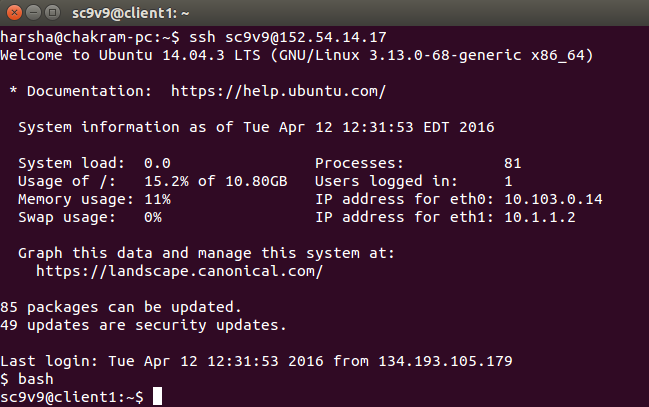
**Step 3:**

Logged in to our client and server nodes using SSH command to the client IP address and server IP address.

Screenshots showing client and server login:



Server Login Screenshot



Client1 login screenshot

**Step 4:**

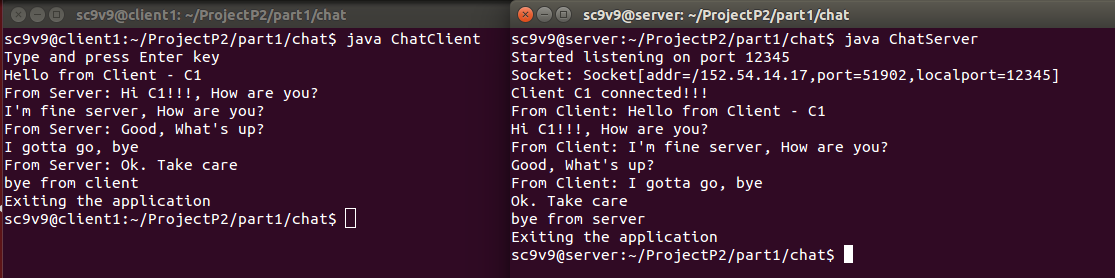
Part 1 contains two tasks, communication through simple messages and file exchanges.

For task 1(a), communication through simple messages we developed two java files ChatClient.java and ChatServer.java.

The mechanism in ChatClient.java and ChatServer.java is as follows:

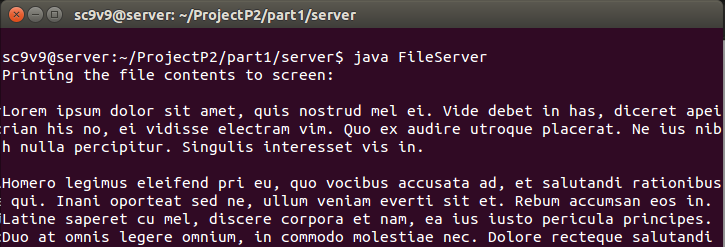
* Running ChatServer.java starts server and waits for the client to connect to it.
* After connection establishment client can send message to server and server can send message to client.
* Server or client needs to send “bye from server” or “bye from client” message to terminate the connection.

Below is screenshots of simple chat between client and server:

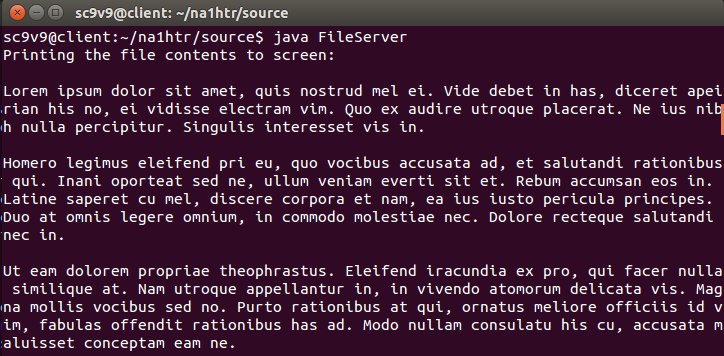


Client chat window Server chat window

For task 1(b) we have developed two applications FileClient and FileServer, running FileClient sends a text file to server and server first displays the contents of received file and saves the file in local system.

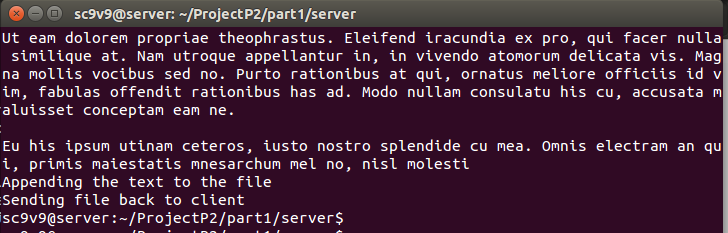


Server displaying the received file contents

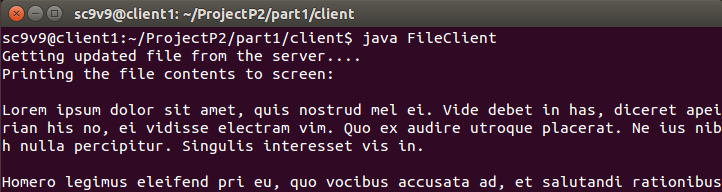


Server displaying the file contents after received from client

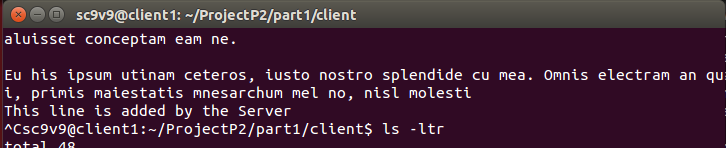
Server appends the one more line to received file and sends the updated file back to client. Client now displays the file on screen after receiving the full updated file.



Server appending new line to the received file



Client getting the updated file from server

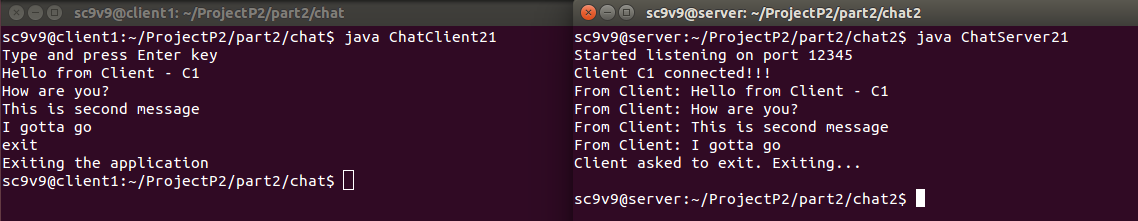


Updated file at the client end

Step 5:

Part 2 contains four tasks - development of client-server chat application

Part 2(a): In this part, the chat server will start listening to the client. Once the client is connected, the server displays all the messages received from client. If the client types “exit”, both the client and server quits.

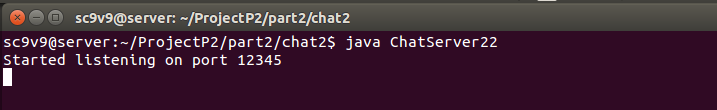


Client chat window Server chat window

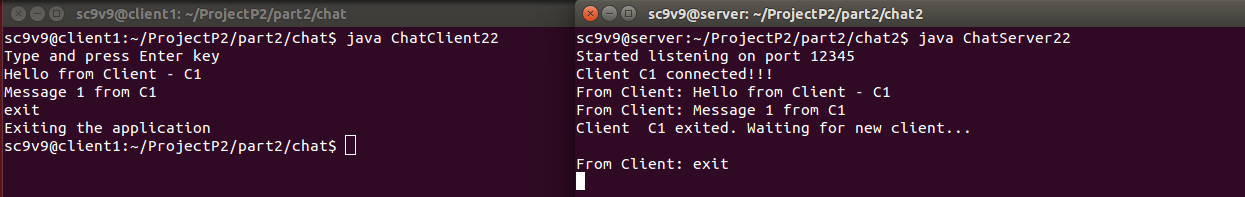
Part 2(b):

This part is similar to part 2(a) but the server remains active and waits to get new connection from client.

Here we test this scenario, with two clients C1, C2 and a Server. First client C1 connects to server sends some messages and exits the application. The server remains active. After sometime, client C2 connects to server and sends some messages. In the end, client C2 exits. Below screenshot shows the workflow.

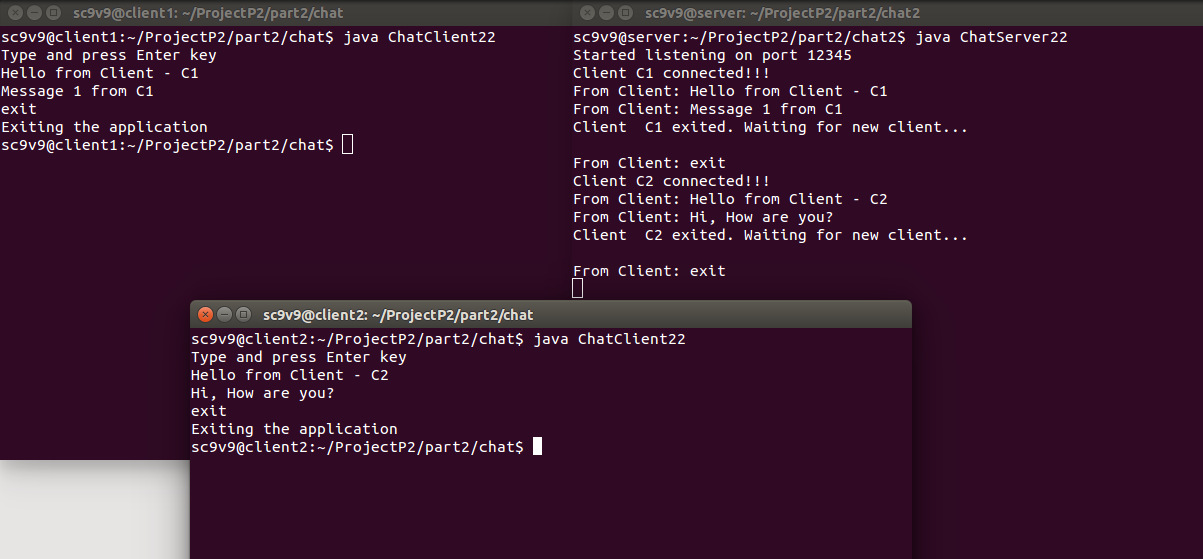


Server started and waiting for the clients to connect



Client C1 chat window Server chat window

Client C1 connected and send messages and exited. Server waiting for new clients

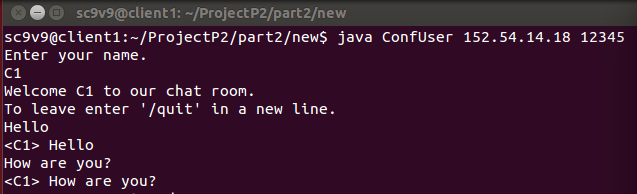


Once a new client C2 connected to the server, the server displays the message “Client C2 connected!!!”

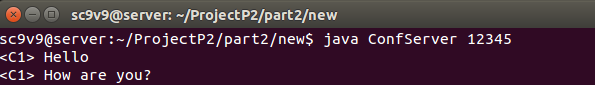
The client C2 sends messages and the messages will be displayed on server. Finally C2 disconnects from server by sending “exit” message.

Part 2(c):

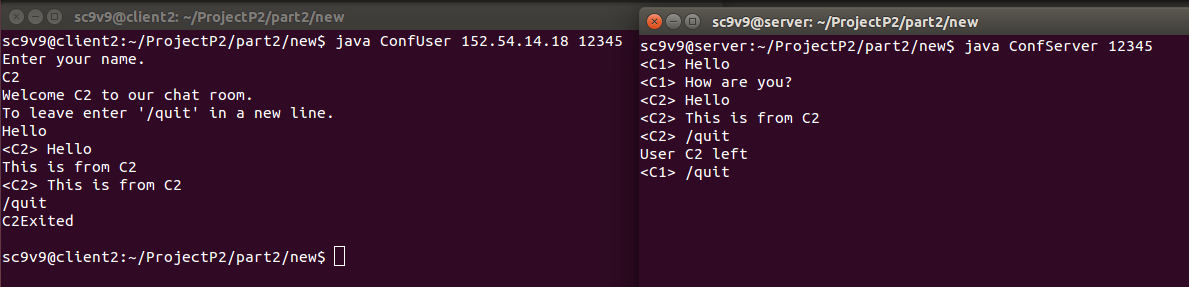
In this case, the server should be capable of serving multiple clients simultaneously. All the messages from the clients will be displayed on server side.



Client C1 sending messages to Server

  
Server displays all the messages typed by client C1

Next C2 enters the chat, and sends messages to server. Once the clients are done with chat, they can exit the application by typing “/quit”.



C2 chat window Server chat window

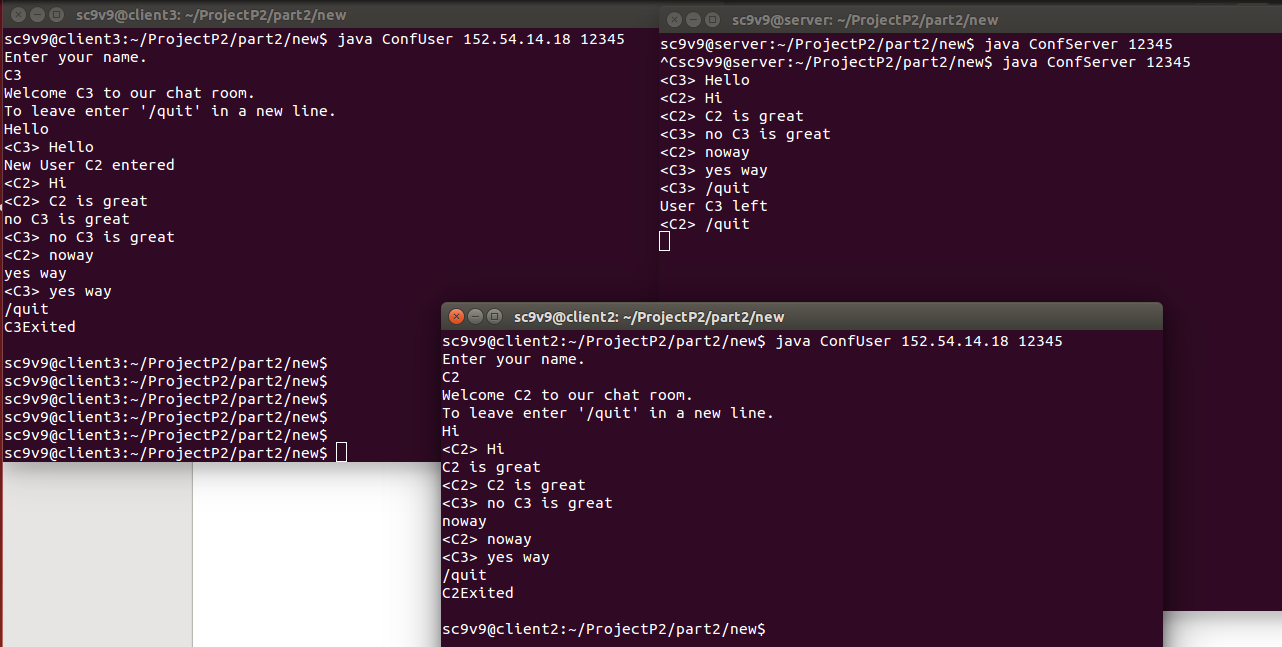
In the above screenshot, server is displaying the messages from both client C1 and client C2 simultaneously.

Part 2(d):

In this part, the messages sent by the clients will be passed through server and will be distributed to all the clients. So, if a client types a message, the message will be displayed on all the other clients.

Here I’ve used wo clients C2, C3 and a server. Below screenshots demonstrates the workflow.

**C3 chat window Server chat window**



**C2 chat window**

In the above screenshot all the messages from C2 and C3 were displayed on server. Also all the messages of C2 were displayed on C3 and vice versa.

Source code Link:

<https://drive.google.com/open?id=0B8RcAeWxxOPWNi1nSUwtblVMVzg>