Multithreaded Web Servers

- Network Protocols
- Socket Programming
- Multithreaded Web Server

What’s a protocol?

human protocols:
- “what’s the time?”
- “I have a question”
- introductions

… specific msgs sent
… specific actions taken when msgs received, or other events

network protocols:
- machines rather than humans
- all communication activity in Internet governed by protocols

protocols define format, order of msgs sent and received among network entities, and actions taken on msg transmission, receipt
What’s a protocol?

A human protocol and a computer network protocol:

Protocol “Layers”

Networks are complex!

- many “pieces”:
  - hosts
  - routers
  - links of various media
  - applications
  - protocols
  - hardware, software

Question:
Is there any hope of organizing structure of network?
Organization of air travel

- a series of steps

Organization of air travel: a different view

Layers: each layer implements a service
- via its own internal-layer actions
- relying on services provided by layer below
Layered air travel: services

- Counter-to-counter delivery of person+bags
- Baggage-claim-to-baggage-claim delivery
- People transfer: loading gate to arrival gate
- Runway-to-runway delivery of plane
- Airplane routing from source to destination

Distributed implementation of layer functionality

- **Departing airport**
  - Ticket (purchase)
  - Baggage (check)
  - Gates (load)
  - Runway takeoff
  - Airplane routing

- **Intermediate air traffic sites**
  - Airplane routing
  - Airplane routing

- **Arriving airport**
  - Ticket (complain)
  - Baggage (claim)
  - Gates (unload)
  - Runway landing
  - Airplane routing

© C Xu, 1998, 2002
Why layering?

Dealing with complex systems:

- explicit structure allows identification, relationship of complex system’s pieces
  - layered reference model for discussion
- modularization eases maintenance, updating of system
  - change of implementation of layer’s service transparent to rest of system
  - e.g., change in gate procedure doesn’t affect rest of system
- layering considered harmful?

Internet protocol stack

- application: supporting network applications
  - FTP, SMTP, STTP
- transport: host-host data transfer
  - TCP, UDP
- network: routing of datagrams from source to destination
  - IP, routing protocols
- link: data transfer between neighboring network elements
  - PPP, Ethernet
- physical: bits “on the wire”
Network Layer

application layer
transport layer
network layer
data link layer

IP Addressing (IPv4)

- 32-bit address for each host: 4 Billion
- two-level address: (network, host)
- dotted-decimal notation (141.217.14.5)
- three address formats

<table>
<thead>
<tr>
<th>Class</th>
<th>Network</th>
<th>Host</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>24</td>
</tr>
<tr>
<td></td>
<td>14</td>
<td>16</td>
</tr>
<tr>
<td>B</td>
<td>10</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>21</td>
<td>8</td>
</tr>
<tr>
<td>C</td>
<td>110</td>
<td>8</td>
</tr>
</tbody>
</table>

© C. Xu, 1998,2002
Host Names and DNS

◆ Domain Naming System (DNS) provides a mapping from human-readable names (ece.eng.wayne.edu) to IP addresses (141.217.14.5)

◆ benefits
  – easier to remember
  – level of indirection allows reconfiguration without telling clients

DNS Naming

◆ top-level servers know IP addresses of name-servers for top-level domains

◆ lookup process
  – ask top-level server for IP address for desired name
  – result is either
    » the answer
    » referral to name-server for a more specific domain
  – iterate until you have the answer
Administrative Details

◆ to allocate a DNS domain name
  – contact NetSolution
  – pay them a fee ($35)
  – they check that you have the necessary servers running
  – they tell top-level servers that you’re there
◆ similar for IP network number allocation

Transport Layer
TCP: Transmission Control Prot.

- Reliable, connection-oriented, bidirectional, in-order byte stream
  - acknowledgement, timeout, and retransmission
  - flow control: slows down sender so receiver isn’t overwhelmed
  - congestion control: slows down sender so network isn’t overwhelmed

UDP (User Datagram Protocol)

- unreliable, unordered datagrams
- simple demultiplexer
  - IP sends datagram to a machine
  - UDP sends datagram to (machine, port) pair
    » port is 16-bit identifier
- how a client learns the server’s port

<table>
<thead>
<tr>
<th></th>
<th>src port</th>
<th>dest port</th>
</tr>
</thead>
<tbody>
<tr>
<td>checksum</td>
<td></td>
<td>length</td>
</tr>
</tbody>
</table>

data follows
Well-Known Ports

- many standard services live at the same port on almost all machines
  - http on port 80
  - telnet on port 23
  - ftp on port 21
  - smtp (email server) on port 25
- port mapping: /etc/services

TCP/UDP Programming in Java
Using TCP in Java

- classes in java.net package
- Socket interface to transport protocols
  - originated in BSD Unix
  - class Socket uses connection-oriented TCP
  - class DatagramSocket uses connectionless UDP
  - class MulticastSocket used for multicast

What is a Socket

- A special type of file, provided by Berkeley Unix, representing an end of comm.
- Under the wrap of sockets, a network connection can be written to or read from as a regular byte stream.
Operations on a socket:

- connect to a remote machine
- send and receive data
- close a connection
- Bind to a port
- listen for incoming data
- accept connections from remote machines on the bound port

Berkeley Sockets

- Socket primitives for TCP/IP.

<table>
<thead>
<tr>
<th>Primitive</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Socket</td>
<td>Create a new communication endpoint</td>
</tr>
<tr>
<td>Bind</td>
<td>Attach a local address to a socket</td>
</tr>
<tr>
<td>Listen</td>
<td>Announce willingness to accept connections</td>
</tr>
<tr>
<td>Accept</td>
<td>Block caller until a connection request arrives</td>
</tr>
<tr>
<td>Connect</td>
<td>Actively attempt to establish a connection</td>
</tr>
<tr>
<td>Send</td>
<td>Send some data over the connection</td>
</tr>
<tr>
<td>Receive</td>
<td>Receive some data over the connection</td>
</tr>
<tr>
<td>Close</td>
<td>Release the connection</td>
</tr>
</tbody>
</table>

© C Xu, 1998,2002
Berkeley Sockets (Cont’)

- Connection-oriented communication pattern using sockets.

Socket programming in Java

- *Socket* class has methods for operations in both client and server sides
  - connect/close, send/receive
- *ServerSocket* class provides for *bind, listen, and accept*
- on client side: create a Socket, connect to a remote host, access, and close
- on server side: create a ServerSocket, keep listening, accepting, and talking.
Client side

```java
try {
    Socket server = new Socket("foo.bar.com", 1234);
    // (DNS name, port)
    InputStream in = server.getInputStream();
    OutputStream out = server.getOutputStream();
    out.write(42); // write a byte

    // Read a new line or CR delimited string
    BufferedReader bin = new BufferedReader(new InputStreamReader(in));
    String response = bin.readLine();
    Byte back = in.read();

    server.close();
} catch (IOException e) {} // handle exceptions
```

Server side

```java
Try {
    ServerSocket listener = new ServerSocket(1234);

    while (!finished) {
        Socket client = listener.accept(); // wait for connection
        InputStream in = client.getInputStream();
        OutputStream out = client.getOutputStream();

        Byte someByte = in.read(); // read a byte
        BufferedReader bin = new BufferedReader(new InputStreamReader(in));
        String someString = bin.readLine();
        out.write(43); // write a byte
        client.close();
    }
    listener.close();
} catch (IOException e) {} // handle exceptions
```
ServerSocket constructors

- ServerSocket(int port)
- ServerSocket(int port, int queuelen)
- ServerSocket(int port, int queuelen, InetAddress bindAddr)

```java
try {
    ServerSocket httpd = new ServerSocket(5600, 100,
                                          java.net.InetAddress.getHostName("ece.eng.wayne.edu");
} catch (IOException e) {} 
```

User-Defined Socket and ServerSocket

```java
class SSLServerSocket extends ServerSocket {
    ...
    public Socket accept() throws IOException {
        SSLSocket s = new SSLSocket(…); { }
    }
    ...
}
class SSLSocket extends java.net.Socket {
    ...
    public SSLSocket(…) {
        super();
        ...
    }
    ...
}
```
A Tiny Multithreaded Web Server

// Support HTTP protocol: GET /path/filename
// java.TinyHttpd 1234
import java.net.*;
import java.io.*;
import java.util.*;

public class TinyHttpd {
    public static void main( String argv[] ) throws IOException {
        ServerSocket ss = new ServerSocket(
            Integer.parseInt( argv[0] ));
        while ( true )
            new TinyHttpdConnection( ss.accept() ).start();
    }
}

class TinyHttpdConnection extends Thread {
    Socket client;
    TinyHttpdConnection ( Socket client) throws SocketException {
        this.client = client;
        setPriority( NORM_PRIORITY -1 );
    }
    public void run() {
        try {
            BufferedReader in = new BufferedReader(
                new InputStreamReader(
                    client.getInputStream(), "8859_1" );
            OutputStream out = client.getOutputStream();
            PrintWriter pout = new PrintWriter(
                new OutputStreamWriter(out , "8859_1"), true );
            String request = in.readLine();
            System.out.println( "Request "+ request );
        }
    }
}

© C Xu, 1998,2002
public void run() {
    try { … …
        StringTokenizer st = new StringTokenizer( request );
        if ( (st.countToken() >=2 ) && st.nextToken().equals("Get") ) {
            if ( (request = st.nextToken()).startsWith("/")
                request = request.substring( 1 );
            if ( request.endsWith("/") || request.equals("")
                request = request + “index.html”;
            try {
                FileInputStream fis = new FileInputStream( request );
                byte []data = new byte[ fis.available() ];
                fis.read( data );
                out.write( data );
                out.flush();
            } catch (FileNotFoundException e) { … } else { … }
            client.close();
        } catch ( IOException e) { … }
    } catch (FileNotFoundException e) { … }

    Security Manager (Java 1.1)
    class TinyHttpdSecutityManager extends SecurityManager {
        public void checkAccess( Thread g) {};
        public void checkListen( int port ) {};
        public void checkLink( String lib ) {};
        public void checkPropertyAccess (String key) {};
        public void checkAccept( String host, int port ) {};
        public void checkWrite( FileDescriptor fd) {};
        public void checkRead( FileDescriptor fd ) ;

        public void checkRead( String s) {
            if (new File(s).isAbsolute() || s.indexOf(“.”)!=-1 )
                throw new SecurityException(“Access to file”+s+”denied”);
        }
    }

    © C Xu, 1998,2002
Secured web server

// HTTP protocol: `GET /path/filename [options]`
// `java.TinyHttpd`

```java
public class TinyHttpd {
    public static void main( String argv[] ) throws IOException {
        System.setSecurityManager( new TinyHttpdSecurityManager() );
        ServerSocket ss = new ServerSocket( Integer.parseInt( argv[0] ) );
        while ( true )
            new TinyHttpdConnection( ss.accept() ).start();
    }
}
```

```java
public void run() {
    try {
        ....
        StringTokenizer st = new StringTokenizer( request );
        if ( (st.countToken() >=2 ) && st.nextToken().equals("Get") ) { ....
            try {
                FileInputStream fis = new FileInputStream( request );
                byte []data = new byte[ fis.available() ];
                fis.read( data );     out.write( data );          out.flush();
            } catch (FileNotFoundException e) { ... }
            catch (SecurityException se ) {... }
            else { ... }
            client.close();
        } catch ( IOException e) { ... }
    }
```

© C Xu, 1998,2002
Datagram Sockets

DatagramPacket + DatagramSocket

Client side:
void sendMsg( String msg ) {
    try {
        byte[] data = msg.getBytes();
        DatagramPacket pack = new DatagramPacket(
            data, data.length, destInetAddr, port);
        DatagramSocket ds = new DatagramSocket();
        // Bound to an anonymous port
        ds.send( pack );
        ds.close();
    } catch (IOException e) {} 
}

Server side
// java dgserver 1234
public static void main( String [] argv ) throws IOException {
    try {
        DatagramSocket ds =
            new DatagramSocket( Integer.parseInt(argv[0]) );
        while (true) {
            byte[] buf = new byte[1024];
            DatagramPacket pack =
                new DatagramPacket(buf, buf.length);
            ds.receive( pack );
            String msg = new String( pack.getData() );
            // pack.getAddress().getHostName() ;
        }
    } catch (SocketException se) {} 
    // SocketException: the port is less than 1024, or
    // the port is in use