

# Multithreaded Client Server Chat Application Implementation

USING JAVA MULTITHREADING

August 02, 2017

Advance Concurrent Programming

Spring 2017

Student# 201692544

## **Abstract**

In this project, A Multithreaded Client-Server Architecture based application is implemented which works like a group chat room. Since Java has a most dominant impact on Network Programming timeline and this application problem inherently come up with network involvement plus concurrency and synchronization in parallel operations of clients. Java is also good in multithreading solutions. So, Java platform will be used in the development of this project and all the sources will be tested and build using JDK later JVM. The server will run continuously receiving requests from connecting clients and grant access to chat group using Java Socket Programming Primitives. Parallelism in Java is no doubt excellent we can take advantage of multithreaded parallelism but host machines may differ in different situations so the code must be optimized generally for all machines. One of the big challenges is using socket programming granularity of synchronization point will increases as compared to interface programming of Java but it is good to catch up those synchronization granularities for more optimized application development. The server has its backlog as a bottleneck to respond limited set of clients at a time. We are going to scale it to a larger number of users in future. Security is a big challenge for intrusion detection and prevention in chat room

## Summary

Numerous chat applications are available online at stores and we just install them and start using them but what programming paradigm they are using in these applications is a question. It is not much known to all. Similarly, web recourses and network properties, policies and protocols how they work? This all stuff will be covered in this project once we understand this client-server model of operation in the multithreaded environment then we simply be sure to get a glimpse of all operational application around us for chatting, web surfing etc. and justifying why Java parallelism is not worthless in this project. An “Advanced Concurrent Programming” course is a big motivation to me for doing this project also Java concurrency and synchronization is a great source to learn practical implementations of concurrent application. I have implemented *echo* and *daytime* servers as sequential processing applications but this is now a challenge and an interesting thing to get some hands on abstract parallelism and concurrency features of Java. We start with introducing the background of concurrent programming and Java threads then we start the design of our application and end up with a suggested algorithm which is used in the development phase of our application. After development implementation is done on using Java language socket and internet libraries. Finally, we launch the application on the host machine and at then end we will discuss some important aspects of the application for future proceedings.

## Table of Contents

Abstract .....	2
Summary .....	3
Table of Contents.....	4
1. Introduction.....	5
1.1 Background.....	5
1.2 Problem Statement.....	6
1.3 Abstract View .....	6
2. Application Design.....	7
2.1 Programming Paradigm .....	7
2.2 Suggested Algorithm .....	8
2.3 Programming Granularity.....	9
3. Application Development Primitives.....	10
3.1 Creating Threads in Java .....	10
3.2 Measuring Turnaround Time of Thread .....	11
3.3 Concurrent Execution of Threads with Time Slicing .....	12
4. Implementation of Sockets.....	13
4.1 Creating Sockets .....	13
4.2 Creating Input/output Streams .....	14
4.3 Closing Sockets.....	15
5. Execution Documentation .....	16
5.1 Host Machine .....	16
5.2 Launching Application .....	16
5.3 Running Examples.....	17
5.4 Closing Application .....	18
6. Conclusion and Suggestions.....	19
References.....	20

## **Introduction**

### **1.1. Background**

Concurrent programming is among advanced structural programming technique where a number of operations executed in same time elapse. These operations are literally called Threads as the name implies these execution flows are much lighter than heavy context programs/processes containing process control blocks and multiple branches. Sequential programs which execute in a single stream of operations one thread o control whereas concurrent processes have more than one threads of control. Concurrent programming applicability is a factor of dependent architecture and how much resources will be utilized by the programmer. In short concurrent programming is used to exploit resources of a single processor and perform a number of tasks on a single processor using time slicing.

### **Java**

It is a programming language. it was firstly released by Sun Microsystems in 1995. Java is dominating in many applications and many programming concerns. There are many examples/ applications where proper functioning is purely dependent on Java including network, graphics, cluster computing, cell phones, the Internet and object-oriented paradigm. It is a fast and reliable. It is free to download and experience.

### **Thread**

The thread is a lightweight process characterized by execution state, execution stack and its local variables whereas the process is a unit of activity characterized by system resources and current state. Threads are faster. Initiation of threads is faster than process and similarly termination too. Thread switching is faster that process switching. Threads of the same parent process communicate with each other without the interference of kernel. When a process run it assumes all of the systems resources accessibility but thread always bounded in same address space of its program and its own only process resources. Threads use to do background work, for example, login window, image request on the web, updating screen, and multiple audio plays etc.

### **Synchronizing threads in java**

Concurrent programming as the name implies execution of many operations interleave and parallel flows of controls are determined during execution. When there are as many resources as execution context needed and there is no dependency between concurrent operations then there is no need for synchronization. In contrast, when the system has limited set of resources

and interaction among operations is needed then synchronization should be applied. In this case, threads will execute parallel and its main parameter of concurrent programming paradigm. This is a trouble for programmers to harness the capability of concurrent program avoiding the interference of operation. This is done by some synchronization primitives of concurrent programming including Locks, Barriers, Semaphores, Shared variables, Rendezvous etc. Critical section can't be done parallel by a number of threads at a single time. Critical section problem arrives when shared resources are vulnerable by unnecessary usage or usage at the wrong state of program (reading/writing shared variables). Synchronization tells that thread must be suspended or delayed waiting for other thread to leave the critical section.

## **1.2. Problem Statement**

A chat room application in which each user first enters by giving his/her name and start sharing textual messages with all other users available in the chat room. The user can see all the messages with the sender names (given on the start entrance into chat room) with a current receiving time of the message. When anybody leaves the chat room a message to all current chat applicants available in the chat room able to see that corresponding user left. Every message from the user contains its receiving time as well. Here application is to be implemented like a multithreaded two-way communication program with necessary synchronization among the threads.

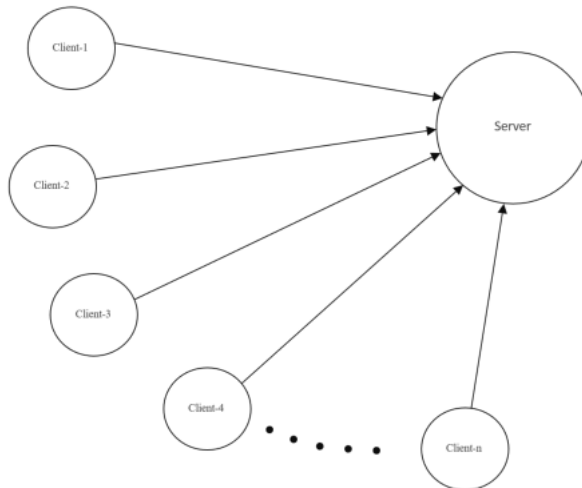
## **1.3. Resolution Abstract**

- One Server running all the time (High Availability) is required to run the application.
- 10's of the client at a single time taking an impression of dedicated server response (Enforcing Concurrency).
- Synchronization solutions to multiple clients i.e. each thread Optimum response speed to the client (less response time).
- Observer Pattern to Spread content received from one client to all others.
- Implementation host will be Intel(R) Core (TM) i7, 6700 HQ CPU @ 2 x 2.6 GHz

## Application Design

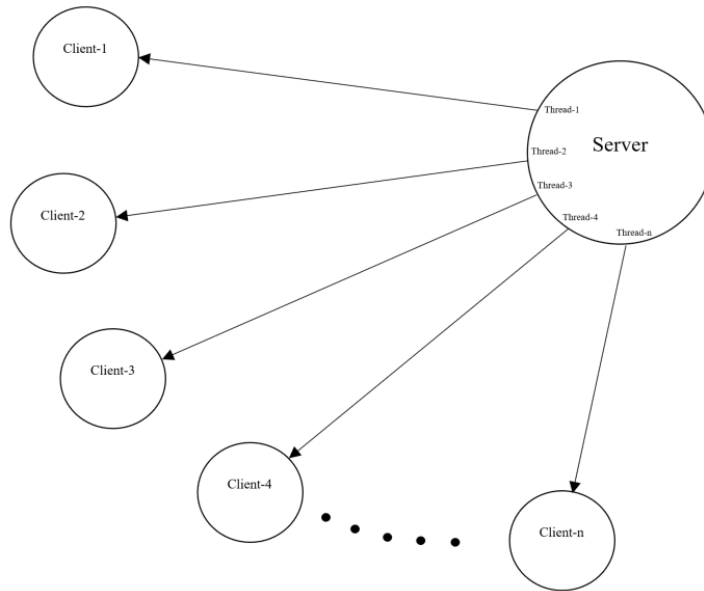
### 2.1 Programming Paradigm

Client-Server Paradigm is used. But the need of this paradigm becomes more interesting when we use java multithreading feature came into action which gives an explicit implementation of threads creation and control synchronization. See **Figure-1**



**Figure-1:** n-Clients Requesting for service

In response to every client server creates a new thread and allow all reads and writes of that client via that thread and implement an observer pattern by receiving requests from the client and throwing a message to all the client threads currently running in on the server. See **Figure-2**.



**Figure 2:** Client request is serviced by server using separate thread for each client

## 2.2 Suggested Algorithm

Algorithm for implementing this chat application based on client-server model is

### Server

Begin

Initiate server process listening for requests for all time

Receive request

Create a thread for client

Create Socket connection with client thread

While Comm. Not Ends

Do input/output streams

Close streams

Close socket

Kill thread

End

### Client

Begin

Initiate client process

Create a thread for making request



```
        Make Request
    Create client thread for Data Transfer
    While Comm. Not Ends
        Do input/output streams
    Close streams
    Close socket
    Kill thread

End
```

### 2.3 Programming Granularity

Programming solution is given at the deeper level of implementation with sockets and input-output streams of communication available in Java. Higher levels of programming abstraction are also available like interfaces of Java but implementing this problem with java network sockets will give lower level control over synchronization. Sockets are complex data structures. Input and output streams are also used in this solution which creates data forwarding and receiving capability on two-way connections of sockets. Internet address which is an internetworking protocol (IP) and port number both necessary to establish a connection to the internet using sockets.

**Machine\_id** is the machine internetworking protocol of host name that is used to access relative IP.

**Port\_No** is identification number of specific application.

### Application Development Primitives

Java is the best candidate for development platform of the network application. Java.net is a package in Java development platform which contains Internet Address resolutions, sockets with TCP and UDP connections.

Here we used TCP protocol which is connection oriented protocol implied that communication connection must be established between client and server before start sending or receiving data from client to server or server to client.

### 3.1 Creating Threads in Java

#### 3.1.1 Using Thread Class

```
public class RunACPThread
{
public static void main(String args[])
{
ACP_Thread my_thread=new ACP_Thread();
my_thread.start();
}
}
class ACP_Thread extends Thread
{
void run() //start() method will invoke this run () method
{
System.out.println("Hello ACP Thread is Created");
}
}
```

#### 3.1.2 Object Oriented Design of Thread Creation

```
Public class RunThreadExtended_SameClass extends Thread
{
public static void main(String args[])
{
new RunThreadExtended_SameClass().start();
}
void run()
{
System.out.println("Hello from ThreadExtended thread\007");
}
}
```

### 3.1.3 Using Runnable Interface

```
class ACPTThreadRunnable implements Runnable
{
// Produce Beep on every thread execution
void run()
{
System.out.println("Hello ACP Thread is Created\007");
}

}

public class RunACPTThread
{
public static void main(String args[])
{
Thread x= new Thread(new ACPTThreadRunnable());
x.start();
}
}
```

### 3.1.4 Object Oriented Design of Thread Creation

```
class RunACPTThread_SameClass implements Runnable
{

public static void main(String args[])
{
Thread x= new Thread(new ACPTThreadRunnable());
x.start();
}
// Produce Beep on every thread execution
public void run()
{
System.out.println("Hello ACP Thread is Created\007");
}
}
```

### 3.2 Measuring Turnaround Time

```
class RunThreadImplementedUsingConstructor implements Runnable
{
Thread myThread;
public void run()
{
System.out.println("Hello ACP Thread is Created using Constructor of
RunThreadImplementedUsingConstructor Class\007"); // Produce
Beep on every thread execution
}
RunThreadImplementedUsingConstructor()
```

```
{
    long millis1 = System.currentTimeMillis();
    System.out.println("Time Before Thread " + millis1);
    myThread = new Thread(this);
    myThread.start();
    long millis2 = System.currentTimeMillis();
    System.out.println("Time After Thread " + millis2);
    long t = millis2 - millis1;
    System.out.println("\nDifference " + t);
}
public static void main (String args[])
{
    //Creating thread with a constructor
    new RunThreadImplementedUsingConstructor();
}
}
```

### 3.3 Concurrent Execution of Threads with Time Slicing

```
public class ParallelThreads extends Thread
{
    String Name = null;
    ParallelThreads(String message)
    {
        Name = message;
    }
    public void run()
    {
        while(true)
        {
            System.out.println(Name);
        }
    }
    public static void main(String args[])
    {
        ParallelThreads x = new ParallelThreads("Thread1");
        ParallelThreads y = new ParallelThreads("Thread2");
        x.start();
        y.start();
    }
}
```

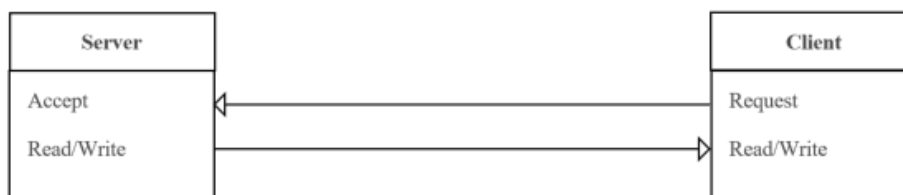
## Implementing Sockets

Sockets are implemented on both server and client side of application for communication. Sockets objects are used to create a connection between client and server. TCP sockets are connection oriented sockets and our chat application also required connection-oriented communication paradigm so we use **Socket** and **Server Socket** objects for communication. Here is the procedure for establishing a connection.

1. Create Network Connection
2. Opening Sockets
3. Creating Input/ Output Streams
4. Closing Sockets

### 4.1 Creating Sockets

The server must be listening on a specific port which is dedicated port and cannot be used for any other network operation because the server must be able to listen to requests all the time on the same port. Client call socket constructor and pass parameters “Internet Address” and “Port number” on which server is running. When the server receives a request from a client it sends back the number after binding it with the client request. Giving the new port number to the client means connections is established. So, at the same time server is listening and writing back a response to the client. Communication is solely dependent on sockets names client socket and server socket as depicted in **Figure-3**.



**Figure 3** : Establishing Connection

#### 4.1.1 Creating Client Socket

Client socket needs Internet Address of Server and Port number on which server is listening to the requests. To open a client socket following source will be implemented.

```

Socket ClientSideSocket;
try
{
ClientSideSocket= = new ClientSideSocket("Name/IP Address",
Port_No);
}
catch (IOException e) {
e.printStackTrace();
}
  
```

Try and catch blocks are guards in case of error/exceptions occurs during establishing connection.

#### 4.1.2 Creating Server Socket

Server socket needs to be implemented on the server. The server socket is just for listening requests from the server on a dedicated port number. The server will continue listening requests from a client on this port number. Once requested is accepted server creates another socket for establishing a complete connection. Process of creating server socket is as below,

```
ServerSocket ServerListeningSocket;
try
{
ServerListeningSocket = ServerListeningSocket(Port_No);
}
catch (IOException e) {
e.printStackTrace();
}
```

#### 4.2 Implementing Input/Output Streams

DataInputStream is used to receive data from another side over a layer of sockets.

```
DataInputStream receiving_data;
try
{
receiving_data= new
DataInputStream(ClientSideSocket.getInputStream());
}
catch(IOException e)
{
e.PrintStackTrace();
}
```

**Dataoutputstream** is used to send data over connection established on socket and server socket.

```
DataOnputStream sending_data;
try
{
Sending_data_data= new
DataOutputStream(ClientSideSocket.getOutputStream());
}
Catch(IOException e)
{
e.printStackTrace();
}
```

For sending data from server to client we can also use java `PrintStream` primitive which implements methods like *write* and *println*.

### 4.3 Closing the sockets

When connection needs to be closed then sockets must be closed explicitly. Input output streams must be closed before closing the socket. For closing the socket just call *close()* methods of input/output and socket objects.

```
try
{
    Sending_Data.close();
    Receiving_Data.close();
    ClientSideSocket.close();
}
catch (IOException e)
{
    e.printStackTrace();
}
```

## Execution Documentation

### 5.1 Host Machine

This project is implemented on Intel(R) Core (TM) i7, 6700 HQ CPU @ 2 x 2.6 GHz machine. All running samples and performance measurements on the same machine and may vary on other machines for the same application. Host machine is used as a server and same time clients will generate requests from this machine too so now we done with multiple command windows dedicated each window to the distinct client and a separate window for Server program. Internet Address of current machine is localhost. Which may be 169.254.x.x/16 which are default address when the machine is disconnected. When it is connected localhost, the request will resolve the request for a current internet address. Port Number we used here is 55555 which any number between 1024-65535 on which this machine will accept the requests.

### 5.2 Running the Application

#### 5.2.1 Launching Server Application

Open command window and go to the directory which contains server file with .java extension and types the following command

```
javac "Server_filename".java
```

JDK will compile the file and if nothing will happen meaning that your server file is successfully compiled. Now it is time to run the file through JVM type following command in the same window

```
java "Server_filename"
```

This time no extension will be used Because on a successful compilation of file it will create Server\_filename.class file which is free to run using JVM and no need to give any extension.

#### 5.2.2 Launching Client Application

Open command window and go to the directory which contains client file with .java extension and does type the following command

```
javac "Client_filename".java
```

JDK will compile the file and if nothing will happen meaning that your server file is successfully compiled. Now it is time to run the file through JVM type following command in the same window

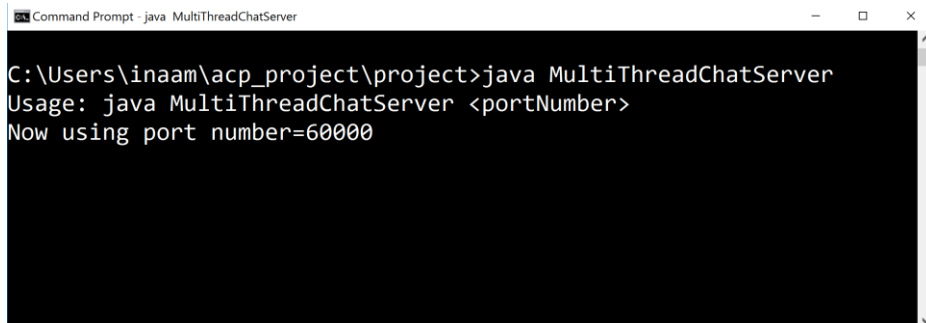
```
java "Client_filename"
```

Same ways this time also no extension will be used Because on a successful compilation of file it will create Server\_filename.class file which is free to run using JVM and no need to give any extension.



### 5.3 Running Examples

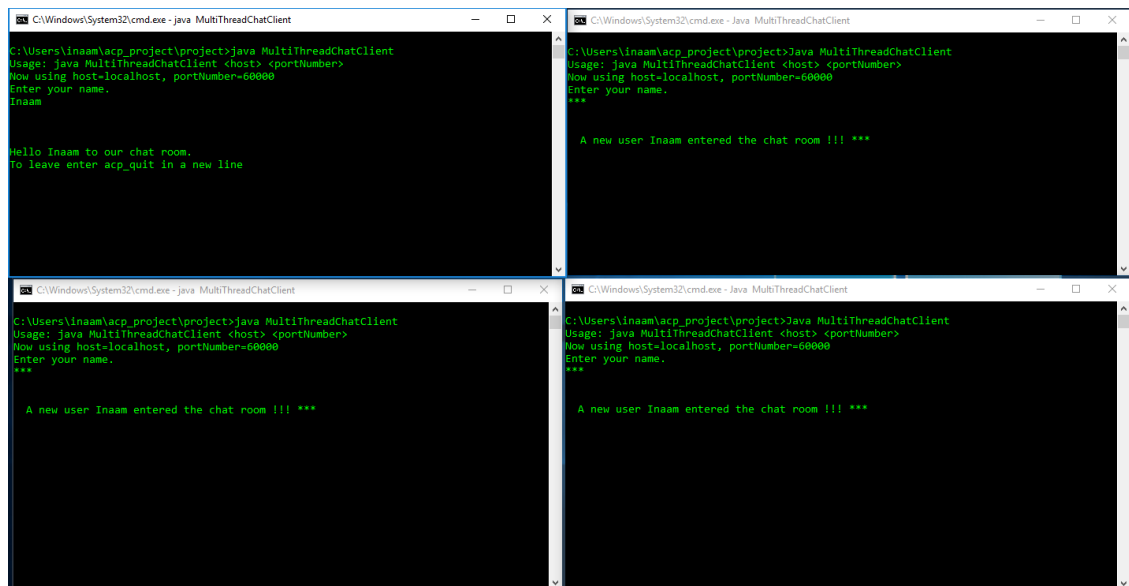
Running server program will be as given in **Figure-4** below,



```
Command Prompt - java MultiThreadChatServer
C:\Users\inaam\acp_project\project>java MultiThreadChatServer
Usage: java MultiThreadChatServer <portNumber>
Now using port number=60000
```

**Figure-4:** Running Server on port 60000

Run Multiple clients by opening multiple windows of command prompt. Each window contains prompt of corresponding client **Figure-5** and **Figure-6**. Refer to **Figure-8** for analysis of java threads taking system resources with 4 clients .



```
C:\Windows\System32\cmd.exe - java MultiThreadChatClient
C:\Users\inaam\acp_project\project>java MultiThreadChatClient
Usage: java MultiThreadChatClient <host> <portNumber>
Now using host=localhost, portNumber=60000
Enter your name.
Inaam

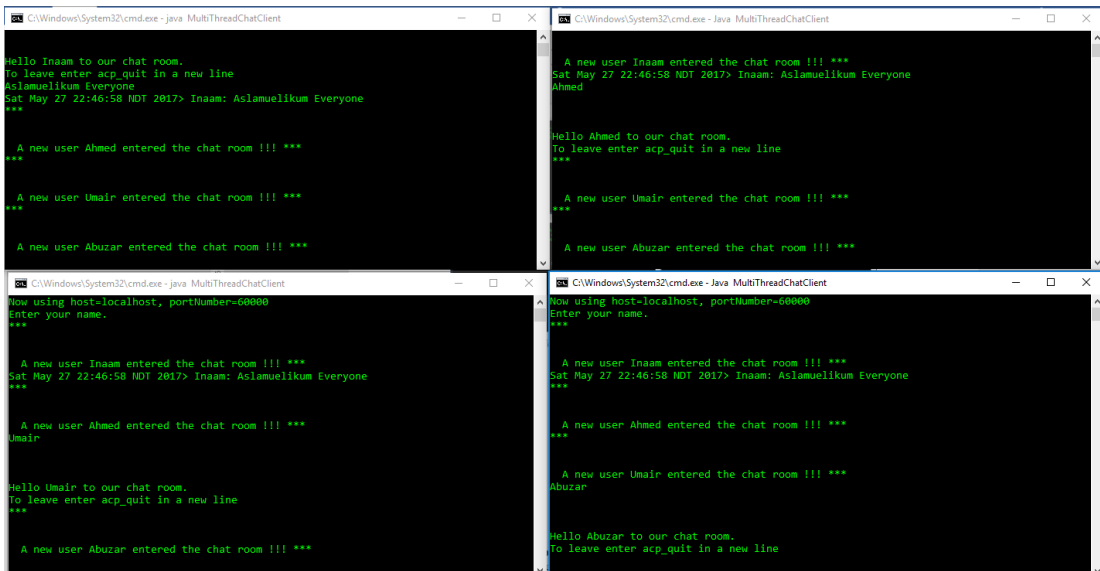
Hello Inaam to our chat room.
To leave enter acp_quit in a new line

C:\Windows\System32\cmd.exe - Java MultiThreadChatClient
C:\Users\inaam\acp_project\project>Java MultiThreadChatClient
Usage: java MultiThreadChatClient <host> <portNumber>
Now using host=localhost, portNumber=60000
Enter your name.
***
A new user Inaam entered the chat room !!! ***

C:\Windows\System32\cmd.exe - java MultiThreadChatClient
C:\Users\inaam\acp_project\project>java MultiThreadChatClient
Usage: java MultiThreadChatClient <host> <portNumber>
Now using host=localhost, portNumber=60000
Enter your name.
***
A new user Inaam entered the chat room !!! ***

C:\Windows\System32\cmd.exe - Java MultiThreadChatClient
C:\Users\inaam\acp_project\project>Java MultiThreadChatClient
Usage: java MultiThreadChatClient <host> <portNumber>
Now using host=localhost, portNumber=60000
Enter your name.
***
A new user Inaam entered the chat room !!! ***
```

**Figure 5:** Four Client Entering Chat Room



```
C:\Windows\System32\cmd.exe - java MultiThreadChatClient
hello Inaam to our chat room.
to leave enter acp_quit in a new line
Aslamuelikum Everyone
Sat May 27 22:46:58 NDT 2017> Inaam: Aslamuelikum Everyone
***

A new user Ahmed entered the chat room !!! ***
***

A new user Umair entered the chat room !!! ***
***

A new user Abuzar entered the chat room !!! ***

C:\Windows\System32\cmd.exe - Java MultiThreadChatClient
A new user Inaam entered the chat room !!! ***
Sat May 27 22:46:58 NDT 2017> Inaam: Aslamuelikum Everyone
Ahmed

Hello Ahmed to our chat room.
to leave enter acp_quit in a new line
***

A new user Umair entered the chat room !!! ***
***

A new user Abuzar entered the chat room !!! ***

C:\Windows\System32\cmd.exe - java MultiThreadChatClient
Now using host=localhost, portNumber=60000
Enter your name.
***

A new user Inaam entered the chat room !!! ***
Sat May 27 22:46:58 NDT 2017> Inaam: Aslamuelikum Everyone
***

A new user Ahmed entered the chat room !!! ***
Umair

Hello Umair to our chat room.
to leave enter acp_quit in a new line
***

A new user Abuzar entered the chat room !!! ***

C:\Windows\System32\cmd.exe - Java MultiThreadChatClient
Now using host=localhost, portNumber=60000
Enter your name.
***

A new user Inaam entered the chat room !!! ***
Sat May 27 22:46:58 NDT 2017> Inaam: Aslamuelikum Everyone
***

A new user Ahmed entered the chat room !!! ***
Abuzar

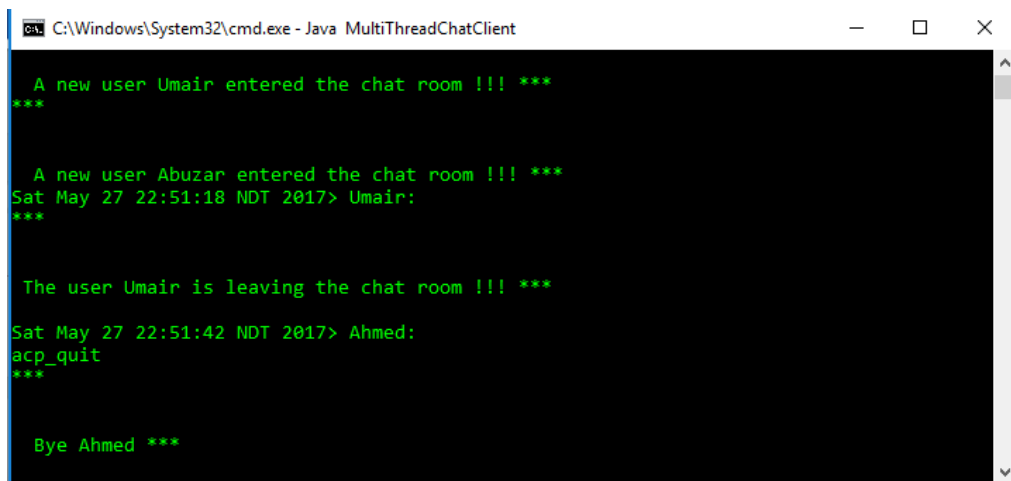
A new user Umair entered the chat room !!! ***
Abuzar

Hello Abuzar to our chat room.
to leave enter acp_quit in a new line
```

Figure 6: Four Clients are Sending Messages in Chat Room

## 5.4 Closing Application

To exit from chat group client must have to enter keyword “quit” and press enter he/she will eventually get out of the chat room. See **Figure-7**.



```
C:\Windows\System32\cmd.exe - Java MultiThreadChatClient

A new user Umair entered the chat room !!! ***
***

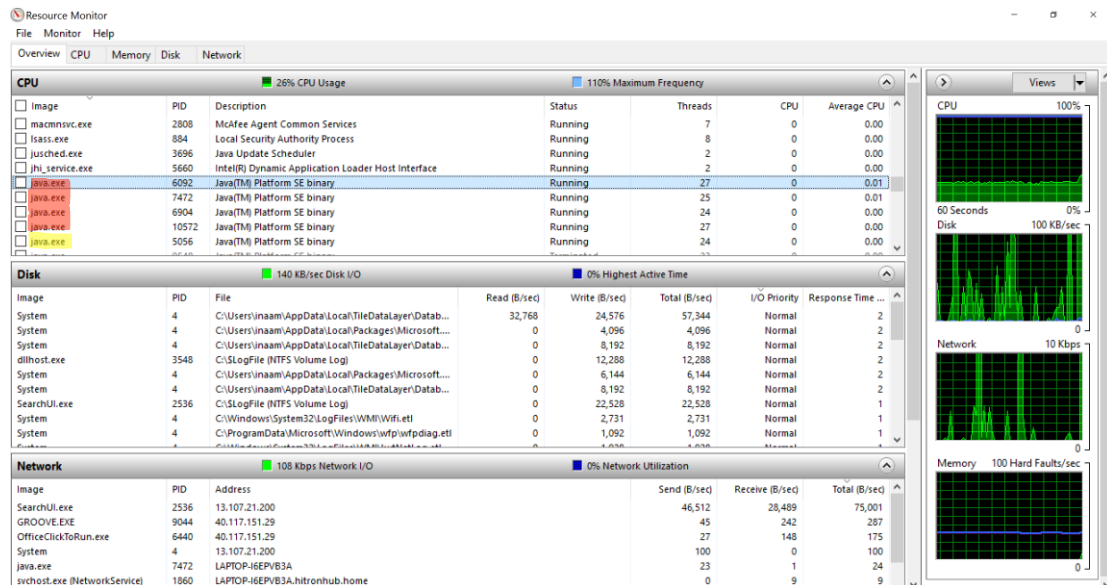
A new user Abuzar entered the chat room !!! ***
Sat May 27 22:51:18 NDT 2017> Umair:
***

The user Umair is leaving the chat room !!! ***

Sat May 27 22:51:42 NDT 2017> Ahmed:
acp_quit
***

Bye Ahmed ***
```

Figure-7: “acp\_quit” will exit client from chat room. Here Ahmed Left the Chat Room



**Figure 8:** Analysis Current Java Threads with 4 Clients Running in Application

## Conclusion and Suggestions

The application is limited to 10 clients only. No authentication for a user to enter the chat room. Authentication is a more important measure for a chat application. Here anyone who knows the public address and port number on which server is listening able to access server and make the connection. It is easy to enter in the chat room. Authentication threads can be used to authenticate the user before entering the chat room. As you can see we have Multithreaded Client-Server application with is only textual communications where no multimedia item can be shared. In future, we can upgrade this application with Java web resources (file sharing, searching media, tracing history etc.). The application can be enhanced to a Graphical Interface also.

## REFERENCES

Andrew, G. R. (n.d.). *Foundation of Multithreaded Parallel and Distributed Networks*.

*Concurrent Programming*. (n.d.). Retrieved from [http://www.di.ase.md/~aursu/Concurrent\\_Programming\\_en.html](http://www.di.ase.md/~aursu/Concurrent_Programming_en.html)

Goetz, B. (n.d.). *JAVA Concurrency in Practice*.

*Java - Multithreading*. (n.d.). Retrieved from [https://www.tutorialspoint.com/java/java\\_multithreading.htm](https://www.tutorialspoint.com/java/java_multithreading.htm)

*Java SE Development Kit 8*. (n.d.). Retrieved from <http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>

*Java Socket Programming Examples*. (n.d.). Retrieved from <http://cs.lmu.edu/~ray/notes/javanetexamples/>

*Java Tutorials*. (n.d.). Retrieved from <https://www.tutorialspoint.com/java/>

*LearnJavaOnline.org Interactive Java Tutorial*. (n.d.). Retrieved from <http://www.learnjavaonline.org/>