

<b>0.</b>	<b>Preamble</b>	<b>1</b>
0.1	Client Server	1
0.2	What you must know from your 1st and 2nd years	2
0.3	What this course does not cover	3
<b>1.</b>	<b>Introduction to Client-Server Computing</b>	<b>4</b>
1.1	What is Client-Server Computing	5
1.2	Examples of Client Server Systems	6
1.3	Characteristics	9
1.4	What is good client server working	10
1.4.1	Good practice	10
1.4.2	Bad practice	10
1.5	Classes of client server systems (2 tiered)	11
1.5.1	Host based processing	11
1.5.2	Server based processing	11
1.5.3	Co-operative based processing	12
1.5.4	Client based processing	12
1.6	3 Tiered systems	13
1.7	Networking layers	14
1.8	Client Server (TCP/IP Unix world)	15
1.9	Definitions	16
1.10	Why use Java	17
1.11	Against Java	17
1.13	UML Sequence diagram	18
1.14	Client code	19
1.15	Class NetObjectWriter	21
1.16	Server	22
1.17	Class NetObjectReader	24
1.18	Putting it all together	25
<b>2.</b>	<b>Java for client server computing</b>	<b>26</b>
2.1	The class java.lang.Thread	27
2.2	The class java.lang.Object	28
2.3	Using threads in a server	29
2.3.1	Implementation	30
2.4	Advantages / Disadvantages	32
2.5	Thread pool	33
2.5.1	Thread pool (An implementation)	34
2.5.2	Code	35
2.6	Simple web server (Not using a thread pool)	37
2.7	Putting it all together	42
<b>3.</b>	<b>Web Client-Server Systems</b>	<b>43</b>
3.1	CGI (Common Gateway Interface)	44
3.2	Languages used to write CGI scripts	44
3.3	CGI What happens	45
3.4	Implementation details	46
3.5	Environment variables	47
3.6	Post vs. get	48
3.7	State information	49
3.8	Cookies	49
3.9	Web based client server processing	50
3.10	Tomcat	51
3.10.1	JSP Java server pages	52
3.10.2	JSP example	53
3.10.3	Output from JSP page	53
3.10.4	Generated code for JSP page	54
3.10.5	Java server pages components (Simplified)	56
3.10.6	JSP Summary	58
3.10.7	Servlet	59
3.11	Java Script	60
3.11.1	Javascript	61
<b>4.</b>	<b>Design consideration</b>	<b>62</b>
<b>5.</b>	<b>Multicast</b>	<b>63</b>
5.1	Multicast	64
5.2	An Internet Chat Application	65

5.3	Client GUI	66
5.3.2	Set up GUI	68
5.3.3	Process command line parameters	69
5.3.4	Action when user send message	70
5.3.5	Main processing loop of client	71
5.4	Server code	72
<b>6.</b>	<b>Remote Procedure Call</b>	<b>77</b>
6.1	Remote Procedure call	78
6.2	Potential problems with RPC and client Server	80
6.3.1	RMI Example: Server	82
6.3.2	RMI Example: Client	83
6.3.3	Compiling the system	84
6.3.4	Running the system	84
<b>7.</b>	<b>Clustering</b>	<b>85</b>
7.1	Overview	86
7.2	Clustering advantages	87
7.3	Types of cluster	88
7.4	Desirable middleware properties for clusters	89
7.5	Operating system design issues	90
7.6	Clustering vs. SMP	91
7.7	Beowulf <a href="http://www.beowulf.org">www.beowulf.org</a>	92
7.8	Beowulf II	92
<b>8.</b>	<b>Types of servers</b>	<b>93</b>
8.1	Connectionless vs. Connection-Oriented	94
8.2	Iterative vs. Concurrent	95
8.3	Stateless vs. Stateful	96
8.4	Classification grid	97
8.5	Server management (Linux)	98

## 0. Preamble

### 0.1 Client Server

#### ● Content

- Overview of client server computing  
TCP/IP, Types of client and server, protocols
- Java for client server computing  
Sockets, Datagram, Multicast, Threads, RMI
- O/S concepts  
Clustering, servers, control of servers
- Java related technologies  
JSP, Servlets

#### ● Assessment

##### 100% Course work

- Modify 'very simple web server written in Java (<250 lines)' to allow dynamic web page content.
- Implement in Java a game to demonstrate aspects of client server computing.
- Viva on coursework.

#### ● Web-site

`http://www.it.brighton.ac.uk/staff/mas  
-> courses -> CI346`

## **0.2 What you must know from your 1st and 2nd years**

- How to construct applications in Java (CI201/CI228)
- OS concepts (CI210/CI229)
- Simple web page construction
- Basics of Database use
- Basics of the UML design notation

### **0.3 What this course does not cover**

- How to create web pages
- How to program in Java
- Interaction useability issues
- Issues in graphic design

## **1. Introduction to Client-Server Computing**

- What is Client-Server Computing
- Overview of Unix style Networking
- Networking components:  
Sockets, ports, IP Address
- 'Simple' Java example

## 1.1 What is Client-Server Computing

### Client-Server

<programming>

A common form of distributed system in which software is split between server tasks and client tasks. A client sends requests to a server, according to some protocol, asking for information or action, and the server responds.

This is analogous to a customer (client) who sends an order (request) on an order form to a supplier (server) who despatches the goods and an invoice (response). The order form and invoice are part of the "protocol" used to communicate in this case.

There may be either one centralised server or several distributed ones. This model allows clients and servers to be placed independently on nodes in a network, possibly on different hardware and operating systems appropriate to their function, e.g. fast server/cheap client.

Examples are the name-server/name-resolver relationship in DNS, the file-server/file-client relationship in NFS and the screen server/client application split in the X Window System.

Usenet newsgroup: comp.client-server.

["The Essential Client/Server Survival Guide", 2nd edition, 1996].

## 1.2 Examples of Client Server Systems

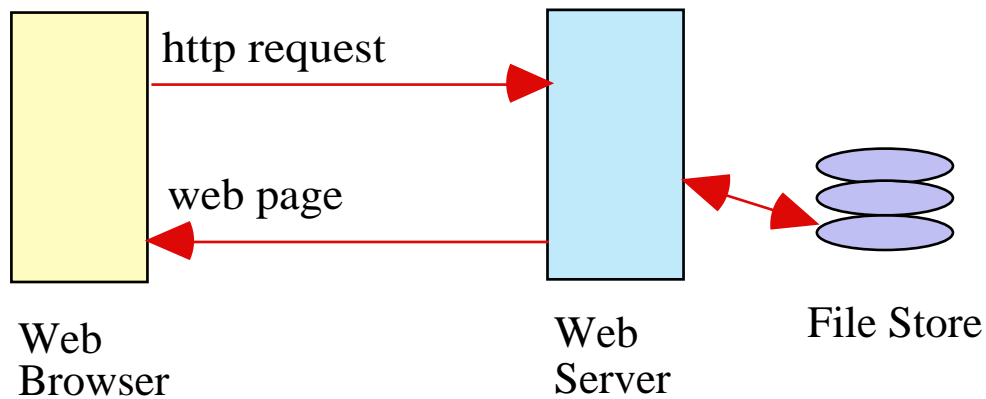
- Web

Web server - Web browser

Web server - Backend application

Web based applications

Web browser accessing ASP, JSP based server to give dynamic web content

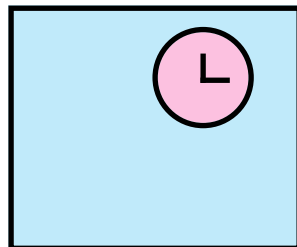


- X windows

X is the client  
Application is the server

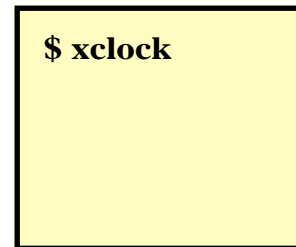
Creating a window on another machine

**Server**  
M/C running X-windows



```
$xhost client  
[2]
```

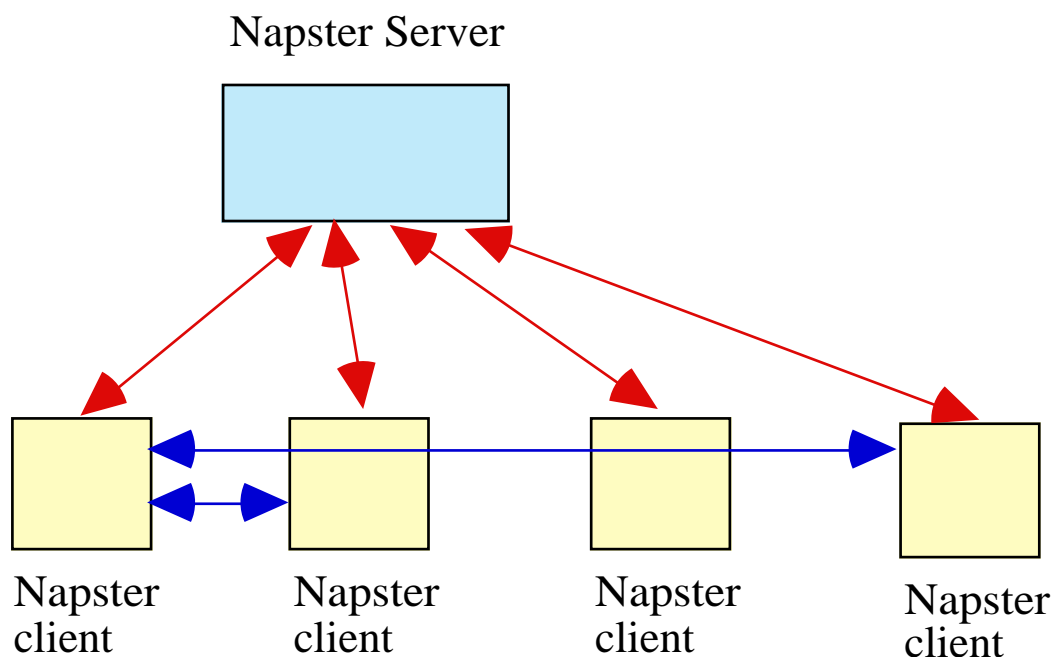
**Client**  
M/C running application



```
$export DISPLAY="server:0"  
[3]
```

- [1] The terminology client server is reversed
- [2] Grants 'client' permission to use x-windowing system
- [3] Which 'server' M/C to send output to.

- ftp  
ftp server - ftp client
- VNC  
Virtual Network Computing  
Allows a user to view the desktop of another machine (Server) using a viewer (Client) on any machine on the internet
- Napster (Now no longer free)  
Introduced peer-peer file sharing to the masses. Central database allowed clients to directly transfer files from other co-operating clients



- ICQ

### 1.3 Characteristics

- User friendly software brought to the end user of the system
- Emphasis on centralised functions  
Database, many network management and utility functions.  
Relieves individual departments or divisions of much of the overhead of maintenance of the system.
- Commitment by organisations and vendors to open systems (Open systems means many things to different organisations and companies)

Open systems: IEEE Definition

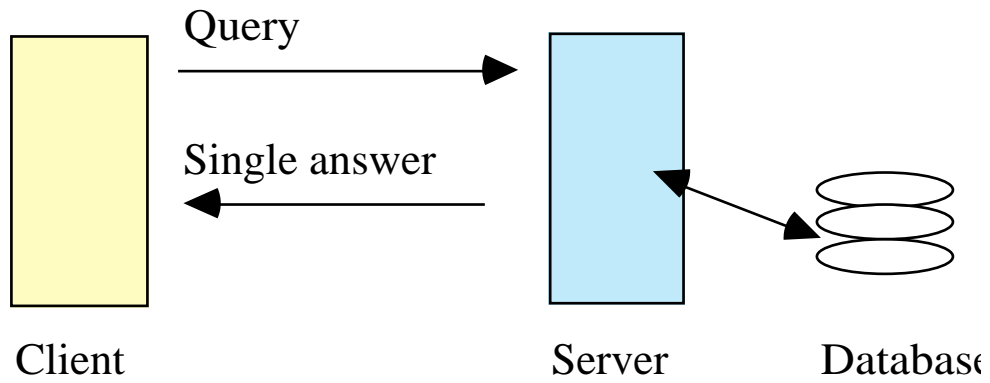
A comprehensive and consistent set of international information technology standards and functional standards profiles that specify interfaces, services and supporting formats to accomplish interoperability and portability of applications, data and people.

- Network is fundamental to the operation

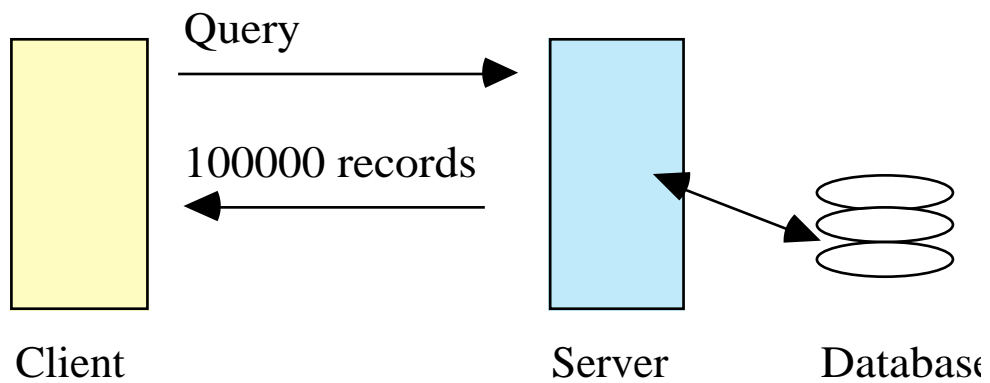
Stallings Operating systems 4th edition

## 1.4 What is good client server working

### 1.4.1 Good practice

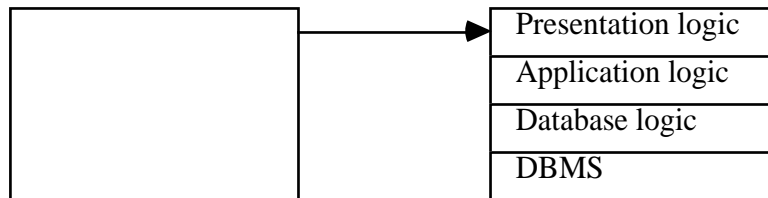


### 1.4.2 Bad practice



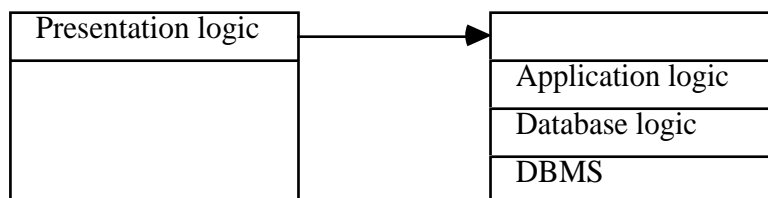
## 1.5 Classes of client server systems (2 tiered)

### 1.5.1 Host based processing



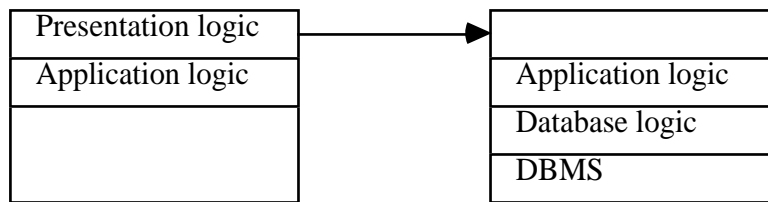
- Dumb terminal on client used to run application
- VNC (Complete desktop of remote computer on users local computer)

### 1.5.2 Server based processing



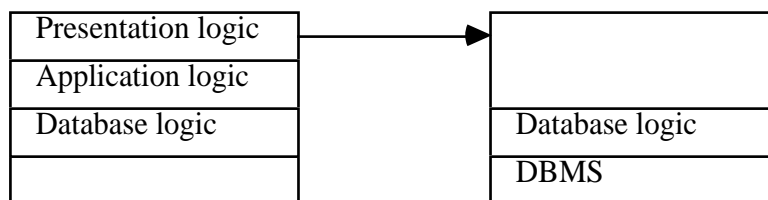
- Client only provides GUI, processing done in server
- Thin client
- webmin (Administer a computer from a remote machine)

### 1.5.3 Co-operative based processing



- Takes advantage of strengths of client and server

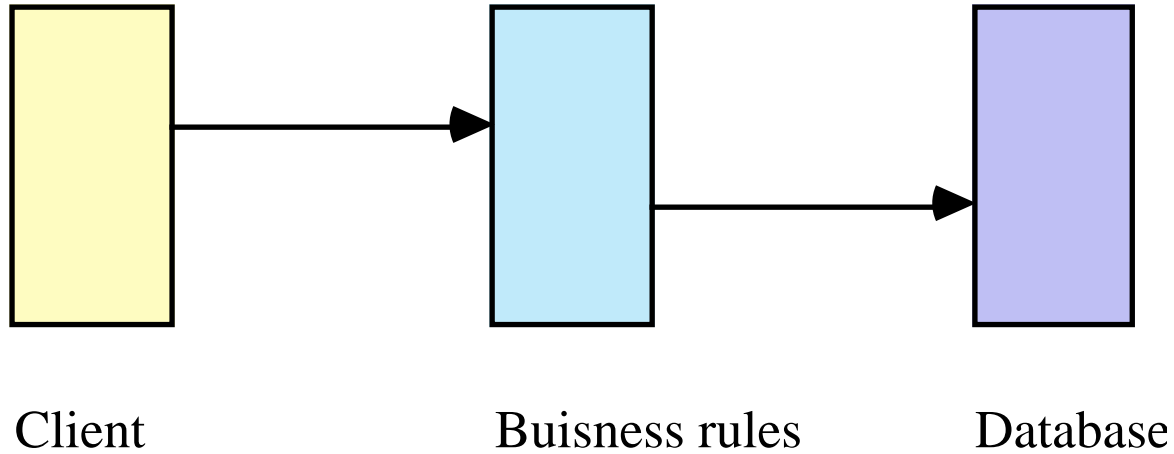
### 1.5.4 Client based processing



- Most processing done on client, thick client

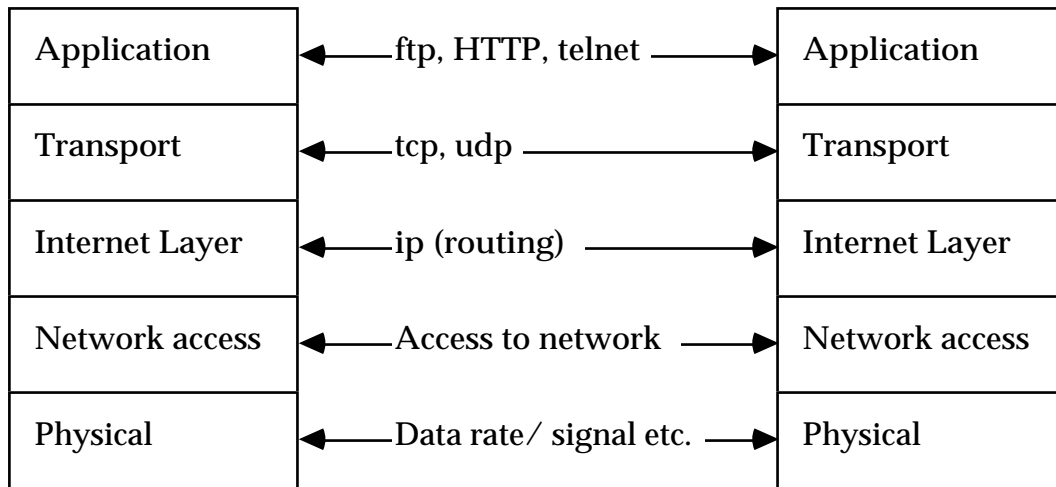
Stallings Operating systems 4th edition

## 1.6 3 Tiered systems



- Client  
Thin client
- Middle tier  
Hides the databases from the client, Implements the rules/ ways of working of the organisation. May provide an interface to legacy systems
- Database  
Backend server, Data of the organisation

## 1.7 Networking layers

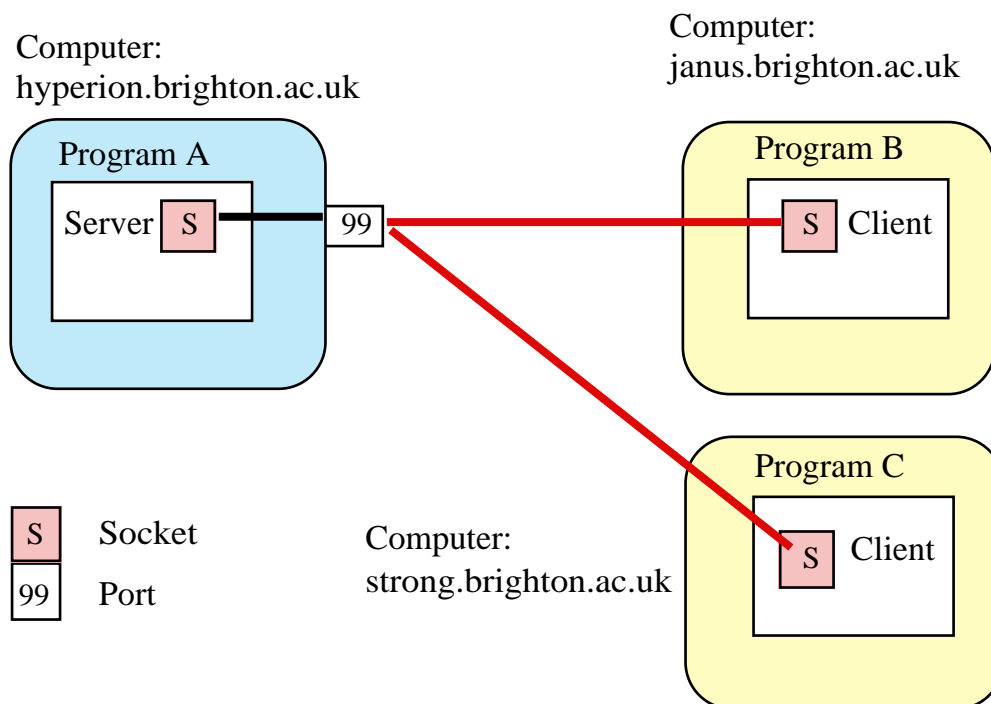


- Transport
  - TCP  
Reliable deliver of data  
point to point connection
  - UDP (User Datagram Protocol)  
Deliver of data is not guaranteed, nor the order in which data is received
- Internet Layer  
Concerned with routing data across the network
- Network access  
How host communicates with network, ethernet
- Physical  
Physical interface between data transmission device

## 1.8 Client Server (TCP/IP Unix world)



<b>Client</b>	A client application sends transactions to a server application or applet for processing.
<b>Server</b>	A server application processes transaction on behalf of one or more clients.



## 1.9 Definitions

<b>IP Address</b>	<p>Currently an IPv4 (IP Version 4) uses a 32 bit integer value, for the address usually written as 4 bytes in the form: 193.62.183.86.</p> <p>Note: The format for an IP address will soon be expanded to IPv6 (IP Version 6 this will allow for 128 bit addresses) to allow for an even greater number of machines to be connected to the internet.</p>								
<b>domain name</b>	<p>A symbolic name for a machine or group of machines. Each domain owner can register other names in that domain. For example, the University of Brighton [brighton.ac.uk] can choose the names of machines or other sub domains in its domain. Each domain will usually have a DNS (Domain Name Server) that will map the name of a machine within its domain to the true IP address. A DNS is a computer that is able to provide an IP address for a domain name or be able to ask another DNS to provide this information.</p>								
<b>TCP</b>	<p>Transport Control Protocol.</p> <p>A protocol that guarantees that when a connection is established and kept open, the receiving machine will get the sent messages.</p> <p>The message will be split into packets, that may be sent by any route to the destination, however the TCP protocol guarantees to re-assemble these individual packets into the correct order for delivery to the receiver.</p>								
<b>UDP</b>	<p>User Datagram Protocol.</p> <p>Messages are split into packets and sent to the destination but there is no guarantee that the packets will arrive in the correct order or even arrive at all.</p>								
<b>IP</b>	<p>Internet Protocol.</p> <p>A collection of protocols of which TCP and UDP are members. Hence TCP/IP.</p>								
<b>port number</b>	<p>A number in the range 1—65536, examples of used port numbers include:</p> <table border="0"> <tr> <td><b>Port</b></td> <td><b>Used by</b></td> </tr> <tr> <td>20-21</td> <td>File transfer programs using FTP</td> </tr> <tr> <td>23</td> <td>Terminal emulators (Telnet)</td> </tr> <tr> <td>79</td> <td>Finger programs</td> </tr> </table>	<b>Port</b>	<b>Used by</b>	20-21	File transfer programs using FTP	23	Terminal emulators (Telnet)	79	Finger programs
<b>Port</b>	<b>Used by</b>								
20-21	File transfer programs using FTP								
23	Terminal emulators (Telnet)								
79	Finger programs								
<b>Host</b>	<p>The machine on which the program is running.</p>								

## **1.10 Why use Java**

- Core system embeds enabling technology for client server systems.

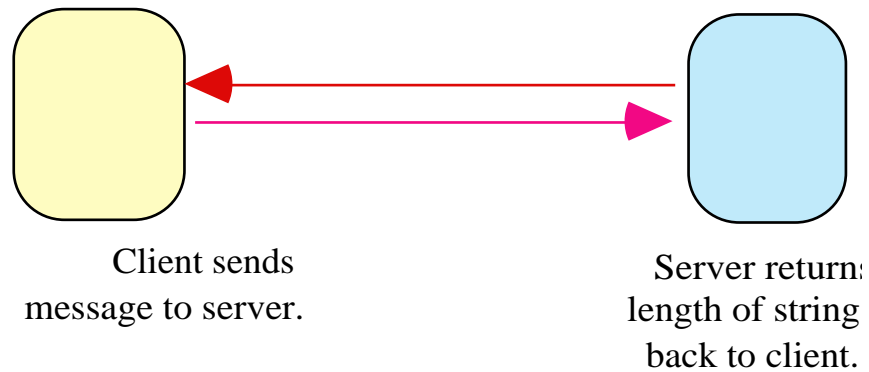
Java language contains/ Implements

- Networking class library
  - Serialization of standard objects
  - Remote Procedure Calls (RMI, CORBA)
  - Threads
- 
- Java uses many open systems
    - TCP/IP
  - Java is portable across many systems
    - Windows, Sun, Apple, Linux
  - Cost of ownership
    - Compiler/ Interpreter freely available

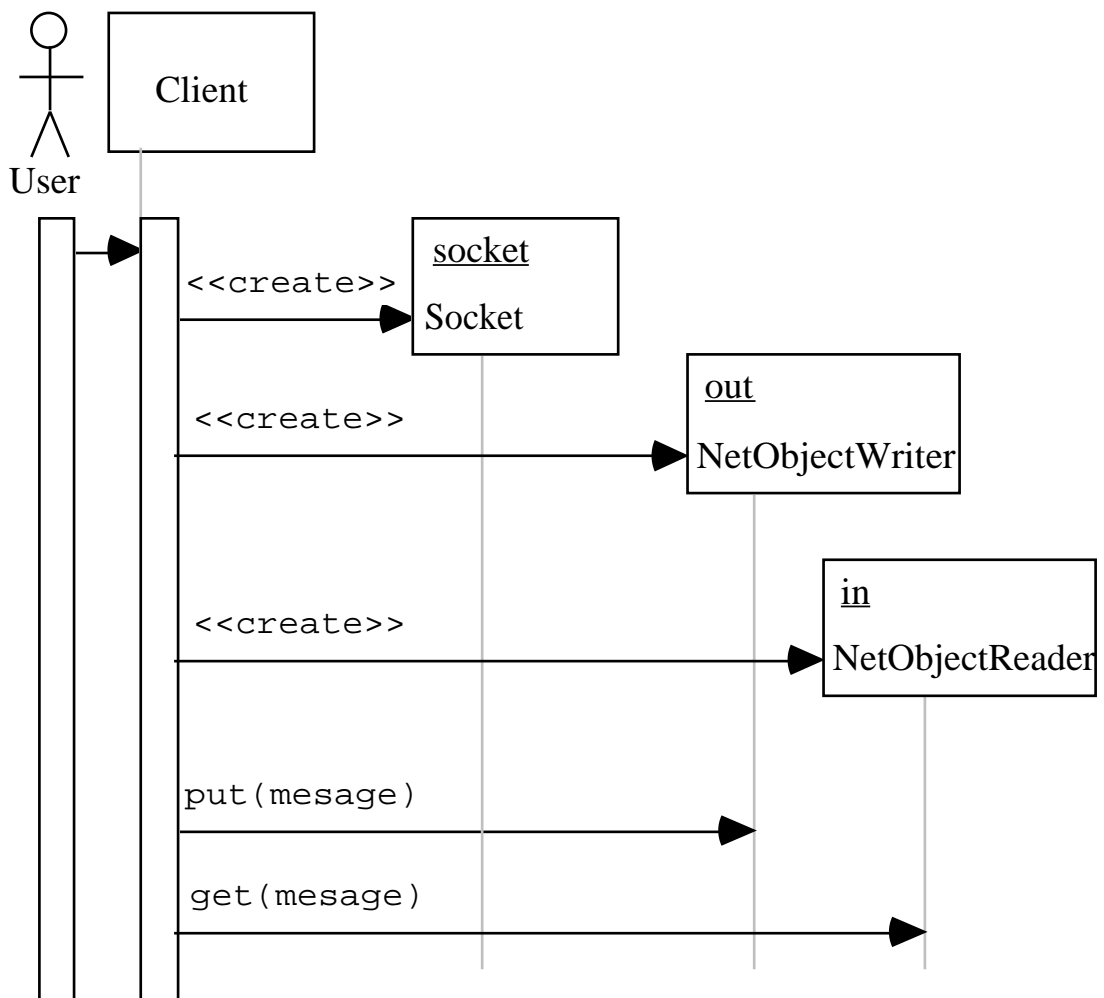
## **1.11 Against Java**

- Interpreted.
- Not Microsoft

## 1.12 Client Server (TCP/IP Java view)



## 1.13 UML Sequence diagram



## 1.14 Client code

Set host name to connect too & port to use for connection

```
import java.net.*;
import java.util.*;

class Client
{
    public static void main( String args[] )
    {
        String host      = "localhost";           //Host name
        int    port      = 2000;                 //Port used
        try
        {
            if ( args.length >= 3 )
            {
                host      = args[0];             //machine
                port      = Integer.parseInt( args[1] ); //Port
            } else {
                throw new Exception("Bad usage");
            }
        }
        catch ( Exception err )
        {
            System.out.println( "Usage Client machine port message(s)" );
            return;
        }
    }
}
```

```
try
{
    NetObjectWriter out;           //Object output stream
    NetObjectReader in;           //Object input stream

    Socket socket = new Socket( host, port ); //Socket
    out = new NetObjectWriter( socket );     //Output
    in = new NetObjectReader( socket );     //Input
    for (int i= 2; i<args.length; i++)      //Send messages
    {
        out.put( args[i] );                // to Server
        String response = (String) in.get(); // Response
        System.out.println("Length of [" +
                            args[i] + "] is " +
                            response );
    }
    out.close();                          //Close connection
}
catch ( Exception err )
{
    System.out.println("Error: " + err.getMessage() );
}
}
```

## 1.15 Class NetObjectWriter

```
class NetObjectWriter extends ObjectOutputStream
{
    public NetObjectWriter( Socket s ) throws IOException
    {
        super( s.getOutputStream() );
    }

    public synchronized boolean put( Object data )
    {
        try
        {
            writeObject( data );           //Write object
            flush();                         //Flush
            return true;                    //Ok
        }
        catch ( IOException err )
        {
            return false;                  //Failed write
        }
    }
}
```

## 1.16 Server

```
import java.lang.*;
import java.net.*;

class Server
{
    public static void main( String args[] )
    {
        try
        {
            if ( args.length == 1 )
            {
                int port = Integer.parseInt( args[0] ); //Port number
                (new Server()).process( port );         //Start server
                return;                                 //Normal exit
            }
        }
        catch ( Exception err )
        {
            System.out.println( "port not integer" ); //Integer port no
        }
        System.out.println( "Usage Server port" );   //Display usage
    }
}
```

```
public void process( final int port )
{
    try
    {
        ServerSocket ss = new ServerSocket(port); //Server Socket
        while( true ) //Loop
        {
            Socket socket = ss.accept(); // Wait
            NetObjectReader in =
                new NetObjectReader(socket); //Input
            NetObjectWriter out=
                new NetObjectWriter(socket); //Output

            while ( true ) //Loop
            {
                Object obj = in.get(); //From Client
                if ( obj == null ) break; //No more data
                String message = (String) obj; //To string
                System.out.println( message ); //Print message
                out.put( "" + message.length() ); //Return length
            }

            in.close(); //Close Read
            out.close(); //Close Write
            socket.close(); //Close Socket
        }
    }

    catch ( Exception err )
    {
        System.out.println("Error: " + err.getMessage() );
    }
}
```

## 1.17 Class NetObjectReader

```
class NetObjectReader extends ObjectInputStream
{
    public NetObjectReader( Socket s ) throws IOException
    {
        super( s.getInputStream() );
    }

    public synchronized Object get()           //Get object from stream
    {
        try                                     //
        {
            return readObject();               //Return read object
        }
        catch ( Exception err )                //Reading error
        {
            return null;                        // On error return null
        }
    }
}
```

## 1.18 Putting it all together

When compiled and run the following results are produced:

<b>Client (Application)</b>	
<b>Command line used</b>	<code>java Client localhost 2000 "A message"</code>
<b>Results returned</b>	<code>Length of [A message] is 9</code>

<b>Server (Application)</b>	
<b>Command line used</b>	<code>java Server 2000</code>
<b>Results returned</b>	<code>A message</code>

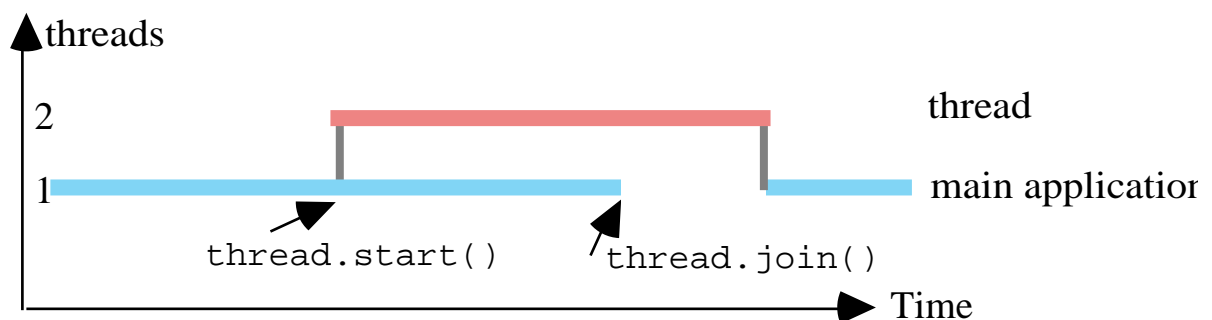
*Note: Now there may be many client sending a message to the server.  
The Server will run forever unless terminated*

## **2. Java for client server computing**

- Threads
- Using threads in a server
- Thread pool

## 2.1 The class `java.lang.Thread`

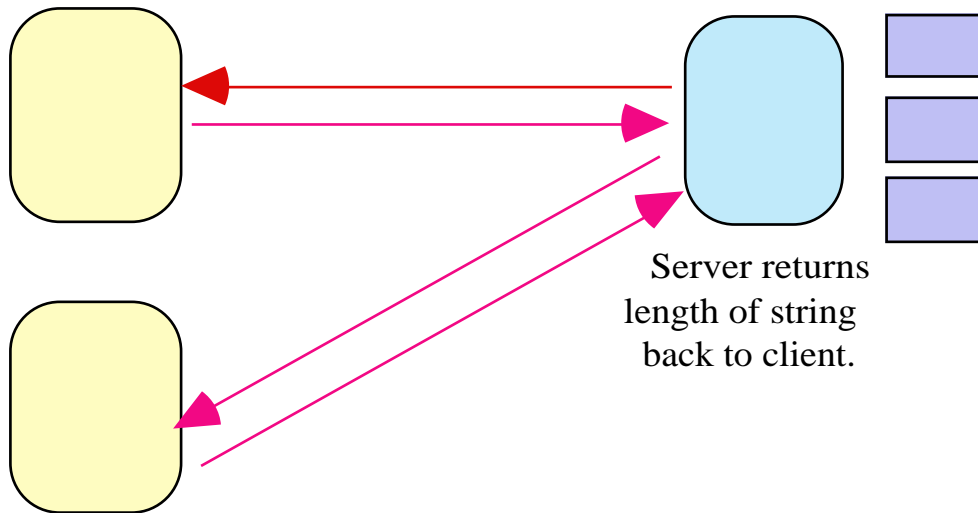
Method	Responsibility
<code>Thread()</code> <code>Thread( name )</code> <code>Thread( fo )</code> <code>Thread( fo, name )</code>	Creates a new thread with name <code>name</code> . Creates a new thread that will run the function object <code>fo</code> that implements the interface <code>Runnable</code> .
<code>activeCount()</code>	Returns the number of active threads in this thread group.
<code>getName()</code>	Returns the name of the thread.
<code>isAlive()</code>	Returns true if the thread is alive.
<code>join()</code>	Waits for the thread to die.
<code>join( delay )</code>	Waits at most <code>delay</code> milliseconds for the thread to die.
<code>run()</code>	Runs the thread's function object that implements the interface <code>Runnable</code> .
<code>sleep( delay )</code>	Causes the thread to sleep for <code>delay</code> milliseconds.
<code>start()</code>	Causes the thread to start, the <code>run</code> method in the thread is called.
<code>yield()</code>	After pausing the thread allow other threads to continue.



## 2.2 The class `java.lang.Object`

Method	Responsibility
<code>notifyAll()</code>	Wakes up all threads that are waiting on this object. A thread enters the wait state when it calls one of the <code>wait</code> methods.
<code>notify()</code>	Wakes up a single thread that is waiting on this object.
<code>wait( delay )</code>	Waits until either of the following two conditions occur: <ul style="list-style-type: none"><li>• One of the methods <code>notify</code> or <code>notifyall</code> is called from another thread on this object.</li><li>• The time <code>delay</code> in milliseconds has passed.</li></ul>
<code>wait()</code>	Waits until one of the methods <code>notify</code> or <code>notifyAll</code> is called from another thread on this object.

## 2.3 Using threads in a server



Clients sends message to server.

- System
  - Multiple clients sends server a text message
  - Server replies with the length of the message
- Strategy

Create thread on demand to process the transaction.  
Thread terminates at end of transaction.

## 2.3.1 Implementation

```
class Server
{
    public static void main( String args[] )
    {
        try
        {
            if ( args.length == 1 )
            {
                int port = Integer.parseInt( args[0] ); //Port number
                (new Server()).process( port );          //Start server
                return;                                   //Normal exit
            }
        }
        catch ( Exception err )
        {
            System.out.println( "port not integer" ); //Integer port no
        }
        System.out.println( "Usage Server port" ); //Display usage
    }

    public void process( final int port )
    {
        try
        {
            ServerSocket ss = new ServerSocket(port); //Server Socket
            while( true )                               //Loop
            {
                ThreadToProcess thread = null;
                Socket socket = ss.accept();           // Wait connection
                thread = new ThreadToProcess(socket); // Create thread
                thread.start();                         // Start thread
            }
        }
        catch ( Exception err )
        {
            System.out.println("Error: " + err.getMessage() );
        }
    }
}
```

```
class ThreadToProcess extends Thread
{
    private Socket          theSocket;           //Socket used
    private NetObjectReader theIn;             //Input stream
    private NetObjectWriter theOut;           //Output stream

    public ThreadToProcess( Socket s )         //Construct
    {
        theSocket = s;                         //Remember socket
    }

    public void run()                          //Execution
    {
        try
        {
            theIn = new NetObjectReader( theSocket ); //Input
            theOut = new NetObjectWriter( theSocket ); //Output

            while ( true )                      //Loop
            {
                Object obj = theIn.get();       //From Client
                if ( obj == null ) break;       //No more data
                String message = (String) obj;  //To string
                System.out.println( message );  //Print message
                theOut.put( "" + message.length() ); //Return length
            }

            theIn.close();                      //Close Read
            theOut.close();                     //Close Write

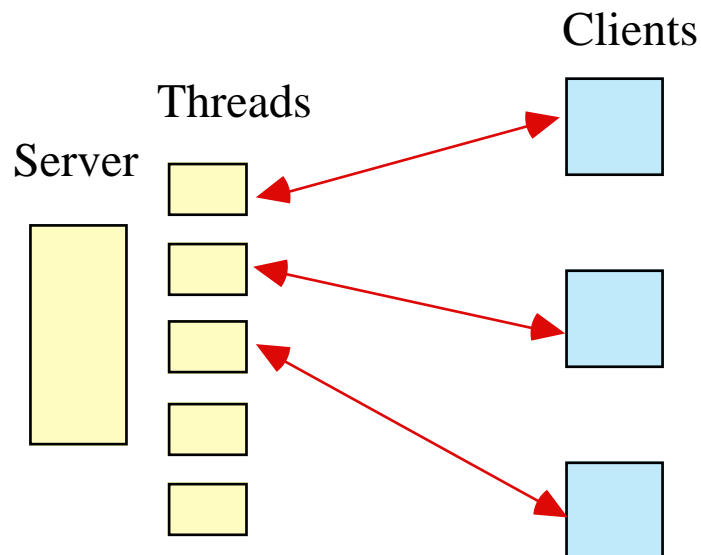
            theSocket.close();                  //Close Socket
        }
        catch ( Exception err ) {}
    }
}
```

## **2.4 Advantages / Disadvantages**

- Advantages
  - Transactions are now processed 'simultaneously', making better use of machine resources.
  - Scalability of server on an SMP machine.
  
- Disadvantages
  - No limit / control on resources used  
Many simultaneous requests could impose major resource issues on the server system
  - Overhead of thread creation for each transaction

## 2.5 Thread pool

- Advantages
  - Transactions are still processed simultaneously  
Server will scale on an SMP machine
  - Control on how many simultaneous threads are created.
  - Threads are re-used to process next transaction, so saving on thread creation overhead.
- Disadvantages
  - Inactive threads in the thread pool still take up resources.



### **2.5.1 Thread pool (An implementation)**

- **Server**  
Simply replies with the length of the text message sent to it.
- **Strategy**  
Create pool of threads (10) A free thread will process transaction.

If all threads used then will have to wait until a thread is free for the transaction to be processed.

- **Variation**  
Could implement the queue of requests waiting for free thread rather than let all free threads compete for next request.

## 2.5.2 Code

```

import java.lang.*;
import java.net.*;

class Server
{
    public static void main( String args[] )
    {
        try
        {
            if ( args.length == 1 )
            {
                int port = Integer.parseInt( args[0] ); //Port number
                (new Server()).process( port );         //Start server
                return;                                 //Normal exit
            }
        }
        catch ( Exception err )
        {
            System.out.println( "port not integer" ); //Integer port no
        }
        System.out.println( "Usage Server port" );   //Display usage
    }

    public void process( final int port )
    {
        try
        {
            ServerSocket ss = new ServerSocket(port); //Server Socket
            for( int i=1; i<=10; i++ )              //
            {
                thread = new ThreadInPool( ss );    // Create thread
                thread.start();                      // in thread pool
            }
        }
        catch ( Exception err )
        {
            System.out.println("Error: " + err.getMessage() );
        }
    }
}

```

```

/*
 * Implemented as a Thread pool
 */

class ThreadInPool extends Thread
{
    private ServerSocket    theServerSocket;    //ServerSocket used
    private NetObjectReader theIn;              //Input stream
    private NetObjectWriter theOut;             //Output stream
    private static int      theUniqueNo=1;
    private String          theName;            //Name of Thread

    public ThreadInPool( ServerSocket ss )      //Construct
    {
        theServerSocket = ss;                  //Remember socket
        theName = "Thread " + theUniqueNo++;    //Name
    }

    public void run()                           //Execution
    {
        while ( true )
        {
            try
            {
                Socket socket = theServerSocket.accept();//Connection
                theIn = new NetObjectReader( socket );//Input
                theOut = new NetObjectWriter( socket );//Output

                while ( true )                   //Loop
                {
                    Object obj = theIn.get();    //From Client
                    if ( obj == null ) break;    //No more data
                    String message = (String) obj; //To string
                    System.out.println(theName + message);
                    theOut.put( "" + message.length() );//Return length
                }

                theIn.close();                   //Close Read
                theOut.close();                  //Close Write

                socket.close();                  //Close Socket
            }
            catch ( Exception err ) {}          //Abandon
        }
    }
}

```

## 2.6 Simple web server (Not using a thread pool)

```

class Server
{
    ThreadForTrans theThread;

    public static void main( String args[] )
    {
        try
        {
            IO.Trace("STATE", "Web server 0.3" );
            if ( args.length == 2 )
            {
                int port = Integer.parseInt( args[0] );//Port number
                IO.Trace("STATE", "Base of file system : " + args[1] );
                (new Server()).process( port, args[1] );//Start server
                return; //Normal exit
            }
        }
        catch ( Exception err )
        {
            IO.Error( "port not integer" ); //Integer port no
        }
        IO.Error( "Usage port FileBase" ); //Display usage
    }

    public void process( final int port, final String base )
    {
        try
        {
            ServerSocket ss = new ServerSocket(port); //Server Socket
            while( true ) //Loop
            {
                Socket socket = ss.accept(); // Wait for connection
                theThread = new ThreadForTrans(socket,base);// Create thread
                //Either run as separate thread or serially
                theThread.start(); // Start thread
                //theThread.run(); // No Thread
            }
        }
        catch ( Exception err )
        {
            IO.Error("process: " + err.getMessage() );
        }
    }
}

```

```

class ThreadForTrans extends Thread
{
    private Socket theSocket;           //Socket used
    private String baseDir = "/";
    private NetInputStream theIn;      //Input object stream
    private NetOutputStream theOut;

    public ThreadForTrans( Socket s, String base )
    {
        theSocket = s;                 //Socket used
        baseDir = base;                //Base of files
    }

    public void run()                  //Execution
    {
        IO.Trace( "STATE", "Open connection " );
        try
        {
            theIn = new NetInputStream( theSocket ); //Input
            theOut = new NetOutputStream( theSocket ); //Output
            String get = null;

            while ( true )              //Loop
            {
                String message = theIn.getLine(); //From Client
                if ( message == null ) break; //No more data
                IO.Trace( "REC", message );

                if ( startsWith( message, "GET " ) )
                    get = message.substring( 4 ); //Remember file name
                if ( startsWith( message, "Host" ) )
                {
                    sendFile( get ); //send file
                    get = null; //Reset
                }
            }
            IO.Trace( "STATE", "Close connection " );
            theIn.close(); //Close Read
            theOut.close(); //Close Write
            theSocket.close(); //Close Socket
        }
        catch( Exception err )
        {
            IO.Error( "ThreadForTrans: " + err.getMessage() );
        }
        IO.Trace( "STATE", "Close conection" );
    }
}

```

```
//Check if message starts with string
boolean startsWith( String message, String start )
{
    return (message.toLowerCase()).startsWith(start.toLowerCase());
}

//Send file to Client
//Extension used to define type of transfer
public void sendFile( String m ) throws IOException
{
    if ( m == null ) return;

    String file = baseDir + m.substring( 0, m.indexOf( ' ' ) );
    String ext  = file.substring( file.indexOf( '.' )+1 );
    ext = ext.toLowerCase();

    DEBUG.Trace( "FILE", "File [" + file + "]" +
                 " Ext [" + ext  + "]" );

    File aFile = new File( file );
    if ( aFile.canRead() )
    {
        theOut.putLine( "HTTP/1.1 200 OK " );
        theOut.putLine( "Connection: close" );
        theOut.putLine( "Content-length " + aFile.length() );
        if ( ext.equals( "jpg" ) || ext.equals( "jpeg" ) )
        {
            theOut.putLine( "Content-type: image/jpeg" );
        }

        if ( ext.equals( "gif" ) )
        {
            theOut.putLine( "Content-type: image/gif" );
        }

        if ( ext.equals( "htm" ) || ext.equals( "html" ) )
        {
            theOut.putLine( "Content-type: text/html " );
        }
        theOut.putLine( "" );
        theOut.copyFile( file );
    } else {
        theOut.putLine( "HTTP/1.1 403 Problem " );
    }
}
}
```

```
import java.io.*;
import java.net.*;

class NetInputStream extends BufferedInputStream
{
    private String theMessage = null; //

    public NetInputStream( Socket s ) throws IOException
    {
        super( s.getInputStream() );
    }

    public String getLine()
    {
        try
        {
            String res = ""; //Accumulating line
            int c; //Character read
            while ( true )
            {
                c = read(); //Next character
                if ( c == '\n' ) return res; //End of line
                if ( c == -1 ) return null; //End of file (EOF)
                if ( c != '\r' ) //Ignore ''
                {
                    res += (char) c; // Append
                }
            }
        }
        catch ( Throwable e ) { } //I/O error

        return null; //Treat as EOF
    }
}
```

```
class NetOutputStream extends BufferedOutputStream
{
    private String theMessage = null; //

    public NetOutputStream( Socket s ) throws IOException
    {
        super( s.getOutputStream() );
    }

    public void write( byte[] buf, int offset, int len )
        throws IOException
    {
        super.write( buf, offset, len );
        super.flush();
    }

    public void putLine( String message ) throws IOException
    {
        byte[] asBytes = (message + "").getBytes();
        write( asBytes, 0, asBytes.length ); //Send Message
        IO.Trace( "SENT", message );
    }

    public void copyFile( String fileName ) throws IOException
    {
        final int bufferSize = 4096;
        byte buffer[] = new byte[bufferSize];

        FileInputStream istream = new FileInputStream(fileName);
        BufferedInputStream theBis = new BufferedInputStream(istream);

        while ( true )
        {
            int len = theBis.read( buffer, 0, bufferSize );
            if ( len < 0 ) break;
            write( buffer, 0, len );
        }
        theBis.close();
        istream.close();
        flush();
        putLine( " " );
        IO.Trace( "FILE", "Sent " + fileName );
    }
}
```

```

class IO
{
    public static void Trace( String state, String mes )
    {
        System.err.println( (state + " ").substring(0, 5) +
                            ": " + mes );
    }

    public static void Error( String s )
    {
        System.err.println( "Error : " + s );
    }
}

```

## 2.7 Putting it all together

When compiled and run the following results are produced:

	<b>Server (Application)</b>
<b>Command line used</b>	java Server 4444 h:\\html\\

*Note:*      4444                      Port used  
                 h:\\html\\              base of file system for web pages

	<b>Client (Web browser)</b>
<b>URL</b>	http://localhost:4444/home.html

*Note:*      Web page accessed is at  
                 h:/html/home.html

### **3. Web Client-Server Systems**

- CGI (Common Gateway Interface)
- Languages used with CGI
- State information
- Tomcat

### 3.1 CGI (Common Gateway Interface)

A CGI script is a program that is stored on the remote web server and executed on the web server in response to a request from a user. For example:

```
<A HREF="http://host/cgi-bin/cgi_program">run cgi</A>
```

CGI programs are held in a special directory usually `cgi-bin` on the web server.

A CGI script file is written in a programming language which can be either:

- Compiled to run on the server.
- Interpreted by an interpreter on the server.

### 3.2 Languages used to write CGI scripts

- Compiled languages: C, C++, Ada, Java (Into bytecode)
- Interpreted languages: perl, shell script  
Usual to use an interpreted language as this can facilitate fast development of the system

### 3.3 CGI What happens

The CGI script is executed when an anchor tag `<A ... >` or an image tag `<IMG ...>` refers to the CGI script file rather than a normal file. The determination of whether this is a CGI script file or just an HTML file is made on the physical placement of the file on the server. Remember, the script file is placed on the same machine on which the web server runs and not on your local machine.

Usually this placement is in the remote web servers `cgi-bin` directory. However the exact location of this directory on the server machine is determined by the web administrator for that machine. This placement and control of the `cgi-bin` directory is determined by the web administrator to prevent security problems, that could occur if arbitrary programs were allowed to be executed by anybody accessing the machine. Some web administrators may not allow you to create CGI scripts on their machine.

### 3.4 Implementation details

The script eventually returns an HTML page or image to be displayed as the result of its execution. When a CGI script file executes it may access environment variables to discover additional information about the process that it is to perform. The first line of the returned data **must** be:

Type of returned data	Text
An HTML page	Content-type: text/html
A gif image	Content-type: image/gif

A simple CGI script on a unix based web server to return a list of the current users who are logged onto the web server is:

<pre>#!/bin/sh echo Content-type: text/html echo echo echo "&lt;HTML&gt;" echo "&lt;HEAD&gt;" echo "&lt;/HEAD&gt;" echo "&lt;BODY&gt;" echo "&lt;H2&gt;Users logged in are:&lt;/H2&gt;" echo "&lt;PRE&gt;" who echo "&lt;/PRE&gt;" echo "&lt;/BODY&gt;" echo "&lt;/HTML&gt;"</pre>	<p><b>Remember:</b></p> <ul style="list-style-type: none"> <li>• The ""s around text with a &lt; or &gt; character.</li> </ul> <p><b>On a Unix system:</b></p> <ul style="list-style-type: none"> <li>• The first line is #!/bin/sh</li> <li>• The file has executable permission set.</li> </ul>
--	---

### 3.5 Environment variables

The major environment variables that can be accessed by the CGI script when it executes are:

<b>Environment variable</b>	<b>Contains</b>
QUERY_STRING	Data sent to the CGI script, by its caller. This may be the output from a form, or other dynamically or statically generated data.
REMOTE_ADDR	The Internet address of the host machine making the request.

When a form is used, the information collected in the form is sent to the CGI script for processing. This information is placed in the environment variable [QUERY\\_STRING](#).

To pass information explicitly to the environment variable [QUERY\\_STRING](#) a modified form of an anchor tag is used. In this modified anchor tag, the data to be sent to the environment variable [QUERY\\_STRING](#) is appended after the URL which denotes the CGI script. The character ? is used to separate the URL denoting the CGI script and the data that is to be sent to the script. For example:

```
<A HREF="/cgi-bin/script?name=Your+name&action=find">
Link </A>
```

The data "name=Your+name&action=find" is placed in the environment variable [QUERY\\_STRING](#) and the cgi script `script` executed.

### 3.6 Post vs. get

So far the method used to send information to the CGI script has been GET. When the method GET is used the data sent is placed in the environment variable QUERY\_STRING for the CGI script to process.

An alternative method is to use POST. When the method POST is used the data is sent by a separate stream and becomes the standard input to the CGI script. The method used is specified on the <FORM ..> tag using the attribute METHOD="get" or METHOD="post". The default method is GET.

For example:

Generated form	HTML markup required
<input type="text" value="Try it (get)"/>	<pre>&lt;FORM METHOD="get "   ACTION="http://host/cgi-bin/mas_form"&gt;   &lt;INPUT TYPE="text" NAME="name"     SIZE=20 VALUE="Try it (get)"&gt; &lt;/FORM&gt;</pre>
<input type="text" value="Try it (post)"/>	<pre>&lt;FORM METHOD="post "   ACTION="http://host/cgi-bin/mas_form"&gt;   &lt;INPUT TYPE="text" NAME="name"     SIZE=20 VALUE="Try it (post)"&gt; &lt;/FORM&gt;</pre>

When using the POST attribute, the following environment variables are set:

Environment variable	Contains
CONTENT_LENGTH	The length of the data sent via the standard input to the CGI program.
CONTENT_TYPE	The MIME type of the data.

### 3.7 State information

- Where is state information held
  - Server  
Then need a mechanism to identify who it belongs to securely.
  - Client  
Then needs to be re-transmitted to server so processing can take place.
- Security  
To be secure the data needs to be encrypted.

### 3.8 Cookies

- Information sent by server to be stored on client
- Information stored on client

Name	likes
Value	red
Host which sent cookie	localhost
Path	/
Expires	At end of session
Server secure	no
- Cookies resent when communicating with same host
- What are the implications of this
  - Used to track which sites you have visited
  - Used to give you customised web pages when next access site

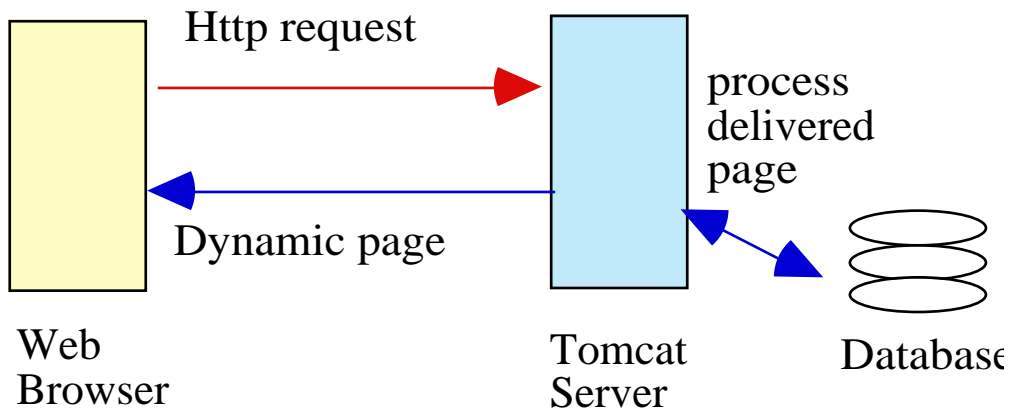
### **3.9 Web based client server processing**

- Philosophies
- Client side  
processing takes place in the web browser
  - + Off loads work to the client
  - How do you save state
- Server side  
Processing takes place on the server
  - + State can be stored on the server
  - Server now has to do work
  - Delay due to communication latency and workload on server
- Server side & client side  
Processing takes place on the server and on the client
  - + Best of both worlds
  - Added complexity

### 3.10 Tomcat

- Tomcat

A web server that implements dynamic web page processing ( JSP enabled web pages or servlets). The tomcat server in essence provides enhanced processing to that provided by a normal web server.  
Server side processing



- Its a client server system
  - Client is the web browser  
Thin client
  - Server is the tomcat server, to which the implementor adds JSP enabled web pages or Servlets

### **3.10.1 JSP Java server pages**

Web pages with embedded script that is executed when page is requested and results in a dynamically created web page.

In reality the web page is pre-compiled into a 'Servlet', and it is the pre-compiled program 'Servlet' that is run when the JSP page is accessed on subsequent occasions.

'Web page' on server may contain:

- Interpolation of values generated by standard Java expressions
- Java variable declarations  
Which can be used later in the web page
- Java control structures  
Which may contain embedded HTML. The exact HTML generated will depend on the execution path through the Java statement

to help control the generation and content of the dynamically generated web page.

## 3.10.2 JSP example

- JSP page

```
<HTML>

<BODY>
  <OL>
    <LI> <%= 2+4 %>

    <LI> Time = <%= System.currentTimeMillis() %>

    <%@ page import="java.util.*" %>
    <% GregorianCalendar c = new GregorianCalendar();
       int year          = c.get(Calendar.YEAR);    %>
    <LI> Year = <%= year %>

    <% if ( (year) %4 == 0 ) { %>
    <LI>It's a leap year
    <% } else { %>
    <LI>It's not a leap year
    <% } %>

  </OL>
</BODY>
</HTML>
```

## 3.10.3 Output from JSP page

```
6
Time = 1042649122868
Year = 2003
It's not a leap year
```

### 3.10.4 Generated code for JSP page

```
package org.apache.jsp;

import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.jsp.*;
import org.apache.jasper.runtime.*;
import java.util.*;

public class date_jsp extends HttpJspBase {

    private static java.util.Vector _jspx_includes;

    public java.util.List getIncludes() {
        return _jspx_includes;
    }

    public void _jspService(HttpServletRequest request,
        HttpServletResponse response)
        throws java.io.IOException, ServletException {

        JspFactory _jspxFactory = null;
        javax.servlet.jsp.PageContext pageContext = null;
        HttpSession session = null;
        ServletContext application = null;
        ServletConfig config = null;
        JspWriter out = null;
        Object page = this;
        JspWriter _jspx_out = null;

        try {
            _jspxFactory = JspFactory.getDefaultFactory();
            response.setContentType("text/html; charset=ISO-8859-1");
            pageContext = _jspxFactory.getPageContext(
                this, request, response,
                null, true, 8192, true);
            application = pageContext.getServletContext();
            config = pageContext.getServletConfig();
            session = pageContext.getSession();
            out = pageContext.getOut();
            _jspx_out = out;
        }
    }
}
```

```
out.write("<HTML>");
out.write("<BODY>");
out.write("<OL>");
out.write("<LI> ");
out.print( 2+4 );
out.write("");
out.write("<LI> Time = ");
out.print( System.currentTimeMillis() );
out.write("");
out.write("");
GregorianCalendar c = new GregorianCalendar();
int year          = c.get(Calendar.YEAR);
out.write("");
out.write("<LI> Year = ");
out.print( year );
out.write("");
if ( (year) %4 == 0 ) {
    out.write(" ");
    out.write("<LI>It's a leap year");
}
else {
    out.write(" ");
    out.write("<LI>It's not a leap year");
}
out.write("");
out.write("</OL>");
out.write("</OL>");
out.write("</BODY>");
out.write("</HTML>");
}
catch (Throwable t) {
    out = __jspx_out;
    if (out != null && out.getBufferSize() != 0)
        out.clearBuffer();
    if (pageContext != null)
        pageContext.handlePageException(t);
}
finally {
    if (__jspxFactory != null)
        __jspxFactory.releasePageContext(pageContext);
}
}
```

### 3.10.5 Java server pages components (Simplified)

- Expression  
Illustrating call of a method

```
<%= System.currentTimeMillis() %>
```

- Page declaration  
Illustrating an import statement

```
<@ page import="java.util.*" %>
```

- Scriptlet  
Illustrating declarations

```
<% GregorianCalendar c = new GregorianCalendar();  
    int year = c.get(Calendar.YEAR); %>
```

- Scriptlet  
Illustrating use of control constructs

```
<% if ( (year) %4 == 0 ) { %>  
<LI>It's a leap year  
<% } else { %>  
<LI>It's not a leap year  
<% } %>
```

- Points to note
  - Its all created in a single method so:

```
<% int year          = 2010    %>
The year is <%= year %>

<% int year          = 2010    %>
The year is <%= year %>
```

Will fail, duplicate declaration of year

### **3.10.6 JSP Summary**

- Advantages
  - Builds on knowledge of HTML, and Java
  - Allows full access to the Java API and language
  - Do not have to write complete Java program 'Servlet'
  - Unlike javascript is run on the server, so can easily access server side resources
  - JSP is portable across server systems
  - Low cost of ownership  
Can use many open source components
- Disadvantages
  - You will get compile time error messages if you access a page containing syntax errors
  - You will get a stack unwind if a run time error is caused.

### 3.10.7 Servlet

- Java program (Servlet)

That delivers as its output a web page. This specially written servlet outputs HTML that is to be the dynamically created web page.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class HelloWorld extends HttpServlet {

    public void doGet(HttpServletRequest request,
                      HttpServletResponse response)
        throws IOException, ServletException
    {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        out.println("<HTML>");
        out.println("<BODY>");
        out.println("<H1>Hello World!</H1>");
        out.println("</BODY>");
        out.println("</HTML>");
    }
}
```

## 3.11 Java Script

- Scripting language that is run client side

The language looks superficially like Java and is embedded in a web page using special tags `<SCRIPT>` `</SCRIPT>` to identify code. It is executed by the browser during the display of the web page. This can result in web pages with dynamic content and or interactive elements.

```
<HTML>

<SCRIPT language="JavaScript">
<!--
function write_message()
{
    var today = new Date();
    var hour = today.getHours()
    var mins = today.getMinutes()
    var secs = today.getSeconds()
    var message = "Time " + ( ( hour<10 ? " " : "" ) + hour ) +
                    ( ( mins<10 ? ":0" : ":" ) + mins ) +
                    ( ( secs<10 ? ":0" : ":" ) + secs ) +
                    " ";
    message = message + "server " + location.host;
    document.ClockForm.data.value = message;
    id = setTimeout( "write_message()", 1000 );
}
//-->
</SCRIPT>

<BODY>

<FORM NAME="ClockForm" ACTION="">
  <INPUT TYPE="text" NAME="data" SIZE=50 VALUE="Data unavailable">
</FORM>

</BODY>

</HTML>
```

### **3.11.1 Javascript**

- Uses
  - Dynamic web page content
  - Validation of data on the client side before being sent for further processing to a server.
  - Creation of interactive web pages.
  
- Advantages
  - Immediacy of client side processing
  - Off loading of processing to client
  
- Disadvantages
  - Executed on client side
  - Not Java, Language is restricted in scope
  - May need to transmit data to server for further processing

## **4. Design consideration**

- Fat vs. Thin
- State vs. Stateless

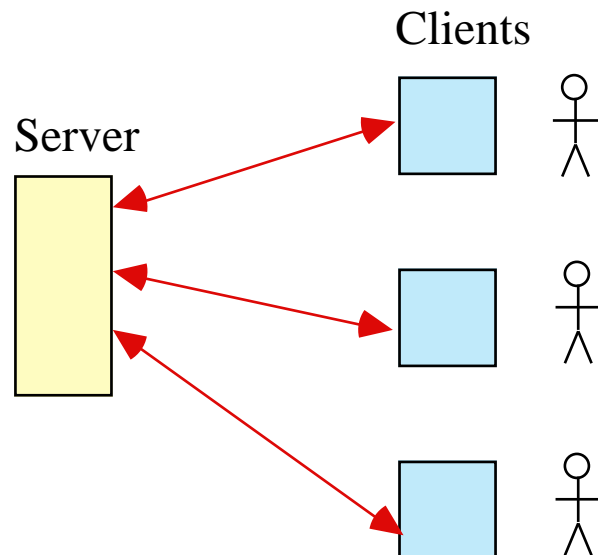
## **5. Multicast**

- What is multicast
- Protocols
- An Internet Chat Application

## 5.1 Multicast

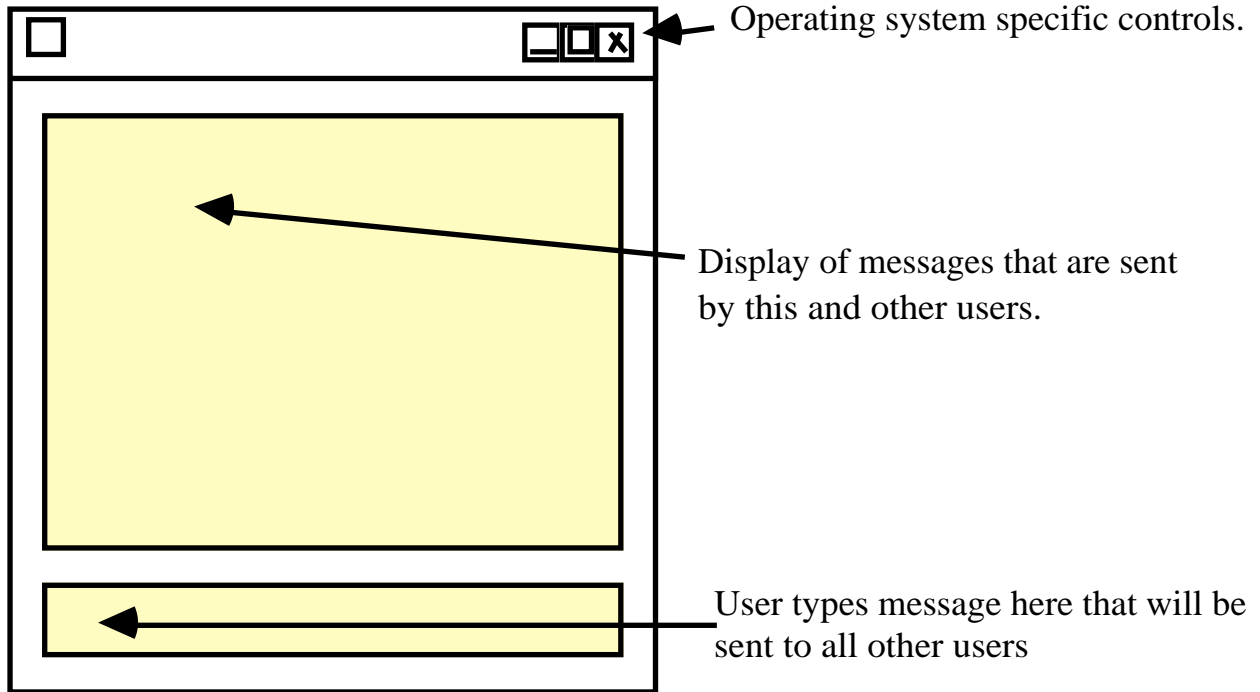
- Ability to send data to multiple clients
- Uses UDP  
So unreliable connection client-server
- Use special IP addresses  
224.0.0.0 - 239.255.255.255  
224.0.0.0 Reserved so do not use
- Time to live parameter

## 5.2 An Internet Chat Application



- Overview
  - Clients send individual entered messages to server
  - Server sends messages back to all clients using Multicast

### 5.3 Client GUI



### 5.3.1 Client code

```
import javax.swing.*;
import javax.swing.event.*;
import java.awt.*;
import java.awt.event.*;
import java.net.*;

import java.io.*;
import java.net.*;
import java.util.*;

import java.io.IOException;

class Client
{
    public static void main( String args[] )
    {
        System.out.println("Chat application");
        Application client = new Application(); //Set up GUI
        client.setVisible(true); //Show GUI
        client.start( args ); //Start
    }
}
```

## 5.3.2 Set up GUI

```

class Application extends JFrame
{
    private static final int H = 400;           //Height of Window
    private static final int W = 300;         //Width of Window

    private JTextField theVIn;                //Visual Input
    private JTextArea theVOut;               //Visual Output
    private JScrollPane theSP;               //Scrollable view

    private String theServer = "localhost";   //Server
    private String theMCA = "224.0.0.1";     //Multicast Add.
    private String theUser = "Guest";        //User of chatline

    private NetMCRReader theIn;              //Input stream
    private NetDGWriter theOut;             //Output stream

    public Application()
    {
        Container cp = getContentPane();     //Content pane
        cp.setLayout( null );                //Set no layout man
        setSize( W, H );                     //Size of Window
        Font font =
            new Font( "Monospaced", Font.PLAIN, 12 ); //Font Used

        theVIn = new JTextField();           //Input area
        theVIn.setBounds( 10, H-70, W-20, 40 ); // Size
        theVIn.setFont( font );              // Font
        cp.add( theVIn );                    //Add to canvas

        theVOut = new JTextArea( "" );       //Output area
        theVOut.setFont( font );             // Font
        theSP = new JScrollPane();           //Scrolling window
        theSP.setBounds( 10, 10, W-20, H-100 ); // Size of window
        theSP.setFont( font );               // Font
        cp.add( theSP );                     // Add to canvas
        theSP.getViewport().add( theVOut ); //onto the text area

        Transaction cb = new Transaction();  //Add listener
        theVIn.addActionListener( cb );     //(Call back)
        setDefaultCloseOperation( EXIT_ON_CLOSE ); //Exit close
    }
}

```

### 5.3.3 Process command line parameters

```
public String params( String args[] )
{
    if ( args.length >= 1 )
    {
        theUser = (args[0] + " ").substring(0,5);
    }
    if ( args.length >= 2 ) theServer= args[1];
    if ( args.length >= 3 ) theMCA   = args[2];
    return "User[" + theUser + "] Server[" + theServer +
           "] MCA[" + theMCA + "] ";
}
```

### 5.3.4 Action when user send message

```
class Transaction implements ActionListener
{
    public void actionPerformed((ActionEvent e) )
    {
        String userInput = theVIn.getText(); //Users response
        theVIn.setText( "" ); //Clear input
        try
        {
            if ( theOut != null )
            {
                theOut.put( theUser + " : " + userInput );
            }
        }
        catch ( IOException err )
        {
            System.out.println("Server: " + err.getMessage());
        }
    }
}
```

### 5.3.5 Main processing loop of client

```
public void start( String args[] )
{
    String info = params( args );           //Initial values
    theVOut.setText( info + " " );         //Display GUI
    System.out.println( info );           //Display Console
    try
    {
        theVOut.setText( "Started" );     //Inform
        theIn = new NetMReader(55000,theMCA); //Connection
        theOut = new NetDGWriter(44000,theServer); //Input

        while ( true )                   //Loop
        {
            String mes = theIn.get();     //From Client
            if ( mes == null ) break;     //No more data
            theVOut.append( mes + " " );  //message
        }
    }
    catch ( IOException err )
    {
        System.out.println("Client " + err.getMessage());
        err.printStackTrace();
    }
}
```

## 5.4 Server code

```
import java.lang.*;
import java.net.*;
import java.util.*;
import java.io.*;

class Server
{
    public static void main( String args[] )
    {
        String mca = "224.0.0.1";           //Multicast address
        if ( args.length == 1 ) mca = args[0];
        System.out.println( "Using MCA[ " + mca + "]" );
        (new Server()).process( mca );     //Start server
    }

    public void process( String mca )
    {
        NetDGReader    in;                 //Input stream
        NetMCWriter    out;                //Output stream

        try
        {
            System.out.println("Server started");
            out = new NetMCWriter(55000, mca ); //Output
            in = new NetDGReader(44000);        //Connection
            while ( true )                     //Forever
            {
                String mes = in.get();         //From Client
                if ( mes != null )
                {
                    out.put( mes );           //Write to all
                }
            }
        }
        catch ( IOException err )
        {
            System.out.println("Server:run " + err.getMessage());
        }
    }
}
```

```

////////////////////////////////////
// Datagram
////////////////////////////////////

// Packets may arrive in different order from that sent
// and some datagram packets may not arrive at all

class NetDGWriter
{
    private DatagramSocket theSocket = null;
    private InetAddress    theAddress = null;
    private int            thePort;
    private String         theHost;

    public NetDGWriter(int port,String host) throws IOException
    {
        thePort = port;
        theHost = host;

        theSocket = new DatagramSocket();
        theAddress = InetAddress.getByName(theHost);
    }

    public void close() throws IOException
    {
        theSocket.close();
    }

    public void put( String message ) throws IOException
    {
        byte[] buf = message.getBytes();
        DatagramPacket packet =
            new DatagramPacket(buf, buf.length, theAddress, thePort);
        theSocket.send(packet);
    }
}

```

```
class NetDGReader
{
    private DatagramSocket theSocket = null;
    private InetAddress    theAddress = null;
    private int            thePort;

    public NetDGReader(int port) throws IOException
    {
        thePort    = port;
        theSocket = new DatagramSocket(port);
    }

    public void close() throws IOException
    {
        theSocket.close();
    }

    public String get() throws IOException
    {
        byte[] buf = new byte[512];
        DatagramPacket packet = new DatagramPacket(buf, buf.length);
        theSocket.receive(packet);

        return new String( packet.getData(), 0, packet.getLength() );
    }
}
```

```

///////////////////////////////////////////////////////////////////
//Multicast using Datagrams
// IP Range 224.0.0.0 to 239.255.255.255
///////////////////////////////////////////////////////////////////

class NetMCWriter
{
    private MulticastSocket theSocket = null;
    private InetAddress     theGroup  = null;
    private int             thePort;

    public NetMCWriter(int port, String mca) throws IOException
    {
        thePort = port;
        theGroup = InetAddress.getByName( mca );
        theSocket = new MulticastSocket( thePort );
        theSocket.setTimeToLive(20);
        theSocket.joinGroup(theGroup);
    }

    public void close() throws IOException
    {
        theSocket.leaveGroup(theGroup);
        theSocket.close();
    }

    public synchronized void put( String message ) throws IOException
    {
        byte[] buf = message.getBytes();
        DatagramPacket packet =
        new DatagramPacket(buf, buf.length, theGroup, thePort);
        theSocket.send(packet);
    }
}

```

```
class NetMCReader
{
    private MulticastSocket theSocket = null;
    private InetAddress      theGroup  = null;
    private int thePort;

    public NetMCReader(int port, String mca ) throws IOException
    {
        thePort    = port;
        theGroup    = InetAddress.getByName( mca );
        theSocket   = new MulticastSocket( thePort );
        theSocket.joinGroup(theGroup);
    }

    public void close() throws IOException
    {
        theSocket.leaveGroup(theGroup);
        theSocket.close();
    }

    public synchronized String get() throws IOException
    {
        byte[] buf = new byte[512];
        DatagramPacket packet = new DatagramPacket(buf, buf.length);
        theSocket.receive(packet);

        return new String( packet.getData(), 0, packet.getLength() );
    }
}
```

## **6. Remote Procedure Call**

- RMI
- Examples

## 6.1 Remote Procedure call

- Ability to call a function/ procedure that is part of any other process which may be executing on a remote machine.
- Semantics similar but not the same as a normal function/ procedure call.
  
- Implementations
  - RMI  
Java
  - CORBA  
Common Object Request Broker Architecture  
(IBM, Sun, Oracle support CORBA)
  - DCOM  
Microsoft
  - SOAP  
Microsoft

- Differences between RPC and normal Procedure calls
  - There may be a significant delay before the function/ procedure is obeyed.
  - The call may fail, due to network problems, remote machine failure, etc.
  - Passing parameters by reference may not be implemented.
  
- Implementations available in Java
  - RMI  
Java code to Java code
  - COBRA  
Java code to other code

## 6.2 Potential problems with RPC and client Server

- Network problems
  - Must be prepared for failure, must be able to recover.
  
- Resource implications
  - Concurrency on server  
A call to an remote procedure will normally execute concurrently with other access to the same procedure on the server  
  
If serialize access to remote procedure  
Then if remote procedure takes a long real time to execute, overall performance could be very slow.
  
- Parameters are serialized
  
- May be an undesirable effect due to call by value of objects (Java normally passes objects by reference) and or serialization of data.

### 6.3 RMI: Simple example

An interface is used to define the protocol that the remote object will implement. As it is a remote object communications to it may fail, in which case the exception `RemoteException` will be thrown

```
interface RemoteAccount extends Remote
{
    public double getBalance()
        throws RemoteException;
    public void deposit( final double money)
        throws RemoteException;
    public double withdraw( final double money )
        throws RemoteException;
    public void setMinBalance( final double money )
        throws RemoteException;
    public double getMinBalance()
        throws RemoteException;
}
```

The class for the remote object is defined as follows:

```
class RAccount extends java.rmi.server.UnicastRemoteObject
    implements RemoteAccount
{
    private double theBalance;           //Balance of account
    private double theMinBalance;       //Minimum bal (Overdraft)

    public RAccount() throws RemoteException {}

    public double getBalance() throws RemoteException
    {
        return theBalance;
    }

    //Rest omitted
}
```

### 6.3.1 RMI Example: Server

The server simply creates an instance of the remote object and binds it to the url `rmi://localhost/account`

```
import java.rmi.*;
import java.io.*;

import java.rmi.server.UnicastRemoteObject;

class Server
{

    public static void main( String args[] )
    {
        connect( "rmi://localhost/account" );
    }

    public static void connect( String url )
    {
        try
        {
            RAccount mike = new RAccount();

            Naming.rebind( url, mike );
            System.out.println( "Bound mike to: " + url );
        }
        catch ( Exception err )
        {
            System.out.println("Error: " + err.getMessage() );
        }
    }
}
```

## 6.3.2 RMI Example: Client

The client binds to the remote object using the url [rmi://localhost/account](#) and then accessing the local stub as if it were the real object. The affect of which is to call the appropriate method on the server.

```
import java.rmi.*;

class Client
{
    public static void main( String args[] )
    {
        process( "rmi://localhost/account" );
    }

    public static void process( String url )
    {
        RemoteAccount mike;
        try
        {
            mike = (RemoteAccount) Naming.lookup( url );

            double obtained;
            System.out.println( "Mike's Balance      = " +
                               mike.getBalance() );

            mike.deposit(100.00);
            System.out.println( "Mike's Balance      = " +
                               mike.getBalance() );

            obtained = mike.withdraw(20.00);
            System.out.println( "Mike has withdrawn : " + obtained );
            System.out.println( "Mike's Balance      = " +
                               mike.getBalance() );
        }
        catch ( RemoteException err )
        {
            System.out.println("Error: " + err.getMessage() );
        }
        catch ( Exception err )
        {
            System.out.println("Error: " + err.getMessage() );
        }
    }
}
```

### 6.3.3 Compiling the system

```
javac RAccount.java
javac Client.java
rmic -v1.2 RAccount
javac Server.java
```

### 6.3.4 Running the system

#### On the server machine

#### VM1: Virtual Machine 1

```
rmiregistry
```

#### VM2: Virtual Machine 2

```
java Server
```

#### On the local machine

```
java Client
Mike's Balance      = 0.0
Mike's Balance      = 100.0
Mike's Balance      = 80.0
```

```
java Client
Mike's Balance      = 80.0
Mike's Balance      = 180.0
Mike's Balance      = 160.0
```

## **7. Clustering**

- Overview
- SMP vs. Clustering

## 7.1 Overview

- A group of interconnected, whole computers working together as a unified computing resource that can create the illusion of being one machine.

Each whole machine in the cluster (Node) can be run on its own.

Stallings Operating systems 4th edition

- Particularly attractive for running server applications.

## 7.2 Clustering advantages

- Absolute scalability  
A cluster can consist of hundreds of machines, each of which may be a multiprocessor
- Increased scalability  
Possible to add new systems to the cluster
- High availability  
Because each node in the cluster is a single machine, loss of node does not result in loss of service.
- Price/performance is superior  
Uses commodity building blocks, cheaper to build a cluster than a single very powerful machine

Stallings Operating systems 4th edition

### 7.3 Types of cluster

Clustering Method	Description	Benefits	Limitations
<b>Passive Standby</b>	Secondary server takes over in case of failure	Easy to Implement	High Cost
<b>Active Secondary</b>	Secondary server does work	Reduced cost work shared between servers	Increased complexity
Separate servers	Data continuous copied from primary to secondary	High availability	High network & server overhead
Servers connected to disks	Servers cabled to same disk. Each server owns its own disks. Disks taken over by other servers on server failure	Reduced network and server overhead as elimination of copying operation	Requires Raid or mirroring to allow for disk failure
Servers share disks	Multiple servers simultaneously share access to disks	Low network & server overhead. Reduced risk of downtime due to disk failure	Requires lock management. Usually used with Raid or disk mirroring

Hewlett Packard White paper on clustering <http://www.hp.com/>

## **7.4 Desirable middleware properties for clusters**

- Looks like a single machine
- Single file system
- Single memory space  
Distributed shared memory allows 'variables' to be shared
- Single job management system
- Single I/O space  
Any peripheral can be accessed
- Single process space  
Any process can communicate with any other process
- Checkpointing  
Periodic save of state to allow for rollback after failure
- Process migration  
A process may move between machines in the cluster

## 7.5 Operating system design issues

- Failure management
  - Highly available systems
    - Offers a high probability that all resources are available
    - If a component goes down then a transaction that is in progress will be re-done on an available machine
    - Application software must handle the effect of the partial completion before failure
  - Fault tolerant systems
    - Ensures that all components are always available.
    - Achieved by the use of redundant shared disks and backing out uncommitted transactions and committing completed transactions.

Failover  
The action of switching an application and data to an alternative system
- Load balancing
  - If a machine is added or removed from the cluster then the load should be redistributed across available resources.
- Parallelizing computation
  - Execution of software from a single application in parallel

## **7.6 Clustering vs. SMP**

- SMP is easier to manage
- SMP is closer to uniprocessor model
- SMP is well established and stable
- Cluster scales better (Incremental & absolute)
- Cluster better availability

## **7.7 Beowulf [www.beowulf.org](http://www.beowulf.org)**

- Made from mass market commodity components  
PC's running Linux
- Dedicated processors  
Make up cluster
- No custom components
- Private network

## **7.8 Beowulf II**

- Looks like a single machine
- Uses modified standard Unix tools for process control

## 8. Types of servers

- Connectionless (UDP) vs. Connection-oriented (TCP)
- Iterative vs. Concurrent
- Stateless vs. Stateful

## **8.1 Connectionless vs. Connection-Oriented**

- **Connectionless**

No specific connection to server, information may be lost  
e.g. Streaming video

+ Low overhead

- Non reliable transmission
- If required client will have to manage re-transmission

- **Connection-Oriented**

Specific connection between client & server  
e.g. ftp, web

+ Reliable transmission of data

- Overhead of initial connection setup
- Require separate sockets for each connection

## 8.2 Iterative vs. Concurrent

- Iterative

Single processor  
Handles requests sequentially  
Requests take short time to process  
E.g. time server

+ Easy to Implement

- Server is unavailable for other work whilst processing the request
- Will not scale, other than by adding a faster CPU/  
Disks

- Concurrent

Handles multiple requests concurrently (Web server)  
Normally uses threads/ lightweight processes  
High volume

- + Requests can be overlapped
- + Will scale on suitable hardware

- Harder to implement / Problems of concurrency

### 8.3 Stateless vs. Stateful

- **State**  
Information maintained by server about on-going transactions between client and server.  
  
How long do you preserve state?  
How do you identify which state belongs to which client?  
How do you stop state information being forged?
- **Stateless**  
No information held by server about on-going transaction  
NFS (Network File System)  
HTTP is stateless as if either the server or client goes down the underlying process may be resumed with no ill-effect.  
(Of course we now put state information into the http protocol with cookies, etc.)
- **Statefull**  
Server maintains the state of the transactions so far made by a client  
  
e.g. On-line Banking system  
  
In statefull mode there may be a login to the server when this has been done then transactions may be performed.  
  
Problem of authentication

## 8.4 Classification grid

	Connection-Oriented	Connectionless
Iterative	Not usually used	Simple servers
Concurrent	Rest e.g. Charon	NFS, DNS

## **8.5 Server management (Linux)**

- Many servers on linux  
NFS, Apache, Webmin, FTP,
- Some of the servers used infrequently
- `inetd`  
Is a demon that starts servers on demand to process a request.
  - This reduces the load (resources used) on the machine. As a server is only run when needed.
  - Load consists of Idle CPU time for process, Memory used (Swap space if paged out) etc.