Socket Programming

Sockets Programming

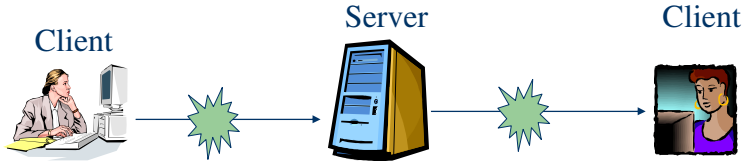
- Client-Server Computing
- What are Sockets
- Sockets Programming in Java
- Programming Examples

Client/Server Computing

- Simple idea:
- Some hosts (clients, typically desk top computers) are specialized to interact with users:
  - Gather input from users
  - Present information to users
- Other hosts (servers) are specialized to manage large data, process that data
- The Web is a good example: Client (Browser) & Server (HTTP server)
- Other examples: FTP, e-mail, …. (in essence, all early Internet "services" were implemented using the client-server paradigm)

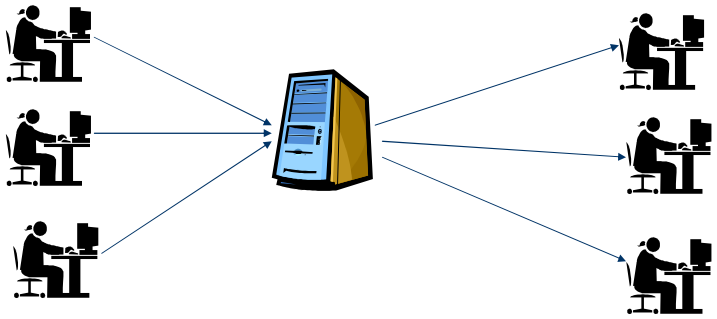
Client/Server Computing

- Other examples:
  - E-mail



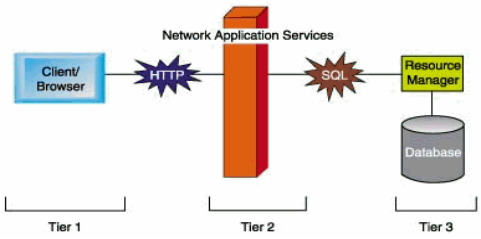
Client/Server Computing

- Other examples:
  - Chatroom



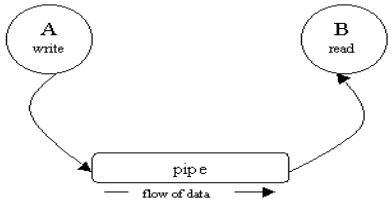
Tiered Client/Server Architecture

- 1-tier: single program
- 2-tier: client/server (e.g. the Web)
- 3-tier: application logic and databases on different servers (e.g. the Web with CGI and databa



Client/Server Communication

- Two related processes on a single machine may communicate through a pipe



- A pipe is a pseudo-file that can be used to connect two processes together

Client/Server Communication

- Two UNRELATED processes may communicate through files (process A write to a file and process B reads from it)
- But HOW two processes located on two different machines communicate? Solution: Berkeley sockets.

What are sockets

- A socket is an end point of a connection
- Or: the interface between user and network
- Two sockets must be connected before they can be used to transfer data (case of TCP)
- A number of connections to choose from:
  - TCP, UDP, Multicast
- Types of Sockets
  - SOCK_STREAM, SOCK_DGRAM, SOCK_RAW

Sockets

- Message destinations are specified as socket addresses
- Each socket address is a communication identifier:
  - Internet address
  - Port number
- The port number is an integer that is needed to distinguish between services running on the same machine
- Port numbers between 0 .. 1023 are reserved

Ports

- Some “well-known” ports:
  - 21: ftp
  - 23: telnet
  - 80: http
  - 161: snmp
- Check out /etc/services file for complete list of ports and services associated to those ports

Which transport protocol (TCP v. UDP)

- TCP -- Transmission Control Protocol
- UDP -- User Datagram Protocol
- What should I use?
  - TCP is a ***reliable*** protocol, UDP is not
  - TCP is ***connection-oriented***, UDP is ***connectionless***
  - TCP incurs overheads, UDP incurs fewer overheads
  - UDP has a size limit of 64k, in TCP no limit

Unix and C specific data structures

- The <netdb.h> library provides the following data structures:

```
struct hostent { // for host info
  char  *h_name;
  char  **h_aliases;
  int   h_addrtype;
  int   h_length;
  char  **h_addr_list;
  #define  h_addr  h_addr_list[0]
};
```

Unix and C specific data structures

```
struct servent { //service info
  char  *s_name;
  char  **s_aliases;
  int   s_port;
  char  *s_proto;
};
```

Unix and C specific data structures

- The following functions return information about a given host or service:

```
struct hostent *gethostbyname (char
  *hostname)
struct servent *getservbyname (char
  *service, char *protocol)
```

UDP Socket Programming

- UDP is simple and efficient, but not reliable
- Communication takes place in a symmetric manner: both ends send and receive messages following an agreed upon protocol

UDP Socket Programming

- Since no connection is created, each message should contain the address of the recipient.
- In Java, messages are contained in instances of class `DatagramPacket`

The DatagramPacket class

- Constructors:
  - One for sending datagrams:

```
DatagramPacket{byte buffer[], int
  length, InetAddress, int port}
```

  - One for receiving datagrams:

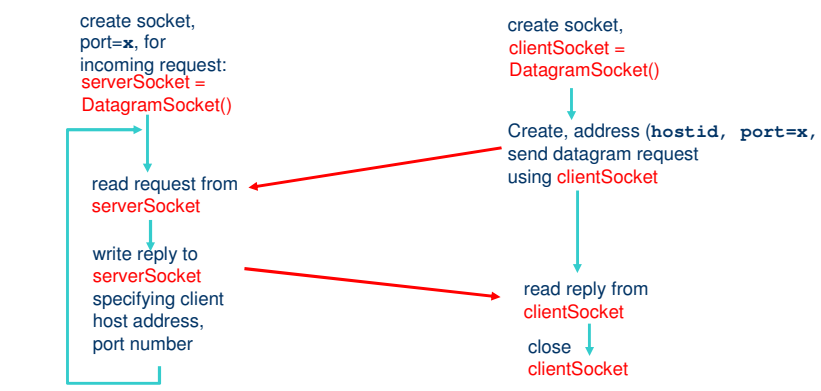
```
DatagramPacket{byte buffer[], int
  length}
```

The DatagramPacket class

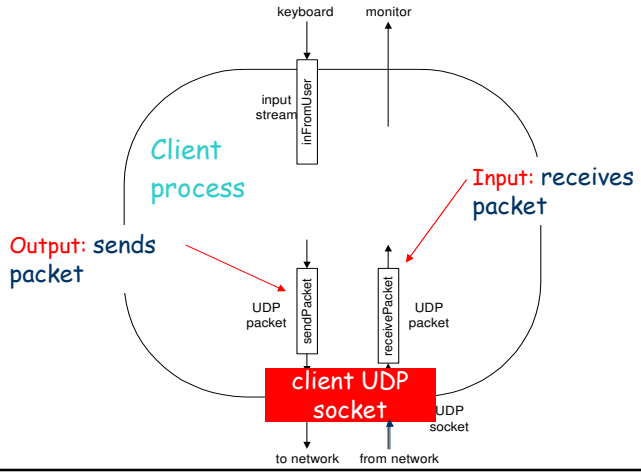
- The useful methods are the accessors:

```
InetAddress getAddress()
Int getPort()
Byte[] getData()
Int getLength()
```

Client/Server Socket Interaction: UDP



Example: Java Client (UDP)



Example: Java Client (UDP)

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

class UDPClient {
   public static void main(String args[]) throws Exception
   {

      BufferedReader inFromUser =
        new BufferedReader(new InputStreamReader(System.in));

      DatagramSocket clientSocket = new DatagramSocket();

      InetAddress IPAddress = InetAddress.getByName("hostname");

      byte[] sendData = new byte[1024];
      byte[] receiveData = new byte[1024];

      String sentence = inFromUser.readLine();
      sendData = sentence.getBytes();
```

Create input stream

Create client socket

Translate hostname to IP address using DNS

Example: Java Client (UDP), cont.

Create datagram with data-to-send, length, IP addr, port

Send datagram to server

Read datagram from server

```
DatagramPacket sendPacket =
  new DatagramPacket(sendData, sendData.length, IPAddress, 9876);

clientSocket.send(sendPacket);

DatagramPacket receivePacket =
   new DatagramPacket(receiveData, receiveData.length);

clientSocket.receive(receivePacket);

String modifiedSentence =
    new String(receivePacket.getData());

System.out.println("FROM SERVER:" + modifiedSentence);
clientSocket.close();
}
}
```

Example: Java Server (UDP)

Create datagram socket at port 9876

Create space for received datagram

Receive datagram

```
import java.io.*;  import java.net.*;
class UDPServer {
  public static void main(String args[]) throws Exception
   {
     DatagramSocket serverSocket = new DatagramSocket(9876);

     byte[] receiveData = new byte[1024];
     byte[] sendData  = new byte[1024];

     while(true)
      {
        DatagramPacket receivePacket =
           new DatagramPacket(receiveData, receiveData.length);
        serverSocket.receive(receivePacket);
```

Example: Java Server (UDP), cont

```
                String sentence = new String(receivePacket.getData());
Get IP addr
port #, of
sender          InetAddress IPAddress = receivePacket.getAddress();
                int port = receivePacket.getPort();

                        String capitalizedSentence = sentence.toUpperCase();

                sendData = capitalizedSentence.getBytes();
Create datagram
to send to client   DatagramPacket sendPacket =
                      new DatagramPacket(sendData, sendData.length, IPAddress,
                                  port);
Write out
datagram
to socket       serverSocket.send(sendPacket);
              }
            }
          }
```

Socket Programming with TCP

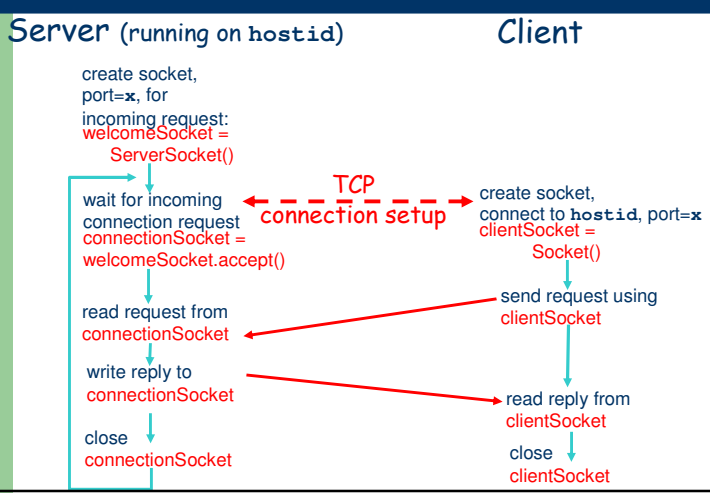
- Client must contact server
- server process must first be running
- server must have created socket (door) that welcomes client's contact
- Client contacts server by:
- creating client-local TCP socket
- specifying IP address, port number of server process
- When client creates socket: client TCP establishes connection to server TCP

- When contacted by client, server TCP creates new socket for server process to communicate with client
  - allows server to talk with multiple clients
  - source port numbers used to distinguish clients (more in Chap 3)

application viewpoint

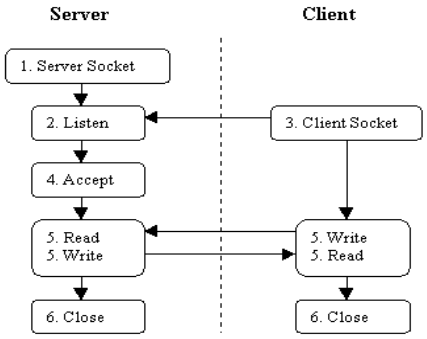
TCP provides reliable, in-order transfer of bytes ("pipe") between client and server

Client/Server Socket Interaction: TCP



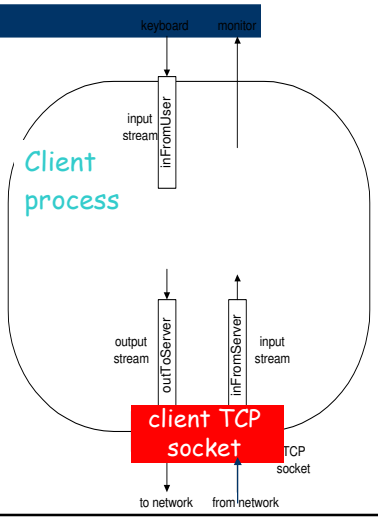
TCP Socket Communication

- Sequence of steps normally taken to set up socket communication and exchange data between C/S



Stream Jargon

- A stream is a sequence of characters that flow into or out of a process.
- An input stream is attached to some input source for the process, e.g., keyboard or socket.
- An output stream is attached to an output source, e.g., monitor or socket.



Socket Programming with TCP

Example client-server app:

1) client reads line from standard input (**`inFromUser`** stream) , sends to server via socket (**`outToServer`** stream)
2) server reads line from socket
3) server converts line to uppercase, sends back to client
4) client reads, prints modified line from socket (**`inFromServer`** stream)

Example: Java Client (TCP)

```
import java.io.*; import java.net.*;
class TCPClient {

   public static void main(String argv[]) throws Exception
   {
      String sentence;
      String modifiedSentence;

      BufferedReader inFromUser =
        new BufferedReader(new InputStreamReader(System.in));

      Socket clientSocket = new Socket("hostname", 6789);

      DataOutputStream outToServer =
        new DataOutputStream(clientSocket.getOutputStream());
```

Create input stream

Create client socket, connect to server

Create output stream attached to socket

Example: Java Client (TCP), cont.

```
      BufferedReader inFromServer =
        new BufferedReader(new
        InputStreamReader(clientSocket.getInputStream()));

      sentence = inFromUser.readLine();

      outToServer.writeBytes(sentence + '\n');

      modifiedSentence = inFromServer.readLine();

      System.out.println("FROM SERVER: " + modifiedSentence);

      clientSocket.close();

   }
}
```

Create input stream attached to socket

Send line to server

Read line from server

Example: Java Server (TCP)

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

class TCPServer {

  public static void main(String argv[]) throws Exception
   {
     String clientSentence;
     String capitalizedSentence;

     ServerSocket welcomeSocket = new ServerSocket(6789);

     while(true) {

       Socket connectionSocket = welcomeSocket.accept();

      BufferedReader inFromClient =
        new BufferedReader(new
        InputStreamReader(connectionSocket.getInputStream()));
```

Create welcoming socket at port 6789

Wait, on welcoming socket for contact by client

Create input stream, attached to socket

Example: Java Server (TCP), cont

```
      DataOutputStream  outToClient =
        new DataOutputStream(connectionSocket.getOutputStream());

      clientSentence = inFromClient.readLine();

      capitalizedSentence = clientSentence.toUpperCase() + '\n';

      outToClient.writeBytes(capitalizedSentence);
     }
   }
}
```

Create output stream, attached to socket

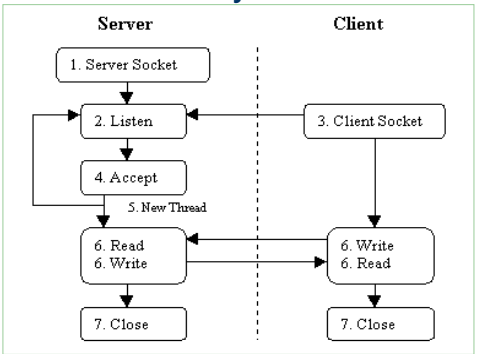
Read in line from socket

Write out line to socket

End of while loop, loop back and wait for another client connection

Multi-threaded Servers

- A server should be able to serve multiple clients simultaneously



Multi-threaded Servers

- See the MTEchoServer example

WWW References

- **SPENCER'S SOCKET SITE:** http://www.lowtek.com/sockets/ (includes tutorials and C code/example programs)
- Java Socket Tutorial: http://java.sun.com/docs/books/tutorial/networking/sockets/
- Course Website: the original Intro and Advanced Socket tutorials, written for BSD 4.4, which introduced the *socket* abstraction

Sockets Programming in Java: TCP

- Streams
  - The basic of all I/O in Java is the data stream
  - A pipeline of data
    - put info into the pipeline (write) and get it (read)
- Programming with Sockets (TCP)
  - Opening a Socket
  - Creating a data input stream
  - Creating a data output stream
  - Closing the socket(s)