# CN Lab (17CSL57)

# Exp 09-10:
## Socket Programming using TCP and UDP

Dr. Ram P Rustagi
Dept of CSE, KSIT
KRP-KSGI
rprustagi@ksit.edu.in

# Ex09-10 Resources

- References:
  - https://docs.oracle.com/javase/tutorial/networking/sockets/index.html
  - https://www.geeksforgeeks.org/socket-programming-in-java/
  - http://www.buyya.com/java/Chapter13.pdf
  - http://www.codejava.net/java-se/networking/java-udp-client-server-program-example
- https://way2java.com/networking/server-to-client-using-udp/
- https://systembash.com/a-simple-java-udp-server-and-udp-client/
- https://www.geeksforgeeks.org/working-udp-datagramsockets-java/

# Lab09/10 Program

- Program 09
  - Using TCP/IP sockets, write a client – server program to make the client send the file name and to make the server send back the contents of the requested file if present
- Program 10
  - Write a program on datagram socket for client/ server to display the messages on client side, typed at the server side.

# Lab Program: UDP Server

- Lab program expectation: Server program
  - A UDP Server program that will communicate with many clients. It should act like a chat server.
  - Port number is taken from command line argument
    - IP is taken by default (all)
  - Whenever it receives data from a client, it should display the same on console (along with client id).
  - Server reads the user response from console (input terminal) and sends it to the client as the response
  - Server forever waits for the data from a client
  - The program runs for ever.
  - All method calls should be checked for errors

# Lab Program: UDP Client

- Lab program expectation: Client program
  - A UDP client program that will communicate with a UDP Server
  - The input command line parameters are
    - Server IP Address
    - Server Port number
    - Ask user to enter the input text on console
    - Client sends the input to server on specified port
    - It should wait for the response from server
    - It should display the response received on terminal
    - The program runs till user enters "Exit"
    - All methods/system calls to be checked for errors

# UDP Data Concepts

- Datagram packet
  - Fully contained message by itself
  - Unaware of any other packet
  - Contains source and destination address/port info
  - Either gets delivered in full or none (lost)
  - No guaranteee of delivery time

- **Java construct for receiving data packet:**
  - `DatagramPacket(byte[] buf, int length)`
- **Java construct for sending data packet:**
  - `DatagramPacket(byte[] buf, int length, InetAddress address, int port)`

# UDP Datagram Socket

- Datagram socket
  - An entity to communicate between two end points
- Java class constructors
  - `https://docs.oracle.com/javase/8/docs/api/java/net/DatagramSocket.html`
  - Socket for client (binds to arbitrary port)
    - `DatagramSocket()`
    - Socket for server (binds to specified port)
      - `DatagramSocket(int port)`
    - Socket for server (binds to specified port, and IP)
      - `DatagramSocket(int port, InetAddress laddr)`
- Error / exception
  - `SocketException`

# Datagram Socket Methods

- To send the datagram packet
  - **`send`**`(DatagramPacket p)`
- To receive a datagram packet
  - **`receive`**`(DatagramPacket p)`
- Timeout (when waiting to receive)
  - **`setSoTimeout`**`(int timeout)`
    - On timeout, throws exception
      - `SocketTimeoutException`
- Closing the socket connection
  - `close()`

# Datagram Socket Errors

- Exceptions thrown on errors
  - `IOException,`
  - `PortUnreachableException,`
  - `SocketTimeoutException`
  - ?

# UDP Client Template : Initialization

- `java udpClient 10.26.30.11 2345`
  - **arg[0]** :`10.26.30.11` **(Server IP Address)**
  - **arg[1]**: `2345` **(Server port number)**
- **Code snippet : initialization**

```
int port;
InetAddress server;
port = Integer.parseInt(arg[1]);
server =
InetAddress.getByName(arg[0]);
DatagramSocket sock = new
DatagramSocket();
byte[] buffer = new Byte[1000];
```

# UDP Client Template: Read Input

- Read message from input terminal

```
BufferedReader userIn =
 new BufferedReader(new
 InputStreamReader(system.in));

 System.out.println("Enter chat data:
 ");
  String sentence = userIn.readLine()
  if (sentence.equals("Exit")) {
    break;
  }
  sendbuf = sentence.getBytes();
```

# UDP Client: Communication

- Read the data from console/Terminal
  - Code for reading from terminal
- Code snippet : sending and receiving data

```
DatagramPacket request = new
DatagramPacket(sendbuf, sendbuf.length,
address, port);
socket.send(request);
:
DatagramPacket response = new
DatagramPacket(recvpkt, recvpkt.length);
socket.receive(response)
String recvdata = new
String(recvpkt.getData());
```

# Errors/Exceptions

- Any socket operation when results in error, throws one of the following exceptions

```
} catch (SocketTimeoutException ex) {
    s.o.p("Timeout error: " +
ex.getMessage());
    ex.printStackTrace();
} catch (IOException ex) {
    s.o.p("Client error: " +
ex.getMessage());
    ex.printStackTrace();
} catch (InterruptedException ex) {
    ex.printStackTrace();
}
```

# UDP Server Template

- `java udpServer 2345`
  - **arg[0]:** `2345` **(Server port number)**
- **Code snippet : initialization**

```
int port;
port = Integer.parseInt(arg[0]);
DatagramSocket sock = new
DatagramSocket(port);
byte[] buffer = new Byte[1000];

DatagramPacket request = new
DatagramPacket(buffer, buffer.length);
socket.receive(request)
```

# UDP Server: Identify Client

- Code snippet : initialization

```
InetAddress client =
request.getAddress();
int clientPort = request.getPort();
// display client IP and Port
s.o.p(client.toString()+”:” +
clientPort);
```

# UDP Server: Identify Client

- Get response from user on server console to send

```
BufferedReader userIn =
  new BufferedReader(new
  InputStreamReader(system.in));
String sentence = userIn.readLine();
sendbuf = sentence.getBytes();
response = DatagramPacket(sendbuf,
sendbuf.length, address, port);
socket.send(response);
```

# Basic UDP Server and Client

- `nc -u -l 3333`
  - Runs an UDP Server on port 3333
- `nc -u 10.26.30.11 3333`
  - Run an UDP Client and connect to a UDP server running on 10.26.33.11 on port 3333.
- Running java program

```
-javac UDPClient.java
-javac UDPServer.java
-java -classpath .  UDPClient
-java -classpath .  UDPServer
```

# Template Programs

- Template programs
- UDPClientTemplate.java
- UDPServerTemplate.java
- How to use Template programs
  - Copy them to UDPClient.java, UDPServer.java
  - Fill in the lines with ??
  - Do all the error checking and throw exceptions
    - The sample program template does not have any error checking.
  - Run the program and test the server with multiple clients.

# Program Expectation.

- The UDP server program should run for ever.
- Your UDP client and server should be able to work with nc
- Your UDP client should be able to work with your UDP Server.
- Your UDP Client should be able to work with any one's UDP server and vice versa,
- Your program should not crash with any bad user input.
- UDP Server program should be able to work with multiple clients concurrently.
- Do all required validation and check all exceptions.

# TCP Template Programs

- Example: Java client server datagram programs
  - TCPClientTemplate.java
  - TCPClient.java (Later)
    - TCPServerTemplate.java
    - TCPServer.java (Later)
      - TCP Server variations: handling concurrent clients
        - TCP server: one thread for each client
        - TCP server using select.

# Lab Program: TCP Server

- Lab program expectation: Server program
  - A TCP Server program that will communicate with many clients. It should act like a chat server.
  - Port number is taken from command line argument
    - IP is taken by default (all)
  - It receives a filename (with path) from client.
  - Server reads the file content from the file and sends it to the requesting client.
  - Server forever waits for a client request
  - The program runs for ever.
  - All method calls should be checked for errors

# Lab Program: TCP Client

- Lab program expectation: Client program
  - A TCP client program that will communicate with a TCP Server
  - The input command line parameters are
    - Server IP Address
    - Server Port number
    - List of filename(s)
  - Client sends the filename(s) to server on specified port, one at a time.
  - It should wait for the response (file content) from the server. If file doesn't exist, server sends "-1".
  - It should write received content into the file.
  - All methods/system calls to be checked for errors

# TCP Data Concepts

- Stream packet
  - Data is delivered in stream of bytes
  - Each client connection is identified by a new socket (file handle).
  - Doesn't need server or client address after the connection is made.
- Key Java classes for sending files over TCP
  - `OutputStream`
  - `InputStream`
  - `BufferedReader`
  - `Socket`
  - `ServerSocket`
  - `File`
  - `PrintWriter`

# TCP Socket

- Java class constructors
  - https://docs.oracle.com/javase/8/docs/technotes/guides/net/index.html
  - Socket for client
    - `sock = new Socket(serverIPAddress, serverPort)`
    - Socket for server (binds to specified port)
      - `ssock = new ServerSocket(serverport, backlog)`
- Error / exception
  - `SocketException`

# Stream Socket Methods

- To send the filename as TCP data

```
ostream=sock.getOutputStream();
printwriter=new
PrintWriter(ostream,true);
printwriter.println(filename);
```

- To receive data by server

```
csock = ssock.accept()
clientIP =
csock.getInetAddress().getHostAddress();
clientPort = csock.getPort();
istream = csock.getInputStream( );
sockReader = new BufferedReader(new
InputStreamReader(istream));
filename = sockReader.readLine();
```

# TCP Server : Reading from File

- Checking if file exists

```
tmpfile = new File(filename);
if (! tmpfile.exists() ) {
  pwrite.println(-1);
  csock.close();
}
```

# TCP Server : Reading from File

- Reading file content and sending to client

```
ostream = csock.getOutputStream();
PrintWriter pwrite=new
PrintWriter(ostream, true);
fileReader = new BufferedReader(new
FileReader(filename));
String str;
while((str=fileReader.readLine())!
=null){
   pwrite.println(str);
}
```

# TCP Client & Server Invocation

- `java tcpClient 10.26.30.11 2345 f1 f2…`
  - arg[0] :`10.26.30.11` (Server IP Address)
  - arg[1]: `2345` (Server port number)
  - arg[2] to arg[n]: list of files
- `java tcpServer 2345`
  - arg[0]: `2345` (Server port number)

- Use `nc` to test TCP Server
  - `nc 10.26.30.11 2345 file1.txt`
  - it should display content of file or (`-1` if file does not exist)

# Basic TCP Server and Client

- `nc -l 3333`
  - Runs an TCP Server on port `3333`
- `nc 10.26.30.11 3333`
  - Run an TCP Client and connect to a TCP server running on `10.26.33.11` on port `3333`.
- Running java program

```
javac TCPClient.java
javac TCPServer.java
java -classpath .  TCPClient
java -classpath .  TCPServer
```

# Template Programs

- Template programs
- TCPClientTemplate.java
- TCPServerTemplate.java
- How to use Template programs
  - Copy them to TCPClient.java, TCPServer.java
  - Fill in the lines with ??
  - Do all the error checking and throw exceptions
    - The sample program template does not have any error checking.
  - Run the program and test the server with multiple clients.

# Program Expectation.

- The TCP server program should run for ever.
- Your TCP client and server should be able to work with nc
- Your TCP client should be able to work with your TCP Server.
- Your TCP Client should be able to work with any one's TCP server and vice versa,
- Your program should not crash with any bad user input.
- TCP Server program should be able to work with multiple clients concurrently.
- Do all required validation and check all exceptions.

# Summary

- UDP Sockets
- TCP Sockets
- Template programs
- Actual programs