CIS 339

Introduction to Client-Server Systems

Client-Server Architecture

- A Server is an application that provides a service (i.e., store/retrieve data, process a transaction, etc.).
- A Client is an application that uses the service (i.e., requests data).
Systems Described

• Custom network programming.
• The Web (HTTP)
  – Javascript
  – Java™ Applets
  – Common Gateway Interface (CGI)
  – Java™ Servlets
  – Java™ Server Pages (JSP™)
• Database Client-Server
  – JDBC™

Network Layers

- Application Layer
- Transport Layer
- Internet Layer
- The Host-To-Network Layer (*Ethernet, etc.*)
- Logical path
- Physical path
Sockets

• A socket consists of an IP address and a port number.
  – IP addresses are 32-bits long and are expressed in a base 256/base 10 format as follows:
    • 123.234.212.121
  – Host names are generally assigned to IP addresses.
  – Ports are 16-bits long and are expressed in base 10. (Note that the values below 1024 are reserved.)

Initiating Communication

The client initiates the connection to a known port on the server from whichever port is available on the client.

The server accepts the connection. Input and output streams are connected to the sockets on the specified ports.
Socket Programming Support

- Socket programming support is provided by various tools including Java.

Java Server Example

```java
public class SimpleServer {
    public static void main(String[] args) {
        int nextNumber = 0;
        try {
            ServerSocket server = new ServerSocket(8001);
            while (true) {
                Socket connection = server.accept();
                try {
                    OutputStreamWriter out =
                        new OutputStreamWriter(connection.getOutputStream());
                    String reply = Integer.toString(++nextNumber) + "\n";
                    out.write(reply);
                    out.flush();
                    connection.close();
                } catch (IOException e) {/* do nothing */}
                finally {
                    try {
                        if (connection != null) {
                            connection.close();
                        }
                    } catch (IOException e) {}
                }
            }
        } catch (IOException e) {System.err.println(e);}
    }
}
```
Java Client Example

```java
public class SimpleClient {
    public static void main(String[] args) {
        String hostname = "localhost";
        if (args.length > 0) {hostname = args[0];}
        BufferedReader networkIn = null;
        try {
            Socket theSocket = new Socket(hostname, 8001);
            networkIn = new BufferedReader(
                new InputStreamReader (theSocket.getInputStream()));
            String theLine = networkIn.readLine();
            int nextNumber = Integer.parseInt(theLine);
            System.out.println("The next number is : "+nextNumber);
        } catch (IOException e) {
            System.err.println(e);
        }
        finally {
            try {
                if (networkIn != null) networkIn.close();
            } catch (IOException e) {/* do nothing */}
        }
    }
}
```

HTTP

- HyperText Transport Protocol (HTTP) is used by a client to request and receive a resource from a server.
- An HTTP client (Web browser) sends a request to a server on a predefined port (80) for a resource.
- The server retrieves the resource (usually a file) and returns it to the client.
- The client then interprets the resource (usually an HTML file, but it may be an image, audio/video stream, or a Java Applet).
HTML

• HTML stands for HyperText Markup Language.
• This is a text file that includes formatting tags.
  – A formatting tag is of the form:
    `<tag parameter="value"> .. </tag>
  – There may be zero or more parameters associated with a tag.
• In addition to formatting, tags are provided to:
  – Generate forms
  – Transmit data back to the server

JavaScript

• Most browsers include the ability to interpret a language known as JavaScript that is included within an HTML file.
• JavaScript can dynamically generate additional HTML that is then interpreted and displayed.
• JavaScript can also validate user’s input before data is transmitted back to the server.
Java Applets

- Embedded within the HTML can be a request for a Java Applet.
- A Java Applet is a class derived from the parent class Applet, that can display graphics in a pre-allocated window.
- A Java Applet can do almost anything, but is confined to its “sandbox”.
  - It cannot write to the client computer’s file system.
  - It can only communicate with the server from which it was loaded.

Common Gateway Interface

- One possible “resource” that can be retrieved from the server is a Common Gateway Interface (CGI) script.
- A CGI script is a program that executes on the server computer.
  - The resource request from the client is piped in to standard input.
  - The CGI script’s output is transmitted back to the client.
- Generally the CGI script generates HTML, but it can generate anything that the client can interpret, e.g. an image.
CGI Example

- CGI scripts are generally written in Perl but can be written in any language.
- Here is a CGI script in C++

```c++
#include <iostream>
using std::cout;
int main()
{
    cout << "Content-type: text/html\r\n\r\n";
    cout << "<html>\r\n";
    cout << "<body>\r\n";
    cout << "<h1>Hello from CGI</h1>\r\n";
    cout << "</body>\r\n";
    cout << "</html>\r\n";
    return 0;
}
```

Java Servlets

- The CGI scripts are generally run by the server as a separate process.
- Some servers, written in Java, are designed to dynamically load and execute a Java class known as a Servlet.
- A Servlet is like a CGI script in that it also (generally) generates HTML that is sent back to the client.
Java Server Pages

- Coding HTML as output statements is quite cumbersome and error prone.
- A Java Server Page is an HTML page with embedded Java code.
- This is translated into a Servlet that generates the page and executes the code fragments.
- Unlike JavaScript, the code is executed by the server as part of the page generation.

Example: HelloWorld Servlet

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

public class HelloWorldServlet extends HttpServlet {
    public void service (HttpServletRequest request,
                         HttpServletResponse response)
                        throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Hello World</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h1>Hello World</h1>");
        out.println("It's " + (new Date().toString()) + " and all is well.");
        out.println("</body>");
        out.println("</html>");
    }
}
```
Output from HelloWorldServlet

Example: HelloWorld.jsp

```jsp
<%@ page language="java" contentType="text/html"%>
<html>
<head>
<title>Hello World</title>
</head>
<body>
<h1>Hello World</h1>
The time is <%= new java.util.Date().toString()%> and all is well.
</body>
</html>
```
Java Bean

- A Java Bean is a Java class that has the following characteristics:
  - Has a default constructor. (I.e., a constructor that takes no arguments.)
  - Has private attributes, and no public attributes.
  - Each attribute that is accessible has an accessor function named:
    - `type get.Name()`
  - Each attribute that is mutable has a mutator function named:
    - `void set.Name(type parameter)`
date.jsp

```jsp
<%@ page language="java" contentType="text/html"%>
<html>
<body>
<jsp:useBean id="clock" class="java.util.Date"/>
The current time at the server is:
<ul>
<li>Date: <jsp:getProperty name="clock" property="date"/>
<li>Month: <%=clock.getMonth()+1 %>
<li>Year: <%=clock.getYear()+1900 %>
<li>Hours: <jsp:getProperty name="clock" property="hours"/>
<li>Minutes: <jsp:getProperty name="clock" property="minutes"/>
</ul>
</body>
</html>
```

Output from date.jsp

The current time at the server is:
- Date: 8
- Month: 9
- Year: 2001
- Hour: 11
- Minute: 13
HTML Forms

• A form on a web page is defined using the following HTML:
  
  ```html
  <form action="url" method="[POST|GET]">
    Form content
  </form>
  ```

• When the form is submitted, the action is to send the data collected to the specified `url` using either a `POST` or `GET` request.

• The form content consists of other HTML text that is displayed intermixed with input field specifications.

---

GET Method

• The default HTML request is a GET.

• Requesting:
  ```
  http://host/resource
  ```

• Results in the browser opening a socket with host and then sending
  ```
  GET /resource HTTP/version
  ```
GET With Parameters

• When parameters are required they are encoded as part of the resource as follows:
  \(?param1=value1\&parm2=value2\&\ldots\&parmn=valuen\)

• Special encoding is applied to the values:
  – Spaces are replaced by +
  – Non-alphabetic characters are replaced by %nn
    • Where nn is the hexadecimal ASCII code.

The POST Method

• With the POST method, the data is sent on separate line(s) following the resource line.

• Advantages:
  – A large amount of data can be transmitted.
  – The data does not appear in the url, thus the request cannot be bookmarked.

• Disadvantage:
  – The data does not appear in the url, thus the request cannot be bookmarked.
Text Input

- To specify a text field:
  
  `<input type="text" name="name"
     size="size" maxlength="maxlength"
     value="default value">`

- This provides a field with the given `name`, with space for `size` characters, input limited to `maxlength`, and the given `default value`.
- Other than `name`, all of the fields are optional.

Password

- To avoid “shoulder surfing” the input type `password` is used the same as `text`.
- However, when the user enters data, the input characters are replaced by *’s.
Multiple-Line Text

- Multiple lines of text can be collected as follows:
  `<textarea rows="nrow" cols="ncol"
    name="name">
    Default text
  </textarea>`
- This allocates a text input area of `nrow` by `ncol`.
- The actual size of the input is not limited.

Pull-Down Menus

- A pull-down menu item is specified as follows:
  `<select name="name" size="size">
    <option>Option Value1</option>
    ...
    <option>Option Value2</option>
  </select>`
- If `size` is specified, a scrolling box with `size` lines is displayed. Otherwise all options are provided.
Check Boxes

• A check box is defined as follows:
  \[
  \text{\textless input type="checkbox" name="name" value="value" \textgreater}
  \]

Radio Button

• A radio button is similar to a check box except that only one option may be selected at any given time.
• To specify a radio button:
  \[
  \text{\textless input type="radio" name="name" value="value" \textgreater}
  \]
Submit and Reset Buttons

- To trigger the action, a submit or reset button is required.
- This is specified as follows:
  ```html
  <input type="[submit|reset]"
         value="value" name="name">
  ```

Tables

- Tables present data in rows and columns, where the information in each column is aligned.
- A simple table is defined as follows:
  ```html
  <table>
    <tr>
      <td data for row 1 col 1</td>
      ...
      <td data for row 1 col n</td>
    </tr>
    ...
    <tr>
      <td data for row 2 col 1</td>
      ...
    </tr>
  </table>
  ```
Form Example

```html
<html>
<head>
<title>User Info Entry</title>
</head>
<body>
<form action="userinfo1.jsp" method="POST">
<table>
<tr>
<td>Name:</td>
<td><input type="text" name="userName"></td>
</tr>
<tr>
<td>Birth Date:</td>
<td><input type="text" name="birthDate"></td>
<td>(Use format yyyy-mm-dd)</td>
</tr>
<tr>
<td>Sex:</td>
<td><input type="radio" name="sex" value="male"> Male
<input type="radio" name="sex" value="female"> Female</td>
</tr>
</table>
<input type="submit">
</form>
</html>
```
Processing Submitted Data

- To process submitted data with a Java Server Page, define a Java Bean with attributes that correspond to the names of the data values.
- The statements:

  `<jsp:useBean id="name" class="class"/>
  <jsp:setProperty name="name" property="*"/>
  </jsp:useBean>

- Will set each of the attributes of the object `name` to the corresponding submitted values.

UserInfoBean

```java
public class UserInfoBean {
    public UserInfoBean() {
        userName = null;
        birthDate = null;
        sex = null;
    }
    public String getUserName() {return userName;}
    public void setUserName(String name) {this.userName = name;}
    public String getBirthDate() {return birthDate;}
    public void setBirthDate(String birthDate) {
        this.birthDate = birthDate;
    }
    public String getSex() {return sex;}
    public void setSex(String sex) {this.sex = sex;}
    private String userName;
    private String birthDate;
    private String sex;
}
```
UserInfo.jsp

```html
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<title>User Info Display</title>
</head>
<body>
<jsp:useBean id="userInfo" class="UserInfoBean">
<jsp:setProperty name="userInfo" property="*"/>
</jsp:useBean>
You entered the following:
<table>
<tr><td>Name:</td><td><%=userInfo.getUserName()%></td></tr>
<tr><td>Birth Date:</td><td><%=userInfo.getBirthDate()%></td></tr>
<tr><td>Sex:</td><td><%=userInfo.getSex()%></td></tr>
</table>
</html>
```

Output from UserInfo.jsp
Relational Database

- A relational database consists of one or more tables.
- A table consists of one or more columns and multiple rows.
- A column represents an attribute of an entity.
- A row represents an entity.
- To speed access, an index can be defined on one or more columns.

Historical Note

- The term relational refers to concepts developed by E. F. Codd.
- He suggested that the columns represent sets.
- The universe is the Cartesian product of these sets.
- An entity is one item from this universe.
- While no actual implementation follows this model, the closest approximation is a relational database in which each column of each table is indexed.
Structured Query Language (SQL)

• To manage, enter data into, and retrieve data from a relational database, the Structured Query Language (SQL, pronounced sequel) was developed by IBM.

Creating a Table

• A table is created as follows:
  ```sql
  CREATE TABLE tblname (  
  col1name col1type,  
  col2name col2type,  
  ...  
  colname colnype)  
  ```

• There are several data types including:
  - INT      an integer
  - FLOAT    a floating
  - DATETIME a fixed length character string
  - CHAR     a varying length character string
  - TEXT     a large amount of text
Entering Data Into a Table

• Data is entered into a table as follows:

```sql
INSERT INTO tablename (col1name, col2name, ... colnname) VALUES ('value1', 'value2', ... 'valuen')
```

Retrieving Data from a Table

• The SQL statement:

```sql
SELECT col1, col2, ... coln FROM table WHERE criteria
```

• Creates a temporary table consisting only of the selected columns and the rows which satisfy the criteria.
Join

- The select statement can select columns from more than one table. For example:
  
  ```sql
  SELECT sup_name, cof_name FROM suppliers, coffees WHERE sup_name='Acme, Inc.' and suppliers.sup_id=coffees.sub_id
  ```

- This joins the tables `suppliers` and `coffees` and matches entries with a common value of `sup_id`.

- It then selects those entries whose `sup_name` column has the value 'Acme, Inc.'

Database Server

- A database server is a program that listens on a predefined port for database commands (generally SQL statements).

- It then executes the command and returns either a status response or, in the case of select statements, the resulting selected data.
Database Client

• A database client is a program that sends commands to a database server, and processes the results.
• For example, a simple client may permit the user to enter SQL commands, which are then sent to the server, and display the results.

JDBC™

• JDBC is part of the Java application programming interface (API).
• The protocol used by database servers is unique (and maybe proprietary) to that server.
• JDBC provides a platform independent programming interface that can be used to develop database client software.
The JDBC Library Structure

- Java Application
- JDBC API
- JDBC Driver Manager
- JDBC Driver
- Database

Loading the Driver

- The driver is loaded using the following Java code:

```java
try {
    Class.forName("full name of driver");
} catch (Exception E) {
    System.err.println("Unable to load driver.");
    E.printStackTrace();
}
```
Establishing a Connection

- A connection to the database server is established as follows:
  ```java
  String url = "jdbc:mysql://host/database";
  try {
      con = DriverManager.getConnection(url, "userId", "password");
  }
  ```

Creating a Statement Object

- The statement object is used to send SQL commands to the database server. It is created as follows:
  ```java
  Statement stmt = con.createStatement();
  ```
Adding or Modifying Data

• The `executeUpdate` method of the `Statement` interface is used to execute a
  INSERT, UPDATE, or DELETE SQL command, or a SQL command that has no return value.

  ```java
  int executeUpdate(String statement)
  throws SQLException
  ```

• The return value is the row count for UPDATE, INSERT, or DELETE or 0 for those SQL
  commands that have no return value.

Retrieving Data

• The method `executeUpdate` of the `Statement` interface returns the results of
  a SQL query as a `ResultSet`.

  ```java
  ResultSet executeUpdate(String statement)
  throws SQLException
  ```
ResultSet

- A table of data representing a database result set, which is usually generated by executing a statement that queries the database.
- A ResultSet object maintains a cursor pointing to its current row of data. Initially the cursor is positioned before the first row. The next method moves the cursor to the next row, and because it returns false when there are no more rows in the ResultSet object, it can be used in a while loop to iterate through the result set.
- Data is retrieved using the getXXX methods, where XXX is the Java data type.

Java Program to Create a Table

```java
import java.sql.*;

public class CreateCoffees {
    public static void main(String args[]) {
        String url = "jdbc:mysql://patw/coffeebreak";
        Connection con;
        String createString = "create table COFFEES " + "("COF_NAME VARCHAR(32), SUP_ID INTEGER, " + "PRICE FLOAT, SALES INTEGER, TOTAL INTEGER)";
        try {
            Class.forName("org.gjt.mm.mysql.Driver");
            catch (Exception E) {
                System.err.println("Unable to load driver.");
                System.err.println(E.getMessage());
                E.printStackTrace();
            }
            try {
                con = DriverManager.getConnection(url, "paul", "secret");
                Statement stmt = con.createStatement();
                stmt.executeUpdate(createString);
                stmt.close();
                con.close();
            } catch (SQLException ex) {
                System.err.println("SQLException: " + ex.getMessage());
            }
        }
    }
}
```
Java Program to Insert Data

```java
import java.sql.*;

public class InsertCoffees {
    public static void main(String args[]) {
        String url = "jdbc:mysql://patw/coffeebreak";
        Connection con;
        Statement stmt;
        try {
            Class.forName("org.gjt.mm.mysql.Driver");
        } catch (Exception E) {
            System.err.println("Unable to load driver.");
            System.err.println(E.getMessage());
            E.printStackTrace();
        }
        try {
            con = DriverManager.getConnection(url, "paul", "secret");
            stmt = con.createStatement();
            stmt.executeUpdate("insert into COFFEES " +
                                "VALUES('Columbian', 101, 7.99, 0, 0)");
            stmt.executeUpdate("insert into COFFEES " +
                                "VALUES('French_Roast', 49, 8.99, 0, 0)");
            stmt.executeUpdate("insert into COFFEES " +
                                "VALUES('Espresso', 150, 9.99, 0, 0)");
            stmt.executeUpdate("insert into COFFEES " +
                                "VALUES('Colombian_Decaf', 101, 8.99, 0, 0)");
            stmt.executeUpdate("insert into COFFEES " +
                                "VALUES('French_Roast_Decaf', 49, 9.99, 0, 0)");
            stmt.close();
            con.close();
        } catch (SQLException ex) {
            System.err.println("SQLException: " + ex.getMessage());
        }
    }
}
```

Java Program To Retrieve Data

```java
import java.sql.*;

public class Join {
    public static void main(String args[]) {
        String url = "jdbc:mysql://patw/coffeebreak";
        Connection con;
        String query = "select SUP_NAME, COF_NAME " +
                        "from COFFEES, SUPPLIERS " +
                        "where SUP_NAME = 'Acme, Inc.' and " +
                        "SUPPLIERS.SUP_ID = COFFEES.SUP_ID";
        Statement stmt;
        try {
            Class.forName("org.gjt.mm.mysql.Driver");
        } catch (Exception E) {
            System.err.println("Unable to load driver.");
            E.printStackTrace();
        }
        try {
            con = DriverManager.getConnection(url, "paul", "secret");
            stmt = con.createStatement();
            ResultSet rs = stmt.executeQuery(query);
            System.out.println("Supplier, Coffee:");
            while (rs.next()) {
                String supName = rs.getString(1);
                String cofName = rs.getString(2);
                System.out.println("    " + supName + " , " + cofName);
            }
            stmt.close();
            con.close();
        } catch (SQLException ex) {
            System.err.println("SQLException: " + ex.getMessage());
        }
    }
}
```
A JSP to Display Data

```jsp
<%@ page language="java" contentType="text/html" %>
<html>
<head>
<title>Coffees supplied by Acme, Inc.</title>
</head>
<body>

String url = "jdbc:mysql://patw/coffeebreak";
java.sql.Connection con;
String query = "select SUP_NAME, COF_NAME " +
"from COFFEES, SUPPLIERS " +
"where SUP_NAME = 'Acme, Inc.' and " +
"SUPPLIERS.SUP_ID = COFFEES.SUP_ID";
java.sql.Statement stmt;
try {
    Class.forName("org.gjt.mm.mysql.Driver");
} catch (Exception E) {
    System.err.println("Unable to load driver.");
    E.printStackTrace();
}
try {
    con = java.sql.DriverManager.getConnection(url, "paul", "secret");
    stmt = con.createStatement();
    java.sql.ResultSet rs = stmt.executeQuery(query);
}

<table border="1" cellpadding="5" cellspacing="0" width="300">
<tr>
    <td>Supplier</td>
    <td>Coffee</td>
</tr>
<% while (rs.next()) { %>
<tr>
    <td><%=rs.getString(1)%></td>
    <td><%=rs.getString(2)%></td>
</tr>
<% } stmt.close();
    con.close();
} catch(java.sql.SQLException ex) {
    System.err.println("SQLException :");
    System.err.println(ex.getMessage());
}
</table>
</body>
</html>
```

Join.jsp (part 2)

```jsp
<table border="1" cellpadding="5" cellspacing="0" width="300">
<tr>
    <td>Supplier</td>
    <td>Coffee</td>
</tr>
<% while (rs.next()) { %>
<tr>
    <td><%=rs.getString(1)%></td>
    <td><%=rs.getString(2)%></td>
</tr>
<% } stmt.close();
    con.close();
} catch(java.sql.SQLException ex) {
    System.err.println("SQLException :");
    System.err.println(ex.getMessage());
}
</table>
</body>
</html>
```
Caveats

- The preceding example has several problems:
  1. The database host, database, user ID, and password are hard-coded in the jsp.
  2. There is a lot of intermixing of Java control structures with HTML structures.
     - Alternatives will be presented in a subsequent lecture(s).