Operating System Lab Assignmnet.

Scheduling Algorithms

1. **Round Robin Scheduling**

**Input: roundrobin.m**

clear

disp('ROUND ROBIN ALGORITHM')

num\_job = 5 % No. of jobs

process\_time = [5 8 2 1 4] % Execution time of each job

process\_time\_copy = process\_time;

time\_slot\_interval = 2; %Length of one time slot

totalprocess\_time= sum(process\_time) %Total execution time of all the jobs

temp\_time = totalprocess\_time;

rnd\_robin = [];

throughput = [];

wait = [];

colors = 'rgbkm';

resp\_time = [];

z = 1;

while (temp\_time ~= 0)

for i=1:1:num\_job

if process\_time(i)>0

if(z == 1)

resp\_time(i) = totalprocess\_time - temp\_time;

end

if(process\_time(i)>time\_slot\_interval)

process\_time(i) = process\_time(i)- time\_slot\_interval;

temp\_time = temp\_time- time\_slot\_interval;

else

temp\_time = temp\_time - process\_time(i);

process\_time(i) = 0;

rnd\_robin(i) = totalprocess\_time-temp\_time;

end

wait(i) = totalprocess\_time - temp\_time-process\_time\_copy(i);

end

end

z = 0;

end

rr\_sum = sum(rnd\_robin);

wait\_sum = sum(wait);

rr\_tput = rr\_sum/num\_job;

avg\_wait\_time = wait\_sum/num\_job;

%to display

wait

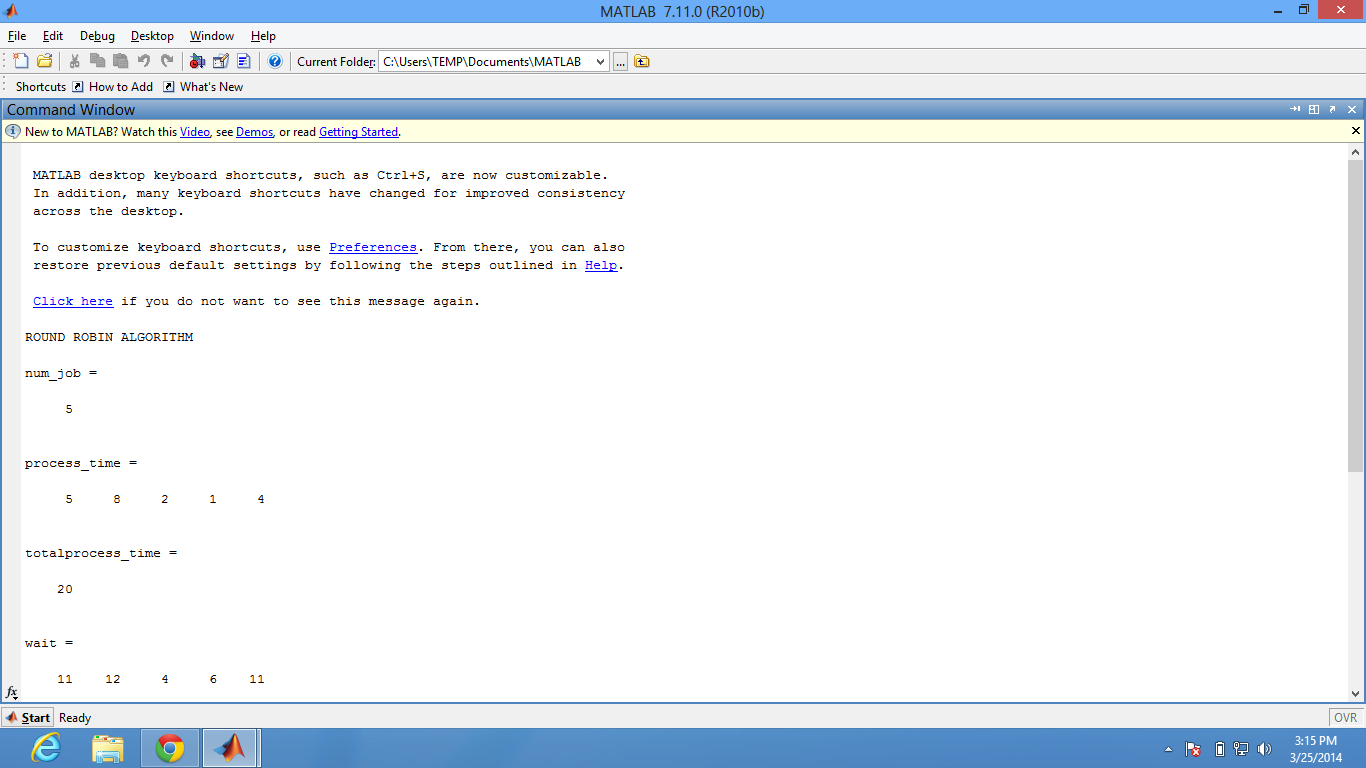
rr\_tput

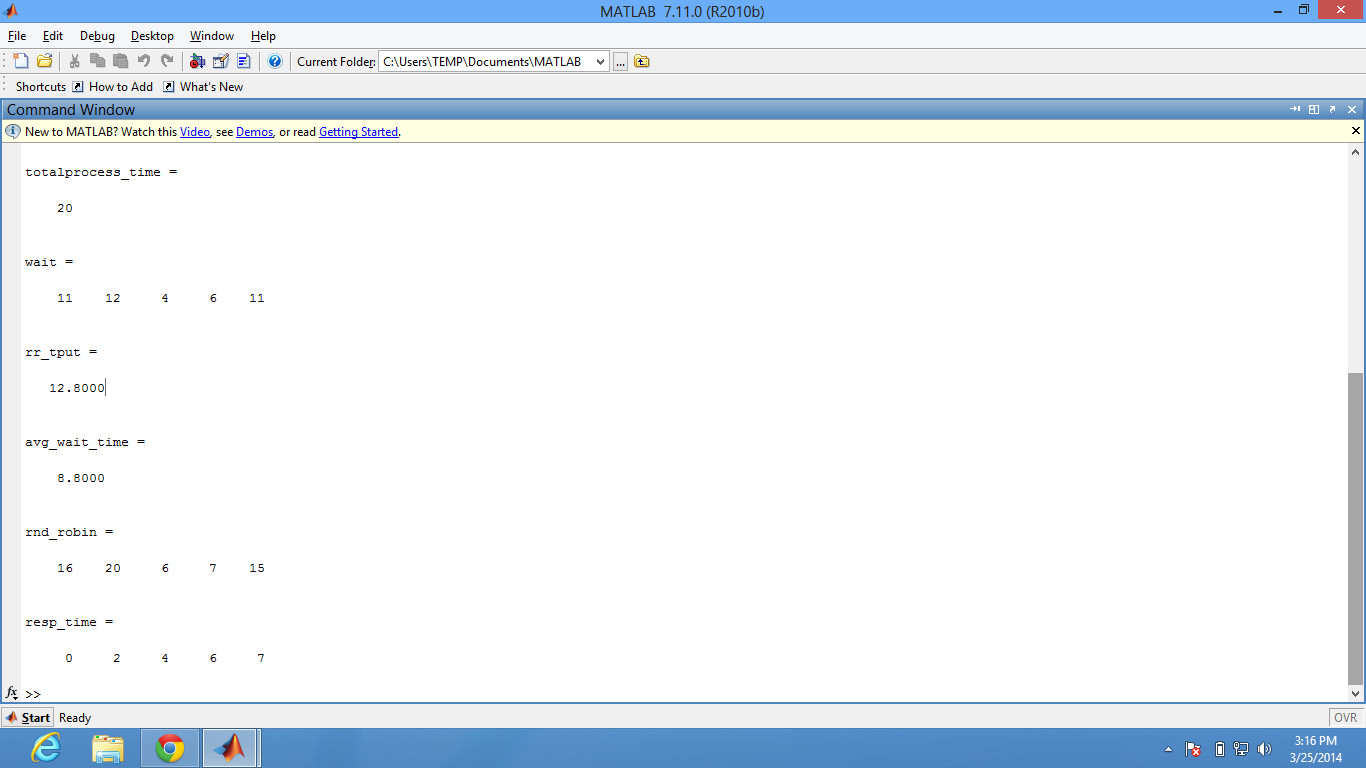
avg\_wait\_time

rnd\_robin

resp\_time

**Output:**





**Rationale:**

**Given the number of jobs, process time of each job and time slot interval we are scheduling the jobs according to round robin algorithm.**

**The outer loop says while all the processes don’t get completed.**

**Inside that we have a for loop running for each process turn by turn and if the remaining time for process of that corresponding process is greater than the quantum time, we just run that process for the quantum time. If it is less than the quantum time, we run it fully and then the time to process the remaining part of it becomes zero as the entire process would have completed.**

**z is a flag, which is made to run only in first time to know the starting time of each process.**

**rnd\_robin variable says the time at which that job completed.**

**Then we calculate the round\_robin throughput by average of all rnd\_robin and average wait time as averages of all wait time.**

**2. Shortest Job First.**

**Input :**

**%no of jobs**

**n = 5;**

**%assuming arrival time is in sorted order**

**arrival\_time = [0;1;2;3;4];**

**processes = [1:1:n];**

**burst\_time = [10;3;2;4;5];**

**turnaround = [ ];**

**waitingtime = [ ];**

**burst\_time(n+1) = 999;**

**sum\_bt = 0**

**for i=1:1:n**

**sum\_bt = burst\_time(i) + sum\_bt**

**end**

**time = 0**

**sum\_turnaround=0**

**sum\_wait = 0**

**while time<sum\_bt**

**smallest = n+1;**

**for i=1:1:n**

**if arrival\_time(i) <= time && burst\_time(i) > 0 && burst\_time(i)<burst\_time(smallest)**

**smallest = i;**

**end**

**end**

**turnaround(i) = time + burst\_time(smallest)-at(smallest)**

**waitingtime(i) = time - at(smallest)**

**sum\_turnaround = sum\_turnaround + turnaround(i)**

**sum\_wait = sum\_wait + waitingtime(i)**

**time = time + burst\_time(smallest)**

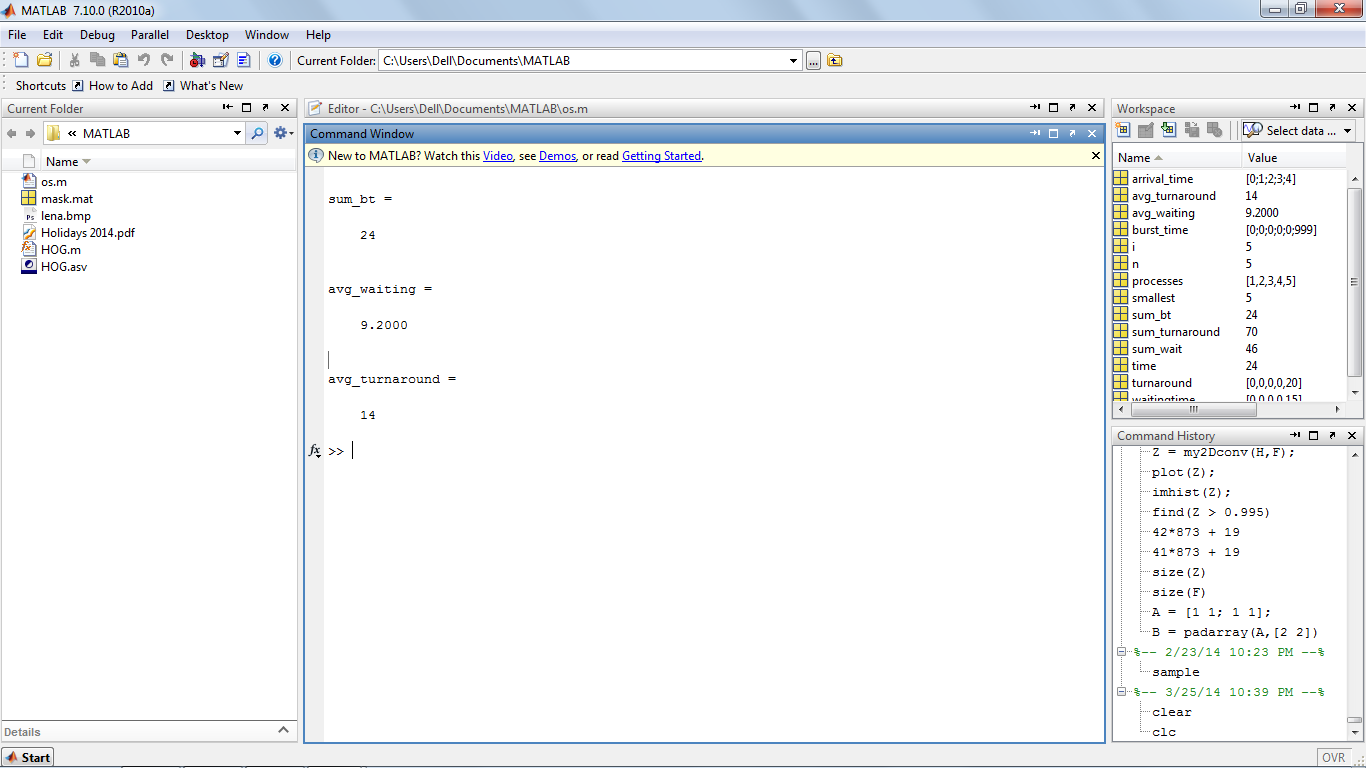
**burst\_time(smallest)=0**

**end**

**avg\_waiting = sum\_wait \* 1.0 /n**

**avg\_turnaround = sum\_turnaround\* 1.0 /n**

**OUTPUT :**



**Rationale:**

**Given the number of jobs and process time of each job we are scheduling the jobs according to shortest job first algorithm.**

**The outer while loop guarantees that the program runs till all the jobs have completed execution fully without leaving any time-gap in between.**

**Inside it, we are running a for loop to find the shortest job. To compare the jobs, we are making a dummy extra job with its burst time 999, which is too high. If any process has burst time greater than or equal to 999, the burst time limit for the dummy job should be increased.**

**Then we run that shortest job and make the burst time zero, as it has already completed.**

**INTERPROCESS COMMUNICATION:**

**Server Code:**

/\*\*

\*

\* @author Aashish and Kesha

\*/

/\*\*

\* This process work as the server process.

\*

\*/

import java.net.\*;

import java.io.\*;

public class process1 {

static ServerSocket ServerSocket = null;

static Socket clientSocket = null;

static String inputLine, outputLine="";

static PrintWriter out[] = new PrintWriter[10];

static BufferedReader in[] = new BufferedReader[10];

//receiver re[] = new receiver[10];

static int count=0;

static String name[] = new String[10];

static int flag=0;

public static void main(String[] args) {

try {

//creating a server Socket

ServerSocket = new ServerSocket(4444);

//can use any port number which is greater then 1024

} catch (IOException e) {

System.out.println("Could not listen on port: 4444");

System.exit(-1);

}

Acceptor ac = new Acceptor(ServerSocket);

ac.start();

System.out.println("The Server has started.");

try {

//accepting a clients request// wait till a client connects

// clientSocket = ServerSocket.accept();

//------------------connection Established----------------------

String msg = "no message";

while (true) {

//flag=0;

//reading from a client

/\* if(in.ready()){

inputLine = in.readLine();

msg = inputLine;

if (msg.equals("exit")) {

break;

}

System.out.println("echo:" + inputLine);

}\*/

//System.out.println("Message:");

for(int i=0;i<count;i++){

if(in[i].ready()){

msg = in[i].readLine();

System.out.println(i+" : "+msg);

for(int j=0;j<count;j++){

if(j!=i){

out[j].println(i+": "+ msg);

}

}

}

}

BufferedReader stdIn = new BufferedReader(new InputStreamReader(System.in));

if(stdIn.ready()){

outputLine = stdIn.readLine();

//outputLine = "Your message: " +outputLine ;

//sending message to a client

for(int i=0;i<count;i++){

out[i].println(outputLine);

}

if (outputLine.equals("exit")) {

//flag=1;

break;

}

}

/\*out = null;

in = null;\*/

}

//out.close();

//in.close();

//clientSocket.close();

ServerSocket.close();

} catch (IOException e) {

System.out.println("Accept failed: 4444");

System.exit(-1);

}

}

public void givethemsg(String message,int id){

flag=1;

System.out.println(id+": "+message);

for(int i=0;i<count;i++){

if(i!=id){

out[i].println(id+": "+message);

}

}

}

public void dothejob(Socket cs) throws IOException{

//create an output stream

out[count] = new PrintWriter(cs.getOutputStream(), true);

//create an input stream

in[count] = new BufferedReader(new InputStreamReader(cs.getInputStream()));

//receiver re = new receiver(cs,in[count],count);

//re.start();

//System.out.println("yahan aya" + count);

out[count].println("What is your name buddy??");

Acceptor ac = new Acceptor(ServerSocket);

ac.start();

//re[count] = re;

count++;

}

}

**Server Acceptor Thread:**

/\*

\* This threads works in the background and waits for a client to connect.

\*/

import java.io.IOException;

import java.net.ServerSocket;

import java.net.Socket;

import java.util.logging.Level;

import java.util.logging.Logger;

/\*\*

\*

\* @author Aashish

\*/

public class Acceptor extends Thread{

ServerSocket sersoc= null;

Socket cs;

process1 pr1= new process1();

public Acceptor(ServerSocket ss){

sersoc = ss;

}

public void run(){

try {

cs = sersoc.accept();

pr1.dothejob(cs);

} catch (IOException ex) {

Logger.getLogger(Acceptor.class.getName()).log(Level.SEVERE, null, ex);

System.out.println(ex.getMessage());

}

}

}

**Client Code:**

/\*\*

\*

\* @author Aashish and Kesha

\*/

/\*\*

\* This process work as the client process.

\*

\*/

import os\_pack1.\*;

import java.net.\*;

import java.io.\*;

public class process2 {

public static void main(String[] args) {

Socket Socket = null;

PrintWriter out = null;

BufferedReader in = null;

try {

//create a client Socket

Socket = new Socket("localhost", 4444);

//-------connection established---------

} catch (UnknownHostException e) {

System.err.println("Don't know about host: localhost.");

System.exit(1);

} catch (IOException e) {

System.err.println("Couldn't get I/O for " + "the connection to: localhost.");

System.exit(1);

}

try {

String msg = "no message";

BufferedReader stdIn = null;

System.out.println("You have been Connected. You may type your messages to be sent.");

while (true) {

//create an output stream

out = new PrintWriter(Socket.getOutputStream(), true);

//create an input stream

in = new BufferedReader(new InputStreamReader(Socket.getInputStream()));

//System.out.println("Message:");

stdIn = new BufferedReader(new InputStreamReader(System.in));

String userInput="";

if(stdIn.ready()){

userInput = stdIn.readLine();

//userInput = "Your message: "+ userInput;

//sending message to the server

out.println(userInput);

}

if (userInput.equals("exit") || msg.equals("exit")) {

break;

}

//receiving message from the server and printing

if(in.ready()){

msg = in.readLine();

System.out.println("echo: " + msg);

}

out = null;

in = null;

}

out.close();

in.close();

stdIn.close();

Socket.close();

} catch (Exception e) {

}

}

}

**Rationale:**

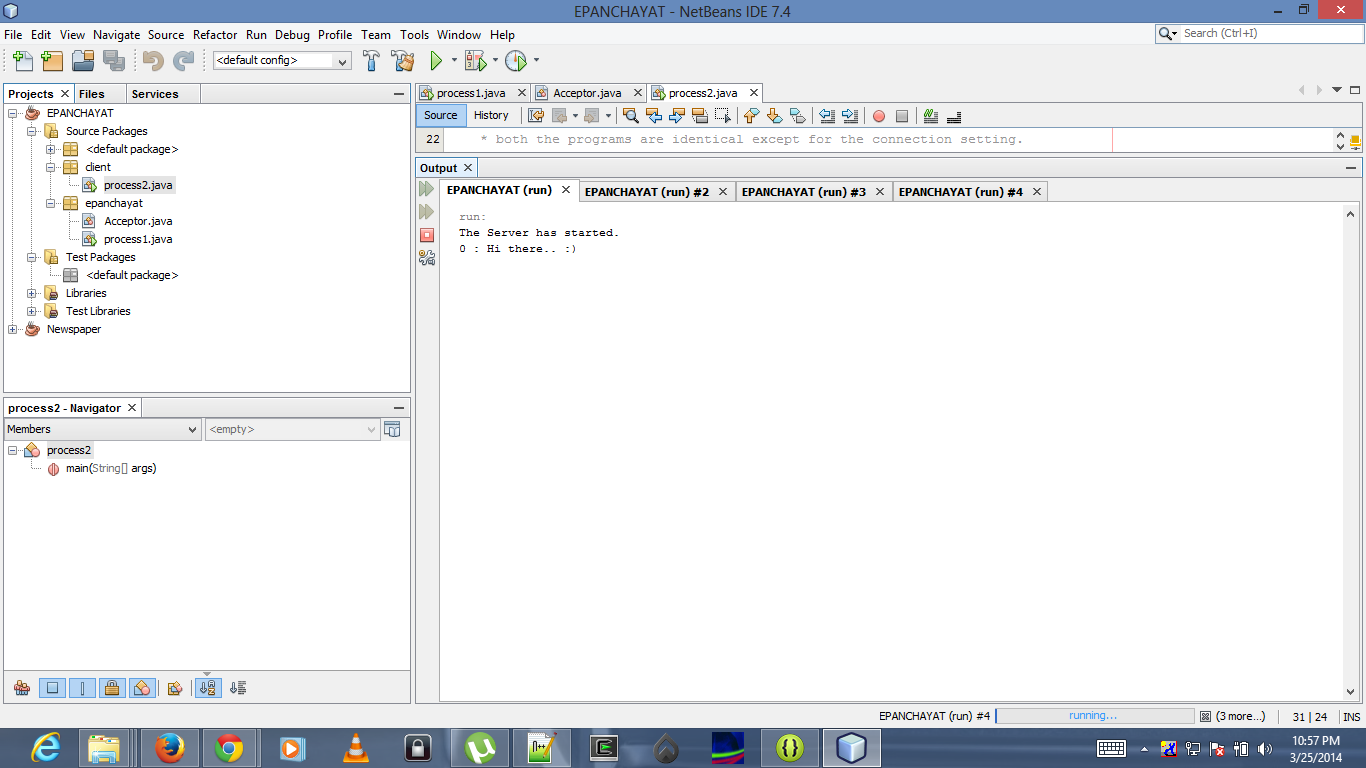
**Server Process and Server Acceptor Thread:**

The server process first of all creates a socket at port numbered “4444”. If that socket is created without any exception/error, then the server starts the Server Acceptor thread. This thread is provided with the Socket that the server just created and hence the thread waits for the clients to connect to that socket and notify the server as soon as any client gets connected to it. This thread works in the background and hence does not blocks the main server process. Now the server, on getting notified by the Acceptor thread, executes the “dothejob(Socket ClientSocket)” function mentioned in the code above. This method takes the ClientSocket of the client, that gets connected to the ServerSocket, as the parameter and hence create an instance of the input and output buffer for that client. Then server continuously keeps on checking the InputStream of all the Clients that are currently connected to the server and if found one ready, then prints that message to the output buffer of all the clients except the one that has sent the message. Server has created an InputBuffer Reader for itself so that if server types and sends a message then that message is sent/written to the output buffer of all the clients currently connected to the server.

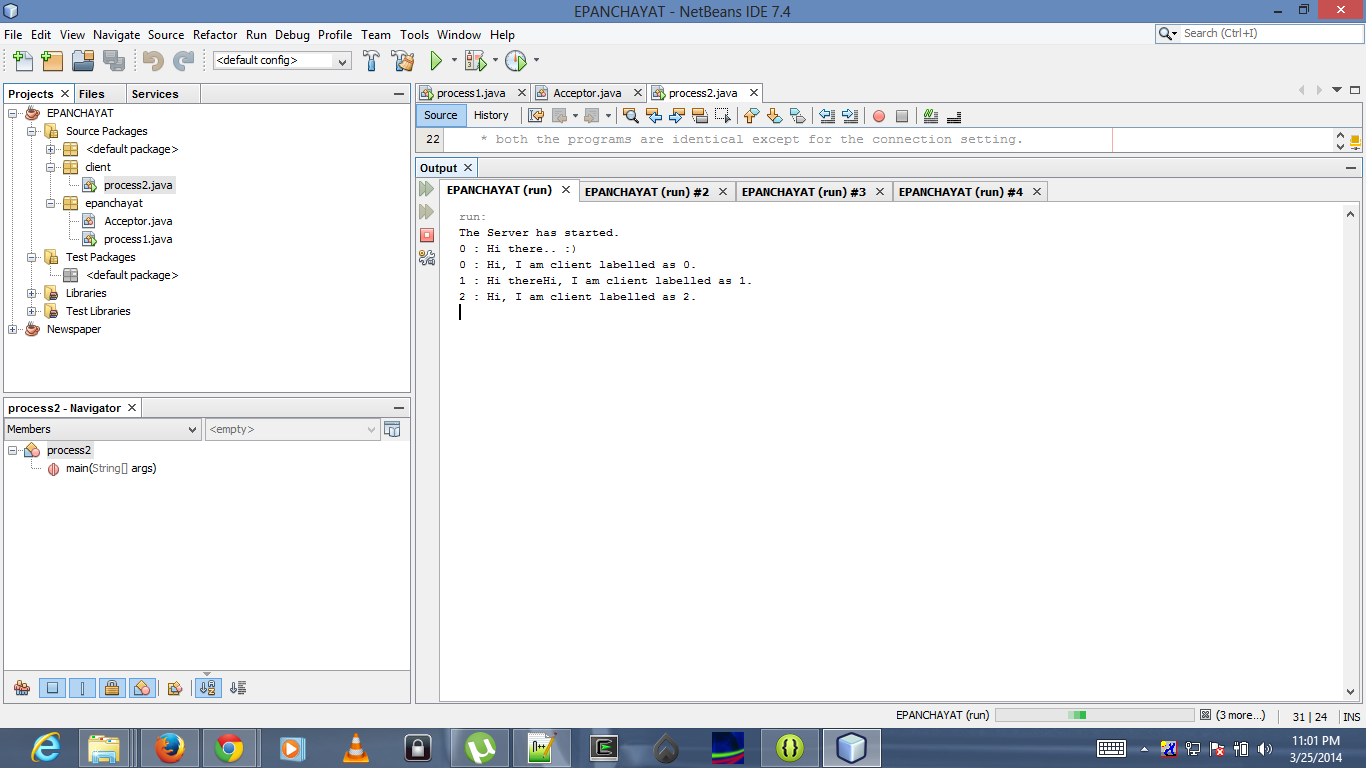
**Client Process:**

The client code works the same as it was provided by the professor. No changes are made into it. As soon as a client gets connected to the server, it can chat on that server with other clients.

-> The following screenshot shows how server looks when it starts running after successfully creating its socket on port number 4444.



The following screenshot shows how the server will look upon receiving messages from different clients. Each client is differentiated using a unique ID labelled as 0,1,2, etc.



The below Screenshot shows how the client will look upon receiving messages from two different users labelled as 0,1,2,etc and server labelled as echo: .

