Introduction to Java and the Network

Michael B. Spring
Department of Information Science and Telecommunications
University of Pittsburgh
spring@imap.pitt.edu
http://www.sis.pitt.edu/~spring
Overview
Java and the Network

• Introduction to Client-Server Computing
• Threads
• Socket programming in Java
• Serialization
• Security Management
• RMI programming in Java
• Application Serving in Java – J2EE
• Enterprise Java Beans
• Server Services and Deployment
Business Case for Client Server

• Supports heterogeneous platforms
• Gives end users more control over computing
• Better utilizes distributed desktop devices
• Integrates enterprise-wide data sources
• Allows use of personal tools with corporate data
• Claims competitive cost/benefit performance
Client-server computing means different things to different people
• Some equate ODBC connections with Client Server
• Some equate Forms/CGI or Applets with Client Server
• Some equate RPC or RMI with Client Server

Client Server systems can be developed at many levels
• low level protocols (TCP/IP, UDP)
• High level protocols (RMI, RPC, CORBA)
• application protocols (SMTP, SNMP, etc.)
• Use of existing application protocols (ODBC, JDBC, XML)

Client Server computing is any computing that distributes functions among two or more processes where the processes are some combination of requestors (clients) and responders (servers)
Technological Underpinnings

- Desktop computing capability that incorporates GUI capabilities and network connections
- Networks that are well connected (Internet), of reasonable bandwidth, and supportive of various connection quality (TCP)
- Enterprise Databases that allow for transaction management, data distribution, and SQL manipulation
- Standardized infrastructure for:
  - Networking (TCP/IP, Ethernet, IPX)
  - Operating System (UNIX, Windows, NT)
  - Data interchange (SQL, ODBC, RPC/RMI, CORBA)
Benefits

- **Management**
  - Flexible, adaptable, and cost effective system
  - Improved productivity and organizational workflow
  - Improved customer relations and satisfaction

- **Information Systems Development Group**
  - Platform independence
  - Processing tasks are optimized among systems
  - Reduced development/implementation costs and time
  - Scalable and portable systems
  - Reduced operational costs
  - Enhanced information deployment and utilization
Disadvantages

- Application development has added networking component
- Systems are complex to test, debug and manage
- Programmers need to be retrained
- Security is more difficult, particularly for the server components
- Requires a highly reliable network
- Consumes a lot of network bandwidth
Redefinition of Client/Server

- A model of computing that distributes functions among two independent and autonomous processes, called clients and servers.
- What is a process?
  - A process is an address space that contains data and instructions required to perform a computing task.
- Server and client processes may execute on the same or different machines.
  - A SERVER process runs continuously providing services in response to CLIENT requests.
  - A CLIENT process requests specific services from one or more SERVER processes.
Client/Server Paradigm (UDP)

Client Initiates Request for services

Request

Network

Response

Server Responds to request for services
Client/Server Paradigm (TCP)

- A server, or daemon, is started and waits for a request for service
- A client initiates a connection to a server
- The server accepts the connection providing an endpoint for communications with the client. (normally, the server will handle the client request in a separate process or thread.)
- The client requests a service and awaits a response
- The server acts on the client request, sends a response, and waits for the next client request.
Client/Server Architecture

Components

- **Client**: a computer process that initiates a request from a server process. Frequently the client has a graphical user interface.
- **Communications middleware**: any computer process through which client and server processes communicate. Usually has several layers to support the transmission and control of messages. TCP is communications middleware.
- **Server**: a computer process listens for a request from client processes and responds with the service it provides.
- **Persistent Store**: permanent storage used by the server to maintain data.
Some Things to Keep in Mind

- The separation of processing tasks is the key difference between client/server computing and mainframe computing.
- It is technically easier to build a client than a server – even though it may have more code!
  - The client will normally require a Graphical User Interface and is most complex in terms of how it responds to interleaved requests from the user and the socket.
  - The server design is complicated by the fact that it is generally invisible once started, must be designed so as to handle virtually simultaneous requests for service from multiple clients without confusing data.
  - The server design is further complicated by the need to insure that malicious connections to the service do not gain access to system resources in a way that is not intended.
Client/Server Example

Database server

HP-RISC

Image server

SUN

Clients

Windows

OSF/1

MAC
Client/Server Examples

- File-Server paradigm is a simple example of client/server computing restricted by a single server process operating on a single computer system.
- True client/server computing is a many-to-many relationship between client and server processes.
- A server process may provide its serve many different clients – e.g. a time server.
- A client process may request services from different servers – e.g. a DBMS server and a time server.
- Examples of services include file services, database services, e-mail services, fax services, print services, and application services.
Protocols for Client Server
Classes of Protocols

• Protocols exist at different levels and at some levels come in different flavors
  • On a local area network, different protocols move information between machines -- IEEE 802.3, 802.4, 802.5, 802.7 are examples
  • Above this level are addressing protocols that identify machines beyond local area networks. IP is the most famous, but LAT, Appletalk, and SPX/IPX are other examples.
  • Above addressing, there are protocols to insure quality of the connection. Again, the most famous of these is TCP, but SPX/IPX and other performs similar functions
  • Finally, there are the application protocols which govern the structure of the client server conversation. Some of the more famous are HTTP, SMTP, FTP, and SNMP
At the simplest level, most modern client server systems are built on top of:

- Local area network protocol: Ethernet
- Addressing protocol: IP
- Transport protocols: TCP

An existing application protocol (FTP, SMTP, SNMP, HTTP, etc.) may be used to support the dialog.

A user defined protocol may be defined using RPC/XDR, RMI, CORBA, J2EE, SFS/JESI
Local Area Transport Protocols are used to move information to specific machines on a segment.

**802.3**
- Preamble (64)
- Destination (16/48)
- Source (16/48)
- Len (16)
- DATA (n)
- LLC-Data-PAD
- CRC(32)

**Ethernet**
- Preamble (64)
- Destination (48)
- Source (48)
- Type (16)
- DATA (n)
- CRC (32)
The IP Protocol/Packet are used to identify individual machines uniquely and move data between them

<table>
<thead>
<tr>
<th>Ver</th>
<th>IHL</th>
<th>Type of Service</th>
<th>Total Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identifier</td>
<td>Flags</td>
<td>Fragment Offset</td>
<td></td>
</tr>
<tr>
<td>Time to Live Protocol</td>
<td>Header Checksum</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Source Address

Destination Address

Options and Padding (20 octet minimum)
IP Packet Notes

- **IHL** - Internet Header Length in 32 bit words
- **Type Service**
  - bits (0-2) network control, Inet control, Flash, Override
  - bit-3 = delay (norm, high), bit-4=Thruput (norm, high)
  - bit-5 = reliability(norm,high), bit-6/7 = reserved
Transport Protocols

- The transport protocols define the contents or data contained within an IP packet. While IP acts as an envelop for an individual note going across a network, UDP and TCP control the message.
  - UDP makes the addressing information available to the application and provides minimal assurance of data integrity. If a communication requires more than one packet, UDP assumes no responsibility for the overall communication.
  - TCP provides an end to end connection between two addresses and provides an assurance that the data transmitted by one is delivered to the other even if multiple packets are involved.
## UDP Packet

<table>
<thead>
<tr>
<th>Source Port</th>
<th>Destination Port</th>
</tr>
</thead>
<tbody>
<tr>
<td>Length</td>
<td>Checksum</td>
</tr>
</tbody>
</table>

Data
<table>
<thead>
<tr>
<th>TCP Packet</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="TCP Packet Diagram" /></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source Port</td>
<td>Destination Port</td>
</tr>
<tr>
<td>Sequence Number</td>
<td>ACK Number</td>
</tr>
<tr>
<td>Offset</td>
<td>Rsrvd</td>
</tr>
<tr>
<td></td>
<td>U</td>
</tr>
<tr>
<td></td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>P</td>
</tr>
<tr>
<td></td>
<td>R</td>
</tr>
<tr>
<td></td>
<td>S</td>
</tr>
<tr>
<td></td>
<td>F</td>
</tr>
<tr>
<td>Window</td>
<td>Urgent Pointer</td>
</tr>
<tr>
<td>Checksum</td>
<td>Options and padding</td>
</tr>
<tr>
<td>Data</td>
<td></td>
</tr>
</tbody>
</table>
TCP Packet Notes

- Minimum Length is 20 octets
- Port addresses are 16 bits - Logical identification of host process
- SEQuence Number - Number for each octet sent
- Flags as follows
  - **U** - Urgent Field Pointer is significant
  - **A** - ACK number field is significant
  - **P** - Push Function (send all queued data)
  - **R** - Reset the connection
  - **S** - SYN - synchronize sequence numbers. When SYN Flag set = Initial Sequence Number (ISN)
  - **F** - FIN - No more data from sender
- **Window** - Number of octets receiver will accept before an ACK
- **Checksum** - On parts of the header (pseudo-header)
- **Urgent Pointer** - Position of non-urgent data in data field
- **Options & Padding** - Options such as max TCP segment size
Using Ethernet or other local area transport protocols, two machines can address each other using IP and can communicate using TCP. However, what is said across this connection is determined by the application protocol that is used.

- Telnet, ftp, http, smtp(mail), snmp, etc are all examples of application protocols that you have used.
- Anyone may develop a protocol to be used between client server processes.
- A protocol is simply a set of rules to maintain order, consistency and correct message handling between to communicants – in this case client and server processes.
Every client server application needs a protocol.

Many client server applications uses existing general protocols. For example, HTML forms and CGI programs use the http protocol to exchange information.

The telnet protocol, which defines format of data in exchanged messages, how characters are encoded, control messages and error handling, is used for many other applications.

Frequently, and especially for RMI and straight socket programming, the developer defines a unique protocol.
Port numbers are used as part of the IP address scheme. Thus, when you connect to a given machine, you identify not only the IP address, but a port number as well.

When a server process is started, it is assigned a port on the machine at which it listens for requests.

For this reason, the protocols used by widely run servers (mail, WWW, etc.) are associated with particular ports on which they are run. These ports are known as well known ports. FTP is run on port 21, Telnet on port 23, http on port 80, etc.

With the emergence of generalized protocols such as RPC and RMI this becomes a little more complex – the “registry” is run on a standard port – 1099. The client connects to the registry which keeps track of what port a “named service” is actually running.

Further levels of complexity are added by application serving.
First rule of application protocol design is:
Use existing application protocols when possible, e.g. SMTP, SNMP, FTP, HTTP

Make the protocol used to access an application service independent of the service.
- Define the protocol first and develop a service that implements it.
- Think of the possible ways the service might be used

Keep the protocol as simple and stateless as possible.
The Design of a Protocol

- The goal is to define the dialog between client and server.
- Two questions must be answered:
  - When does each utterance end
  - When does the dialog end
- The utterance ending has four variations
  - One shot dialogs can use partial closes
  - Each utterance can be a fixed length
  - Special characters, or delimiters can signal the end
  - Each message can involve a length and value the value is normally a tag with data
- The dialog ending has four main variations
  - One shot i.e. client request, server response, done.
  - Client specified end
  - Server specified end
  - Negotiated end
Design of Client Server Systems
Client and server processes must be autonomous entities with clearly defined boundaries and functions.

Local utilization of resources must be maximized by assigning functionality to the computer best suited to the task.

The server process must be designed to process single or multiple requests concurrently based on the task.
Design Guidelines

- Both the client and server task must be upgradeable to run on more powerful platforms
- Client and server processes must be interfaced such that replacing any one will be transparent to the other
- Distribution of Responsibilities
  - Client responsible for user interface, local data validation, processing logic, presentation
  - Server listens for requests, processes request, executes request and sends results
The server normally consists of three parts:
  - Communications modules
  - Persistent data store modules
  - Business logic modules

Increasingly, the means through which clients and servers communicate has been relegated to standard APIs.

Control of the data, and its integrity has been delegated to DBMSs and access has been standardized by high level SQL interfaces. I.e. OBDC and JDBC.

The server proper has become the glue that holds these pieces together. Sometimes called middleware, this software:
  - manages the communication transport protocol and data exchanges between client and server
  - deals with accessing the DBMS
Division of Responsibilities

- Presentation Logic
  - Input
  - Output

- Processing Logic
  - I/O logic
  - Business Logic
  - Data Management Logic

- Data Manipulation Logic

- storage
Typical Divisions

- Presentation at the client
- I/O processing at the client
- Application logic mostly on client
- Data management mostly on the server
- Data manipulation on the server
Contrasting Divisions

• File Server
  • Client does everything except data manipulation and storage
  • Requires server optimized for I/O and storage
• Database Server
  • Presentation, I/O logic and application logic on the client
  • Data management split between client and server
  • Data manipulation and storage at the server
• Transaction C/S Architecture
  • Presentation at the client
  • Application logic split between client and server
  • Data management, data manipulation and storage at the server
The Server Component
Special Considerations in Server Design

• Because a server must operate over a prolonged period of time, there are special considerations in the design of a server that are relaxed in the design of standalone software. These include:
  • Stability. If a standalone program crashes, the user simply restarts the program. If a server crashes, all clients making use of the server are denied access. Particular care must be taken to gracefully recover from error conditions.
  • State. Failure of a client server communication can be caused by failure of the server, the client, or the connection. Care needs to be taken to insure that a server does not become burdened by failures outside its control. This generally involves making the server “idempotent” or stateless.
  • Security. Especially when servers operate using published protocols, designers must be aware that non-authorized users may endeavor to connect false clients to the server and attempt to exploit design weaknesses to gain unauthorized access to the host system
Stateful and Stateless Servers

- From Imagine that a file server keeps track of a file a user is reading and sends 20 lines every time the user finishes reading a screen.
- An efficiency point of view, it is simplest to have the server open the file when the client connects, read twenty lines, and keep the file pointer at the point of the next read.

<table>
<thead>
<tr>
<th>File Handle</th>
<th>File</th>
<th>location</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Act.dat</td>
<td>20</td>
</tr>
<tr>
<td>6</td>
<td>Stuff.asc</td>
<td>600</td>
</tr>
<tr>
<td>9</td>
<td>Employee.dat</td>
<td>460</td>
</tr>
<tr>
<td>10</td>
<td>Org.tbl</td>
<td>200</td>
</tr>
</tbody>
</table>
Stateful and Stateless Servers (2)

- What happens if client or server crashes?
  - If the client crashes, the server is holding a place in a file and potentially locking other users out
  - If the server crashes, the client loses track of where it was.
- Alternatively, the server can be made idempotent by viewing each request as completely isolated
  - The client makes a request which includes a file name and a starting line number
  - The server sends back the lines requested, the location of the next line, or a –1 indicating that the end of file has been reached.
- The server or client can now crash and restart without loss of data.
Stability

- Stability of servers involves many considerations. Among the more common are:
  - Design of protocols and try catch blocks that allow the server to indicate to the client that the requested operation could not be performed.
  - The use of alarms to capture the emergence of a deadlock condition – i.e. having the server check with itself via an operating system signal if it does not complete an action in a given period of time.
  - The use of file locks to let one process alert another process to the fact that a given file is in use.
  - Copious checking of method return values to insure that invoked methods acted as expected.
Security

- Security is a particular concern in client server computing because servers await connections without knowledge of where the connection is coming from.
- In the development of servers using C, one of the biggest security loopholes was overflowed arrays that allowed hackers to insert code into the running process. While this flaw is largely overcome by Java, other checks must still be made.
- Care must be taken to insure that data being passed to the system via the server conforms to what is expected. For example, if you are passing a command to type or “cat” a file on Unix, you want to make sure that the filename provided by the user is not something like “xyz.dat;rm –rf /*” passed to the operating system, the instruction could result in the file being typed, and all files on the system deleted.
Concurrency in Servers
Real versus apparent concurrency

- Timesharing is apparent concurrency – in reality, there is only one process running at a time, but from the user perspective, they are using the resources concurrently.
  - Parallel processors provide real concurrency.
- A server may be designed to handle one client at a time and to hold other clients in a queue until the clients who arrived first are dealt with. This is called an iterative server.
- A server may also be designed to handle multiple clients simultaneously providing apparent concurrency.
- Concurrent processing is fundamental to client/server computing.
Concurrent Servers

- Access rules permit multiple pairs of processes to communicate using same network resources.
- A server process may duplicate itself, and all its system resources using a “fork” command.
  - This allows one copy of the server – one process, to attend to the task of listening for connections from clients while one or more copies of the process serve the requests of a particular client.
- Server processes must carefully delineate which resources are shared with other copies of the process and which are unique to a particular client.
- Processes must guarantee non-interference between client/server pairs.
- Java uses multiple threads as a less costly way to provide concurrency.
Most programs have a set of variables (data space) in the program.

If multiple processes can execute the same copy of a program, then each must have its own copies of the variables.

```c
for (i=0; i < 10; i++) printf(“ %d\n”,i);
```

Each program must have its own copy of i.
Context Switching

- Concurrency causes context switching to take place via fork, spawn, etc.
- This will add 20% - 50% of execution time overhead
- Threads are much less costly in terms of overhead, but can be more difficult to program in that care must be taken that shared data spaces are not contaminated
The Client Component
Client Design: GUI and IO

- Typical a client reads bytes from TCP/IP port connection and displays data on screen.
- The user may also want to enter commands, etc. on the keyboard, mouse, etc.
- Unfortunately, a program reading from a port will “block” until data arrives to let it complete its I/O. User input ignored. Care needs to be taken in the design of a client to insure that other messages are acknowledged and blocks are avoided.