Objectives

- Web Server
- Socket programming in Java
- Project 1

Open up a terminal

Administration

- Background on Web Server project
  - Abstracting away networking
- Back to regularly scheduled program on Wednesday
  - Wed: Distributed Systems challenges, networking
  - Fri: threads/synchronization, End to end argument

Perusall

- Access through Sakai
  - No need for new account

What is a Web Server?

- What does it do?
- What are the parties involved?
- Know any web server software/applications?

What is a Web Server?

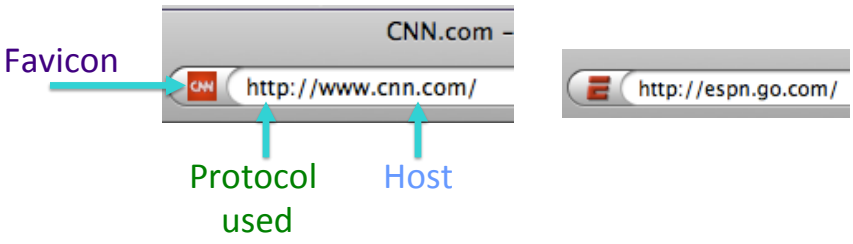
- What does it do?
  - Serves requested files to user
- What are the parties involved?
  - Browser (client), HTTP (communication), Web server, HTML documents
- Know any web server software/applications?
  - Apache, Microsoft IIS

Clients and Servers

- Server tasks
  - Listen, accept, receive, send, loop
- Client tasks
  - Connect, request, receive, close

How Does The Browser Get a Page?

- In Web browser, enter a URL
  - URL: Uniform Resource Locator



  - May not have explicitly typed in "`http`"
    - Default protocol
    - Other protocols: https, ftp

How Does The Browser Get a Page?

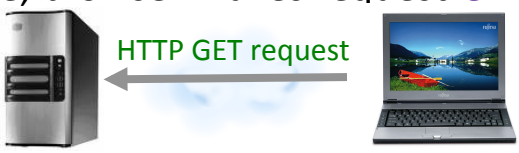
- Look up Host's IP Address using DNS
  - Need to be able to "find" host on the Internet
  - Routing through Internet is by IP address
- Domain Name System (DNS)
  - Set of servers that map domain name to IP Address(es) and vice versa

`www.espn.com` ⟷ `54.149.104.165`

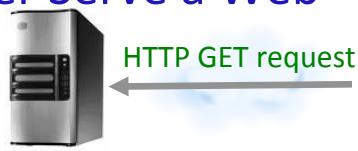
- Unix commands **`host`** and **`nslookup`** can lookup this information

How Does The Browser Get a Page?

- Browser now makes the request using HTTP
  - HTTP: HyperText Transfer Protocol
- Common Types of HTTP Requests:
  - GET: download a page
  - POST: download a page
  - HEAD: just get the "header" for a page
- For our example, browser makes request GET /



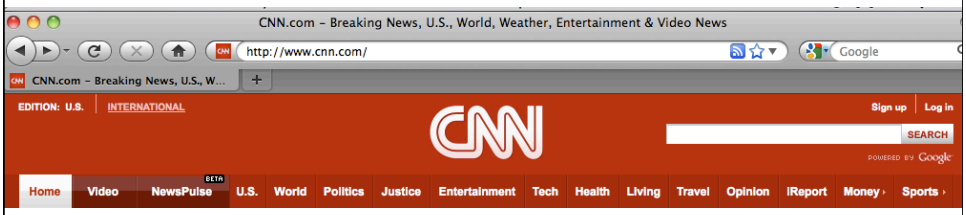
How Does the Web Server Serve a Web Page?



- Receives request for a resource on TCP port 80
- Looks for the resource in the Web Document directory
  - Not all files on a Web server are meant for others to see
  - Specific directory holds these files
- If the file is found, server sends an HTTP 200 response with the requested document
  - Otherwise, sends appropriate error response

How Does Browser Get a Page?

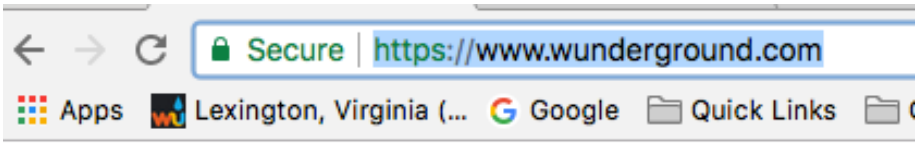
- Receives response from server
- Renders file in appropriate format



HTTP Status Codes

Code	Meaning
200	OK: Request succeeded
3xx	Redirection (temporary or permanent)
403	Error: No permission
404	Error: File not found
500	Internal server error

Uh oh!




```
404 page not found
```


More on URLs

- Specifies the location of a resource
- Format: <protocol>://<host>/<path>
  - Examples:

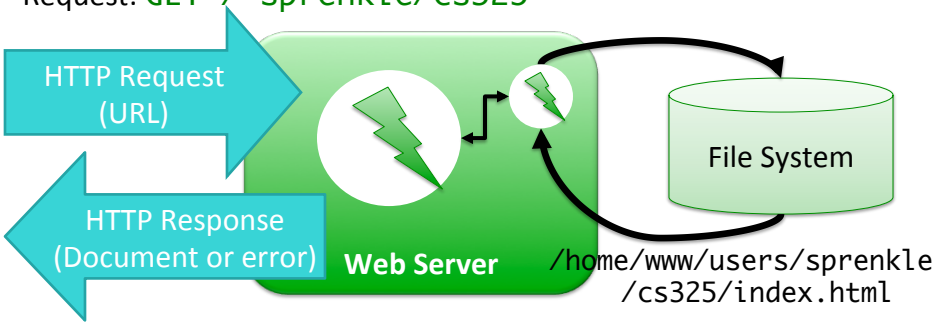
```
http://www.cs.wlu.edu/~sprenkle/cs325/

http://www.cs.wlu.edu/~sprenkle/cs325/
  schedule.php
```

Web Server

- Server tasks:
  Listen, accept, send, receive, loop

Request: `GET /~sprenkle/cs325`



- Implement your own web server
- Teams of 2-3

Network Programming and Java

- Java abstracts details of underlying network and how OS interacts with it
  - Hidden and encapsulated in the `java.net` package
  - Makes network programming easier

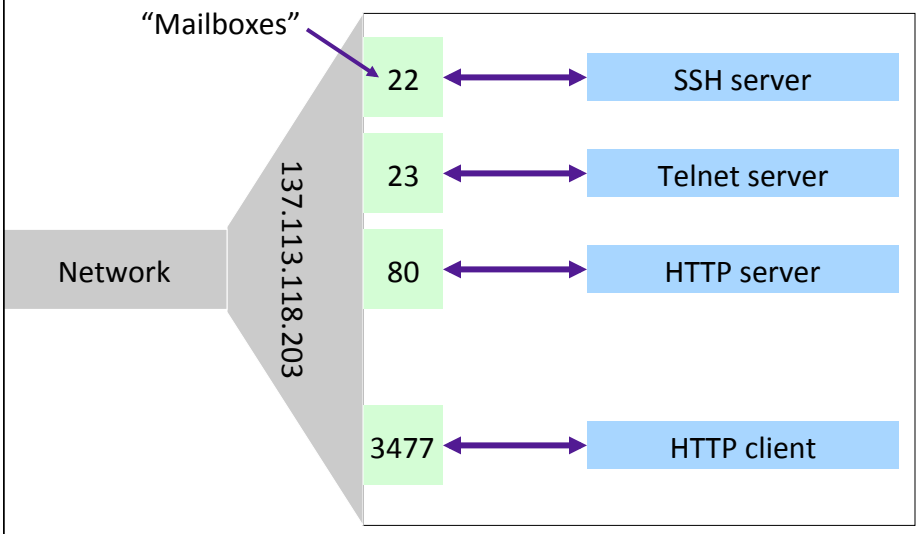
Network Addresses

- A computer or host on a network has an address
  - Uniquely identifies computer on the network
- Most common address system in use is the Internet Protocol (IPv4) addressing system
  - 32-bit address
  - Typically written as "dotted-quad"
    - four numbers, 0 through 254, separated by periods, e.g., 137.113.118.200
  - Final exhaustion occurred on February 3, 2011
  - June 8, 2011 World IPv6 Day, a global 24-hour test of IPv6

Ports

- Each host on the network has a set of ports
  - Ports are like mailboxes:
    - Address specifies the host
    - Port specifies application on host
  - Ports range from 1 to 65535
- Allow multiple applications to use a network interface/ address to communicate over network
- Examples:
  - A web server communicates on network using port 80
  - An ssh server on the same host will have the same address but use port 22

A Machine's Ports



Well-Known Ports

- Port numbers < 1024 are well-known ports
  - Assigned to *application servers*
  - Port 80 always has an HTTP server
  - Port 22 always has an SSH server
- Client listens on another port (above 1024) to receive responses from a server
- No technical reason servers must conform to these standards
  - Convention so that clients know where to find web server, FTP server, …
  - Can have an HTTP server at a port > 1024

Sockets

- A socket is an abstraction of an endpoint of a two-way communications link
- An application creates a socket that is bound to a remote address and remote port
  - Port on the host (client) could be random
- A *connection* is created between the client using this port and the specified remote address at the remote port

Services provided by networks

- Connection-oriented
- Connection-less

Connection-Oriented Service

- A connection-oriented service is like the telephone system
  - Acts like a pipe

  - Sender pushes bits into pipe and then come out of the receiver in same condition as they were pushed in
  - Pipe is connected to a port on the sender and a port on the receiver
- Implemented in Java using stream sockets

Stream Sockets

- Creates a pipe that connects endpoints and provides a *reliable byte stream*
- Communicates using TCP
  - Hides details of TCP from programmer
  - TCP: Most popular protocol that implements a stream, or connection-oriented, service
  - Reliable service: when data is sent from one end to the other, arrives in order, in the same state, and is not lost or duplicated in the network

Connectionless Service

- A connectionless service is like the postal system
  - One side sends messages to the other side
  - Each message is independent
  - Can lose messages in the network, duplicate messages, corrupt data during transport
    - An unreliable service
  - One side creates a message and sends it to the other side
- Implemented in Java using *datagram sockets*

Java's `DatagramSockets`

- User Datagram Protocol (UDP)
  - Popular protocol that Java uses to implement datagram sockets
- No connection between sockets
  - A socket is opened to another socket but no connection is actually made
  - When a message is passed to socket, it is sent over network to other socket
    - Most of the time it gets there

Example Java Client Program

1. Connect to a server (another host on the network)
2. Open a stream to a certain port
3. Display what the server sends

What Does This Code Do?

```
public class SocketTest {

    public static void main(String argv[]) {
        try {
            Socket s = new Socket("time-d.nist.gov", 13);
            BufferedReader in = new BufferedReader(
                new InputStreamReader(s.getInputStream()));

            String line = null;
            while ((line = in.readLine()) != null) {
                System.out.println(line);
            }
        } catch (IOException exc) {
            System.out.println("Error:" + exc);
        }
    }
}
```

Review: What's the difference between Readers and Streams?

Reading from a Socket

```
Socket s = new Socket("time-d.nist.gov", 13);
BufferedReader in = new BufferedReader(
                new InputStreamReader(s.getInputStream()));
String line = null;
while ((line = in.readLine()) != null) {
   System.out.println(line);
}
```

- Creates a socket that connects to host with specified name at port 13
- `getInputStream()` gets a byte stream that reads from the socket
- `InputStreamReader` wraps the byte stream and a `BufferedReader` wraps the `InputStreamReader`
- `BufferedReader` reads all characters sent by the server using `readLine()` and displays each line to `System.out`

Network I/O and Exceptions

- Networking code is inside of a `try` block
- A number of things can go wrong with network communications
  - A power failure knocks out an intermediate router or switch
  - A misconfiguration
  - Someone tripping over a cable
- If any of these errors are detected, an `IOException` is generated
- Any program performing network communication should handle such exceptions

Host Names and IP Addresses

- A host name is passed into the `Socket` constructor
  - Not an IP address (if desired, pass in an `InetAddress` object)
- Java uses the Domain Name Service (DNS) to resolve the host name into an IP address

Host Names and IP Addresses

- Alternative: use constructor with `InetAddress`
  - No `InetAddress` constructor
- `InetAddress`'s static method, `getByName`()

```
InetAddress addr =
      InetAddress.getByName("www.cs.wlu.edu");
```

returns an `InetAddress` object that encapsulates the sequence of four bytes `137.113.118.203`

Multiple IP Addresses per Host

- A host can have > 1 IP address
  - Facilitate load-balancing
  - Example: `www.espn.com` corresponds to 8 IP addresses
    - One can be picked at random whenever host is accessed (usually just the first)
- To determine all of the IP addresses of a specific host, call `getAllByName()`…

```
InetAddress[] addresses = InetAddress.getAllByName(
                                "www.cnn.com");
```

Returns the IPv4 and IPv6 address

The Loopback Address and localhost

- Hostname `localhost` represents the local host
- `localhost` corresponds to the IP address 127.0.0.1, which is known as the loopback address
  - A special IP address that means "the computer connected right here"

Determining the Local Address

- If program calls `getByName("localhost")`, returns IP address 127.0.0.1
- To get the actual IP address of the host, call `getLocalHost()`
  - Returns actual IP address of the host on the network
- Example:

```
InetAddress address =
      InetAddress.getLocalHost();
```

InetAddressTest.java

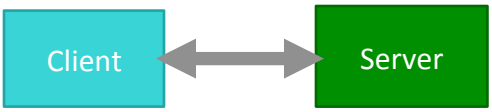
Web Server: Processing Requests

- Receives GET/POST requests from users
- Processes requests
  - Given to appropriate application to handle
    - PHP, ASP, Java Servlet Container, …
  - Handles static requests by sending document to requestor
    - Or appropriate error message if the file does not exist or the file does not have appropriate read permissions

A Web Server: Handling Requests

- Has one thread per client to handle request
  - Limit on number of threads, as discussed
- Serves files from some directory
  - My web-accessible files are in `/home/www/users/sprenkle`
  - But users access with resource name `~sprenkle`
  - Server maintains mapping from `~sprenkle` to appropriate location
  - (Called `DocumentRoot` in Apache)

HTTP Protocol



- Client request:

```
GET /index.html HTTP/1.0 \n
<optional body, multiple lines> \n
\n
```

Don't type \n, just use carriage return

- Server
  - Parses request
    - Request type, resource, protocol
  - Responds to request

HTTP Protocol: Server Response

- Initial response line (status line)

```
HTTP/1.0 200 OK
HTTP/1.0 404 Not Found
```

- Header lines

Can also be found in request

  - Information about response or about object sent in message body

- Requested document
- Example:

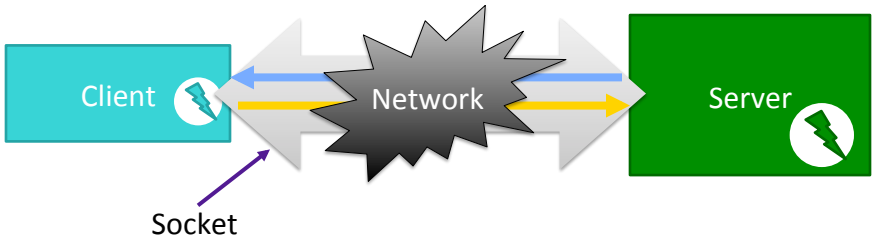
```
HTTP/1.1 200 OK
Date: Wed, 06 Sep 2017 23:44:26 GMT
Server: Apache/2.4.6 (Red Hat Enterprise Linux)
PHP/5.5.38
…

<html> <body> (file contents) . . .
</body>
</html>
```


Status Codes

Code	Meaning
200	OK: Request succeeded
3xx	Redirection (temporary or permanent)
400	Error: Bad Request. Could not be understood by server b/c malformed syntax
403	Error: No permission
404	Error: File not found
500	Internal server error

The Socket Abstraction



- Often, client wants to send data to server as well as receive data from the server
- Sockets are *bi*-directional
- Each end of socket has input/output
  - Need to open an output stream on the socket

Program opens both input and output streams on the same socket – to both read from and write to the server.

```
public class ClientBiDirectionalSocketTest {

    public static void main(String[] args) {
        try {
            Socket s = new Socket("time-d.nist.gov", 13);
            BufferedReader in = new BufferedReader(new InputStreamReader(
                                    s.getInputStream()));
            PrintWriter out = new PrintWriter(s.getOutputStream(),
                                    true); // auto-flush
            // do stuff
        } catch (IOException exp) {
            System.out.println("Error:" + exp);
        }
    }
}
```

Clients and Servers

- Client opens a connection to a host (the server) at a certain address at a certain port
- The server on remote host **must** be ***listening*** to that port and waiting for a client to connect to that port
- After client connects, server obtains a socket that is an abstraction of its end of the stream, connected to the client

How do we implement the server?

The ServerSocket Class

- Create a ServerSocket object by specifying the port number to listen to …

```
ServerSocket server = new ServerSocket(1999);
```

  - Creates a server socket on port 1999
    - Not a well-known port number because it is > 1024
  - ServerSocket object listens for connection requests on this port

Accepting a Connection

```
// will block until a client connects
Socket incoming = server.accept();
```

- Server waits for a client request to connect on that port by calling `accept()`
  - `accept:` blocking method that waits indefinitely until a client connects to the port
- When client connects, `accept()` returns a `Socket` object
  - That Socket is how the server communicates with the client…

Example: An Echo Server

- Specification: When a client connects, server reads a line from the client and then returns a line identical to what it has received
  - As an added twist, have server echo back what it receives in all capital letters
- Known as an echo server because it echoes back what it receives from the client

What Do We Do From Here?

```
public class CapsEchoServer {

    public static void main(String[] args) {
        try {
                ServerSocket server = new ServerSocket(1999);
                Socket incoming = server.accept();

        } catch (IOException exc) {
                System.out.println("Error:" + exc);
        }
    }
}
```

Any issues we'll need to handle?

What Do We Do From Here?

```
public class CapsEchoServer {

    public static void main(String[] args) {
        try {
                ServerSocket server = new ServerSocket(1999);
                Socket incoming = server.accept();
                // get incoming stream
                // get outstream
                // read from input, uppercase, send to output
        } catch (IOException exc) {
                System.out.println("Error:" + exc);
        }
    }
}
```

Issue to handle:
Something that says that client is done?

```
public static void main(String[] args) {
    try {
        ServerSocket server = new ServerSocket(1999);
        // will block until a client connects
        Socket incoming = server.accept();
        BufferedReader in = new BufferedReader(new InputStreamReader(
                        incoming.getInputStream()));
        PrintWriter out = new PrintWriter(
                incoming.getOutputStream(), true);
        out.println("Echo Server. Type BYE to exit");
        String line = null;
        while ((line = in.readLine()) != null) {
            if (line.trim().equals("BYE"))
                break;
            else
                out.println("Echo:" + line.trim());
        }
        incoming.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```


ServerSocket Summary

- Purpose of a ServerSocket is to wait for connections
- When a client connects, the server generates a new Socket object, which is the server's endpoint of the connection, and returns the socket from the call to accept()

Servers and Multiple Clients

- Servers should handle multiple concurrent clients
- If server only allowed 1 client to connect at any given time, a client can monopolize the service by remaining connected to the server for a long time

What does this sound like a job for?

Clients and Servers

- Question: How do we service other clients?
  - We don't want to consume the server's resources with just one client…
- Three choices
  - Multiple threads 
  - Multiple processes
  - Event queue

Multiple Threads Approach

- When server returns from `accept()` with the `Socket`, start a new thread to handle the new connection
- Main server thread can go back and call `accept()` again, waiting for a new client to connect

A Multithreaded Server

```
while (true) {
   Socket incoming = server.accept();
   Thread clientThread = new ThreadedEchoHandler(incoming);
   clientThread.start();
}
```

- User-defined `ThreadedEchoHandler` class derives from `Thread`
- Client communication loop is its `run()` method…

MultiThreadedServer.java
ThreadedEchoHandler.java

A Multithreaded Server Summary

- Each connection starts a new thread
  - Multiple clients can connect to the server at the same time
- As soon as a client connects, `accept`() returns a `Socket` that encapsulates this new connection
  - `socket` is passed into new thread to handle connection
  - The thread is then started and deals with the connection from then on
- The main thread goes back to waiting for a new connection

Multithreaded Server Issues

- Any problems with having a thread handle each incoming request?

Multithreaded Server Issues

- Any problems with having a thread handle each incoming request?
  - Performance

Multithreaded Server Issues

- For each request, must create a thread
  - Overhead in creating threads
- What happens if receive too many client requests and have to start/fork too many threads?
  - Machine runs out of memory
  - Machine gets bogged down
  - Threads can't make progress

Multi-threaded Server Solutions

Solution: limit # of incoming connections/threads available

- `ServerSocket( int port, int backlog )`
  - Maximum length of queue
  - After `backlog` requests, additional requests are refused
- Create a thread pool
  - Create available threads at startup
  - Get one of these threads when to handle requests
  - See `java.util.concurrent.Executors`

Socket Timeouts

- Reading from a socket indefinitely is a bad idea
  - Network could go down, causing program to wait on socket forever
- Java supports a timeout value
  - If program has been waiting for socket for specified timeout interval, an `Exception` is generated
  - Call `setSoTimeout()` on the socket, in ms

```
Socket socket = new Socket("host", 1998);
socket.setSoTimeout(10000); // 10 second timeout
```

Example Code

- On the course web site
- Few comments
  - Encourage you to read code and figure out what it is doing

Summary: Implementing a Server

- How do we create network connections?
  - Sockets!
  - Java uses `ServerSockets` and `Sockets` for clients
  - (C/C++ makes no distinction between client and server connections)
- How does the server support multiple clients at once?
  - Using multiple threads or processes
  - Using an event queue

Helpful Hints

- Check out the links on the assignment page
- Start small and test often
  - Small implementation, small tests
- Use telnet or nc for preliminary testing and experimentation
- Use browser:
  - http://localhost:8888/test.html

Things to Watch Out For

- Sockets/ports are "already in use"
  - Check if the process is still running
  - Just pick a new port for a few minutes…
- Leaving ports open indefinitely
  - This is really bad!

WRITE UPS

Why writeups at all?

- Important skill
  - Present and make others interested in what you're doing!
  - Organize your thoughts
    - May reveal issues, gaps in knowledge
- Verification of your understanding
- Reading good papers → examples for good writing
  - Learn from the bad examples too
- Practice, practice, practice!

Content of Write Up

- Introduction
  - Motivation, goals, challenges
- Approach
  - Architecture
  - Implementation
- Evaluation (if necessary)
- Discussion
  - Problems, challenges, future work
- Conclusions

Common Issues

- Not presenting the high-level problems and challenges
- Not using correct grammar, spell check

TODO

- Start on Web Server project
  - Read through project
  - Make a team (2-3)
  - Look at Socket examples
- Read E2E Argument paper for Friday
  - Skim through once: review section headings
  - 3 hours max
  - Annotations – in Perusall
  - Friday: Discuss paper and questions