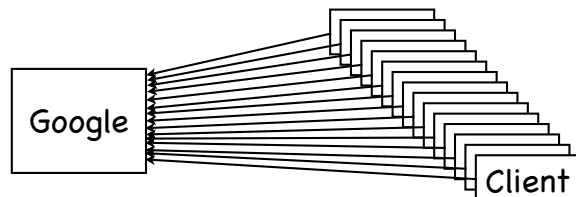


Multi-Threaded Servers

December 6, 2007

1

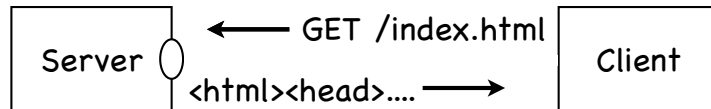
Client-Server Communication



- ◉ Many clients; 1 server
- ◉ Server starts and then waits for clients to connect
- ◉ Client initiates communication
- ◉ Server must handle client requests concurrently
- ◉ Server must not confuse client requests

2

Sockets



1. Server creates ServerSocket at a port and waits
2. Client creates Socket to connect to Server
3. Client writes request on Socket
4. Server reads request from socket
5. Server computes response
6. Server writes response on socket
7. Client reads response and processes it

3

What Server Does

1. Initialize the application
2. Listen on a port by constructing a "server socket"

```
ServerSocket ss = new ServerSocket (port)
```
3. Wait for a client to connect

```
Socket s = ss.accept();
```
4. Read from the socket to get the client's request
5. Write to the socket to respond

4

What Client Does

1. Create a socket connected to the server computer and port

```
Socket s = new Socket ("www.google.com", 80);
```

2. Write the request to the socket
3. Read the response

5

Reading Sockets

- ⦿ A socket has an input stream (to read from) and an output stream (to write to)

```
InputStream inStream = socket.getInputStream();
```

- ⦿ To get a more convenient and more efficient way to read regular text:

```
BufferedReader reader = new BufferedReader(  
    new InputStreamReader(inStream));
```

- ⦿ To read a line of text:

```
String line = reader.readLine();
```

6

Writing Sockets

- Get the output stream (of the same socket)

```
OutputStream outputStream = socket.getOutputStream();
```

- To get a more convenient and more efficient way to write regular text:

```
PrintStream writer = new PrintStream(outputStream);
```

- To write a line of text:

```
writer.println(line);
```

7

Multi-Threaded Server

- Start a new thread to handle each request that comes in

```
while (true) {  
    Socket s = ss.accept();  
    Worker ws = new Worker(root, workerNumber);  
    ws.setSocket(s);  
    (new Thread(ws, "worker " + workerNumber)).start();  
    workerNumber++;  
}
```

8

Worker Thread

```
public void run() {
    System.out.println("Starting worker " + id);
    try {
        handleClient();
    } catch (IOException e) {
        e.printStackTrace();
    }
    System.out.println("Done worker " + id);

    // Release the socket
    s = null;
}
```

9

Apache, Tomcat and Java Servlets

◀ Apache

- ◉ a popular, open source Web server
- ◉ handles HTTP requests
- ◉ Responds directly to normal file/directory requests

◀ Tomcat

- ◉ Interfaces between Apache and servlets
- ◉ Manages a pool of threads that servlets run in

◀ Java Servlet

- ◉ Java program accessible via the Web
- ◉ Runs on the server

10

Worker Pool

- ⦿ Better response to requests:
 - ⦿ No need to create Worker object
 - ⦿ If keep Worker thread running, no need to start and stop threads
- ⦿ Too many worker threads can rapidly degrade performance due to thrashing

11