

---

# Network Applications and Network Programming: Web

9/23/2009

1

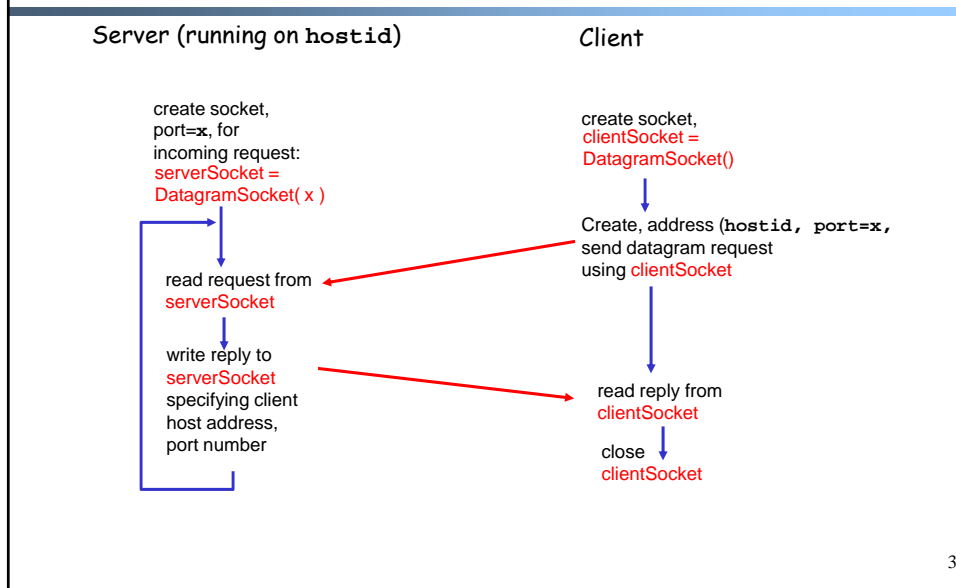
## Outline

---

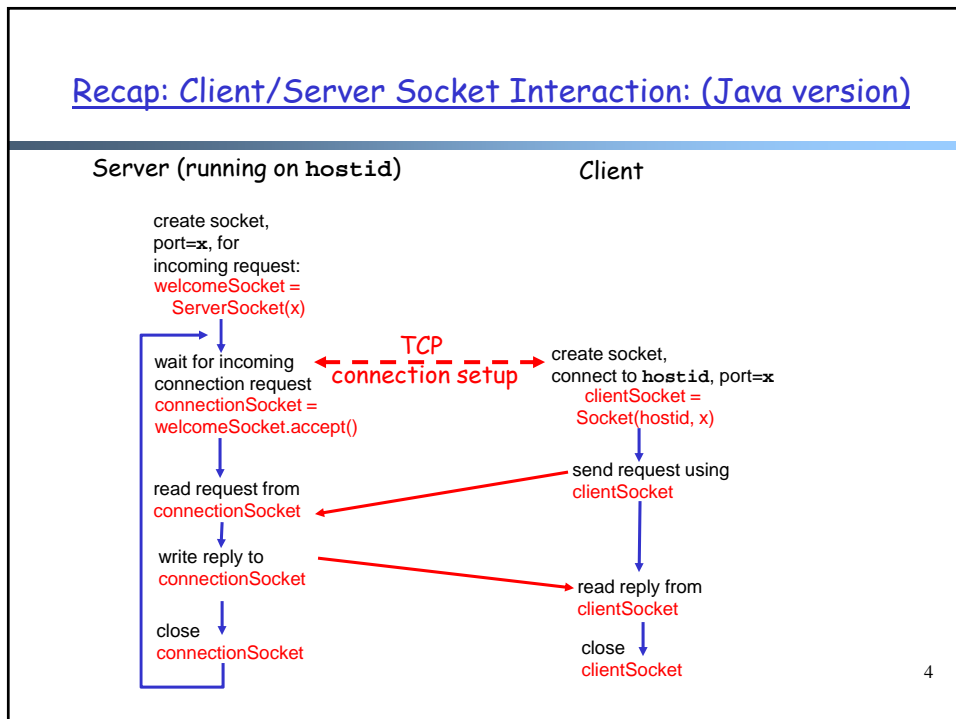
- Recap
- FTP
- Web

2

## Recap: Connectionless: Big Picture (Java version)



## Recap: Client/Server Socket Interaction: (Java version)



## Recap: Some Issues to Consider

- ❑ Pay attention to encoding of data: what transport layer handles only a sequence of bytes, the meaning of the bytes is by app.
  - String/char <-> bytes depends on charset
  - `DataOutputStream writeBytes(String)`
  - Please read chapter 4 of Java Network Programming for more details
  - Typically network protocols are using big-endian order: `int x = 0x0A0B0C0D`
  
- Use programming assignment 0 as a good practice: try `ByteBuffer`

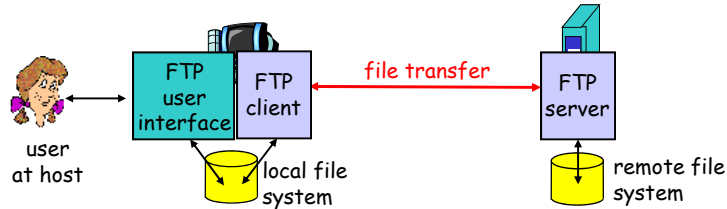
5

## Outline

- ❑ Recap
- ❑ FTP
- ❑ Web

6

## FTP: the File Transfer Protocol

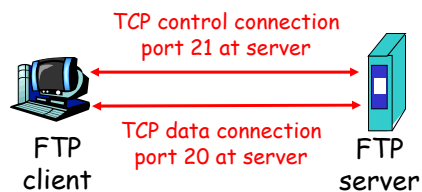


- ❑ Transfer files to/from remote host
- ❑ Client/server model
  - *client*: side that initiates transfer (either to/from remote)
  - *server*: remote host
- ❑ ftp: RFC 959
- ❑ ftp server: port 21 (smtp 25, http 80)

7

## FTP: A Client-Server Application with Separate Control, Data Connections

- ❑ Two parallel TCP connections opened:
  - **control**: exchange commands, responses between client, server.  
"out of band control"
    - port 21 at server
  - **data**: file data to/from server
    - port 20 at server
- ❑ ftp server maintains "state", e.g.,
  - current directory,
  - earlier authentication



8

## FTP Commands, Responses

- Is the application extensible, scalable, robust, secure?

### Sample commands:

- ❑ sent as ASCII text over control channel
- ❑ `USER username`
- ❑ `PASS password`
- ❑ `PORT h1,h2,h3,h4,p1,p2` specifies the IP address and port the client receives its data
- ❑ `LIST` return list of file in current directory
- ❑ `RETR filename` retrieves (gets) file
- ❑ `STOR filename` stores file

### Sample return codes

- ❑ status code and phrase (as in http)
- ❑ 331 Username OK, password required
- ❑ 125 data connection already open; transfer starting
- ❑ 425 Can't open data connection
- ❑ 452 Error writing file

Discussion: why separate control/data connections?

9

## Outline

- ❑ Recap
- ❑ FTP
- ❑ Web

10

## The Web: Some Jargon

- Web page:
    - consists of "objects"
    - addressed by a URL
  - Most Web pages consist of:
    - base HTML page, and
    - several referenced objects
  - URL has two components: host name, port number and path name:
- User agent for Web is called a browser, e.g.
    - Mozilla Firefox
    - MS Internet Explorer
  - Server for Web is called Web server:
    - Apache
    - MS Internet Information Server

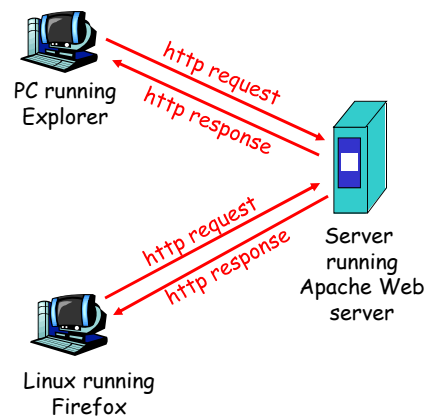
<http://www.cs.yale.edu:80/index.html>

11

## The Web: the HTTP Protocol

### HTTP: hypertext transfer protocol

- Web's application layer protocol
- HTTP uses TCP as transport service
- client/server model
  - *client*: browser that requests, receives, "displays" Web objects
  - *server*: Web server sends objects in response to requests
- http1.0: RFC 1945
- http1.1: RFC 2068



12

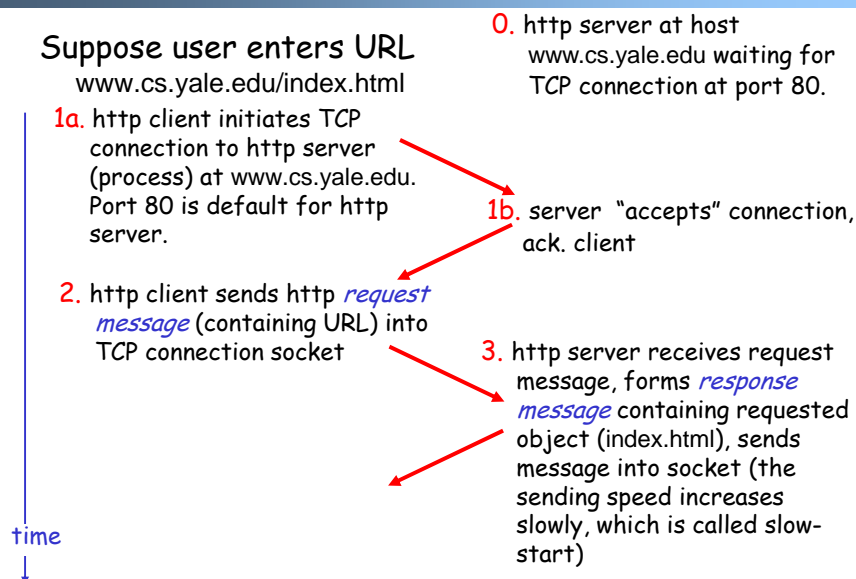
## HTTP 1.0 Message Flow

- ❑ Client initiates TCP connection (creates socket) to server, port 80
- ❑ Server waits for requests from clients
- ❑ Client sends request for a document
- ❑ Web server sends back the document
- ❑ TCP connection closed
  
- ❑ Client parses the document to find embedded objects (images)
  - repeat above for each image

13

## HTTP 1.0 Message Flow (more detail)

Suppose user enters URL  
www.cs.yale.edu/index.html



14

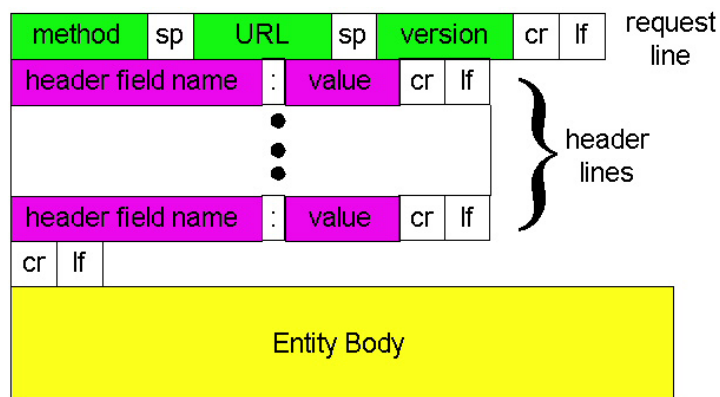
## HTTP 1.0 Message Flow (cont.)

4. http server closes TCP connection.
5. http client receives response message containing html file, parses html file, finds embedded image
- time ↓
6. Steps 1-5 repeated for each of the embedded images

15

## HTTP Request Message: General Format

- ASCII (human-readable format)



16

## HTTP Request Message Example: GET

request line  
(GET, POST,  
HEAD commands) → GET /somedir/page.html HTTP/1.0

header  
lines → Host: [www.somechool.edu](http://www.somechool.edu)  
Connection: close  
User-agent: Mozilla/4.0  
Accept: text/html, image/gif, image/jpeg  
Accept-language: en

Carriage return,  
line feed  
indicates end  
of message → (extra carriage return, line feed)

17

## HTTP Response Message

status line  
(protocol  
status code  
status phrase) → HTTP/1.0 200 OK

header  
lines → Date: Wed, 23 Jan 2008 12:00:15 GMT  
Server: Apache/1.3.0 (Unix)  
Last-Modified: Mon, 22 Jun 1998 .....  
Content-Length: 6821  
Content-Type: text/html

data, e.g.,  
requested  
html file → data data data data data ...

18

## HTTP Response Status Codes

In the first line of the server->client response message. A few sample codes:

### **200 OK**

- request succeeded, requested object later in this message

### **301 Moved Permanently**

- requested object moved, new location specified later in this message (Location:)

### **400 Bad Request**

- request message not understood by server

### **404 Not Found**

- requested document not found on this server

### **505 HTTP Version Not Supported**

19

## Trying out HTTP (client side) for yourself

1. Telnet to your favorite Web server:

```
telnet www.yale.edu 80
```

Opens TCP connection to port 80 (default http server port) at www.yale.edu. Anything typed in sent to port 80 at www.yale.edu

2. Type in a GET http request:

```
GET /index.html HTTP/1.0
```

By typing this in (hit carriage return twice), you send this minimal (but complete) GET request to http server

3. Look at response message sent by the http server.

20

## HTTP/1.0 Delay

---

- ❑ For each object:
  - TCP handshake --- 1 RTT
  - client request and server responds --- at least 1 RTT (if object can be contained in one packet)
  
- ❑ Discussion: how to reduce delay?

21

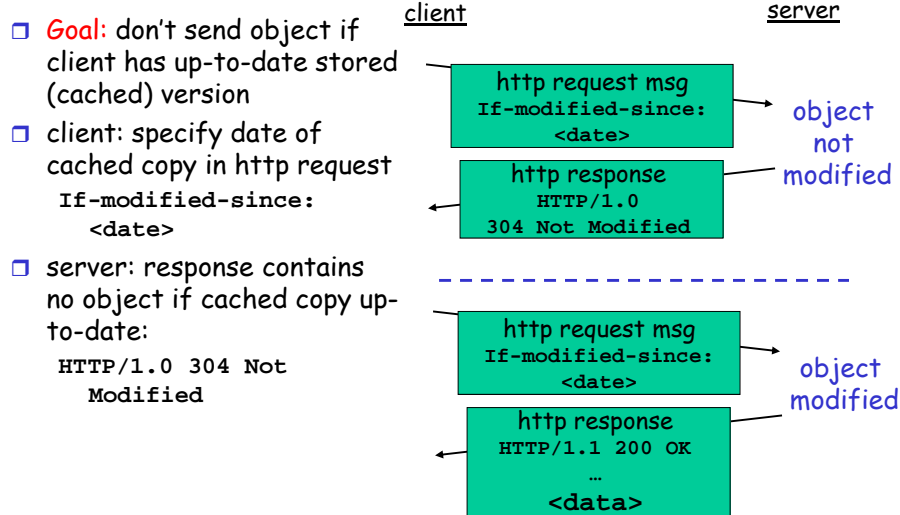
## HTTP Message Flow: Persistent HTTP

---

- ❑ Default for HTTP/1.1
  
- ❑ On same TCP connection: server parses request, responds, parses new request, ...
  
- ❑ Client sends requests for all referenced objects as soon as it receives base HTML
  
- ❑ Fewer RTTs

22

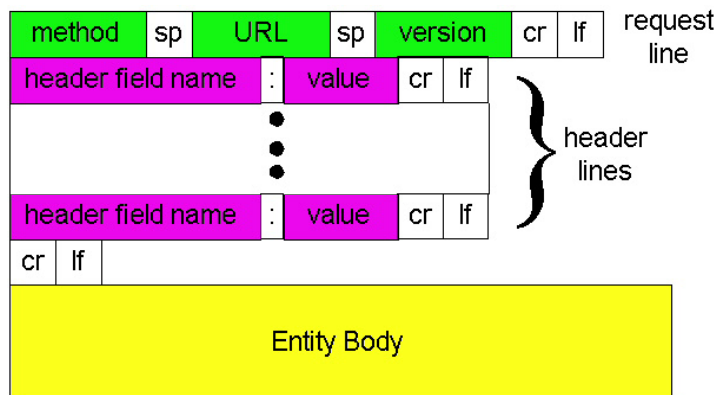
## Browser Cache and Conditional GET



23

## HTTP Message Extension: Form

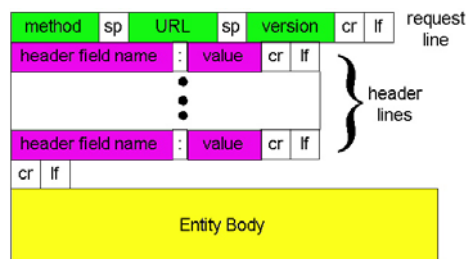
- if an HTML page contains forms, they are encoded in message body



24

## HTTP Message Flow Extensions: Keeping State

- ❑ Why do we need to keep state?
- ❑ How does FTP keep state (e.g., current dir) and why does HTTP not use it?

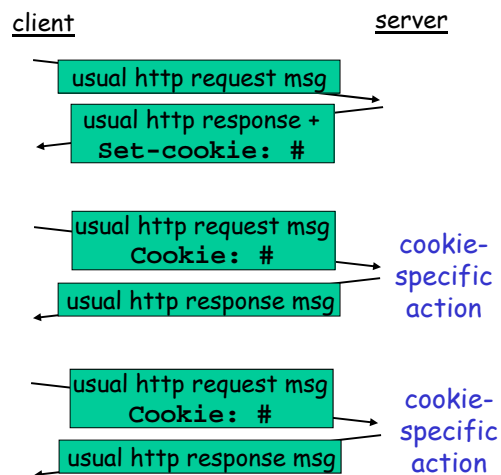


25

## User-server Interaction: Cookies

Goal: no explicit application level session

- ❑ Server sends "cookie" to client in response msg  
Set-cookie: 1678453
- ❑ Client presents cookie in later requests  
Cookie: 1678453
- ❑ Server matches presented-cookie with server-stored info
  - authentication
  - remembering user preferences, previous choices



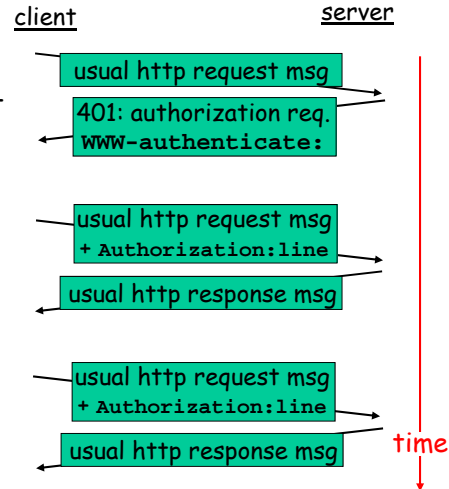
26

## User-Server Interaction: Authentication

**Authentication goal:** control access to server documents

- **stateless:** client must present authorization in each request
- authorization: typically name, password
  - Authorization: header line in request
  - if no authorization presented, server refuses access, sends WWW-authenticate: header line in response

Browser caches name & password so that user does not have to repeatedly enter it.

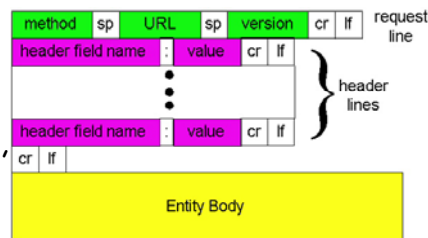


27

## Summary: HTTP

- Is the application extensible, scalable, robust, secure?

- HTTP message format
  - ASCII (human-readable format) requests, header lines, entity body, and responses line
- HTTP message flow
  - stateless server
    - each request is self-contained; thus cookie and authentication, are needed in each message
  - reducing latency
    - persistent HTTP
      - the problem is introduced by layering !
    - conditional GET reduces server/network workload and latency
    - cache and proxy reduce traffic and latency



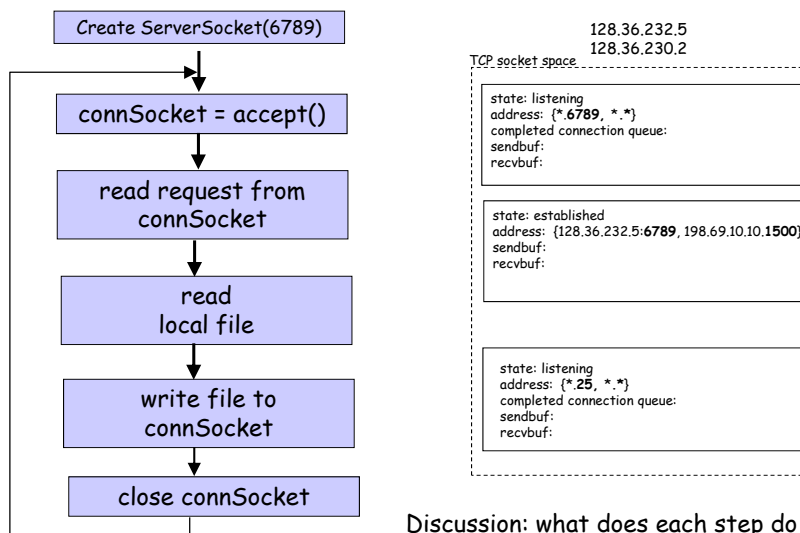
28

## Example: WebServer

- A simple web server which supports only simple HTTP/1.0

<http://zoo.cs.yale.edu/classes/cs433/programming/examples-java-socket/WebServer/>

## WebServer Flow



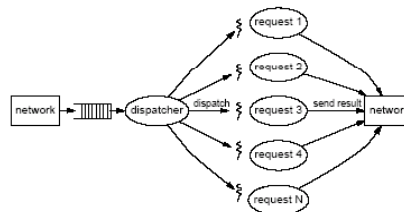
Discussion: what does each step do and how long does it take?

## Writing High Performance Servers: Major Issues

- ❑ Many socket/IO operations can cause a process to block, e.g.,
  - accept: waiting for new connection;
  - read a socket waiting for data or close;
  - write a socket waiting for buffer space;
  - I/O read/write for disk to finish
  
- ❑ Thus a crucial perspective of network server design is the concurrency design (non-blocking)
  - for high performance
  - to avoid denial of service
  
- ❑ Concurrency is also important for clients!

## Writing High Performance Servers: Using Multi-Threads

- ❑ Using multiple threads
  - so that only the flow processing a particular request is blocked
  - A thread is a sequence of instructions which may execute in parallel with other threads



## Java Thread

- ❑ Two ways to implement Java thread
  - Extend the Thread class, or
  - Implement the Runnable interface

```
class RequestHandler
    extends Thread {
    public void run() { }
    ...
}

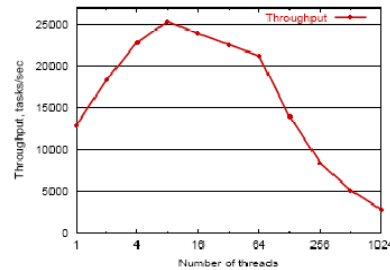
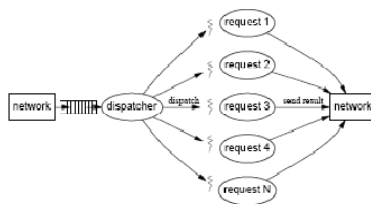
Thread t =
    new RequestHandler(requestInfo);
t.start();
```

```
class RequestHandler
    implements Runnable {
    public void run() { }
    ...
}

RequestHandler rh =
    new RequestHandler(requestInfo);
Thread t = new Thread(rh);
t.start();
```

Example: a Multi-threaded WebServer, which creates a thread for each request 33

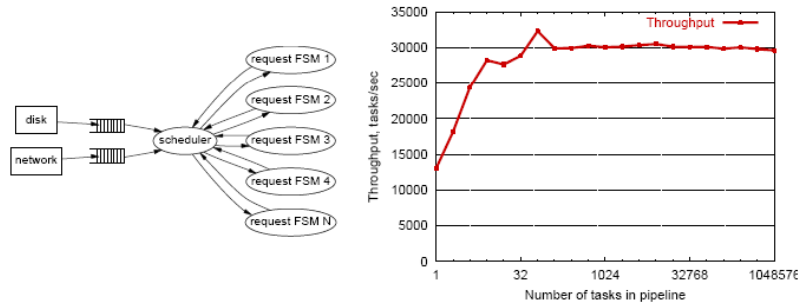
## Problems of Multi-Thread Server



(937 MHz x86, Linux 2.2.14, each thread reading 8KB file)

- ❑ High resource usage, context switch overhead, contended locks
- ❑ Too many threads → throughput meltdown, response time explosion
- ❑ In practice: bound total number of threads

## Event-Driven Programming



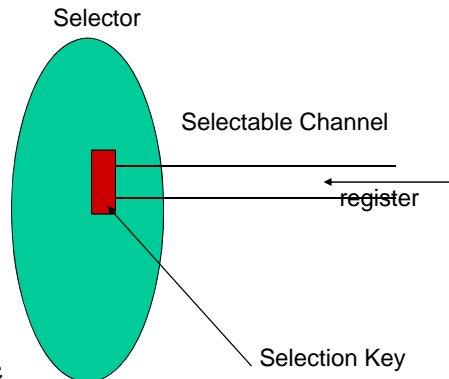
- ❑ Event-driven programming, also called asynchronous i/o
- ❑ Using Finite State Machines (FSM) to monitor the progress of requests
- ❑ Yields efficient and scalable concurrency
- ❑ Many examples: Click router, Flash web server, TP Monitors, etc.
  
- ❑ Java: asynchronous i/o
  - for an example see: <http://www.cafeaulait.org/books/jnp3/examples/12/>

## Async I/O in Java

- ❑ An important class is the class `Selector`, which is a multiplexer of selectable channel objects
  - example channels: `DatagramChannel`, `ServerSocketChannel`, `SocketChannel`
  - use `configureBlocking(false)` to make a channel non-blocking
  
- ❑ A selector may be created by invoking the `open` method of this class

## Async I/O in Java

- ❑ A selectable channel registers events (called a `SelectionKey`) with a selector with the `register` method
- ❑ A `SelectionKey` object contains two *operation sets*
  - interest Set
  - ready Set
- ❑ A `SelectionKey` object has an attachment which can store data
  - often the attachment is a buffer



## Async I/O in Java

- ❑ Call `select` (or `selectNow()`, or `select(int timeout)`) to check for ready events, called the selected key set
- ❑ Iterate over the set to process all ready events

## Problems of Event-Driven Server

---

- ❑ Difficult to engineer, modularize, and tune
- ❑ Little OS and tool support: "roll your own"
- ❑ No performance/failure isolation between FSMs
- ❑ FSM code can never block (but page faults, garbage collection may still force a block)
  - thus still need multiple threads

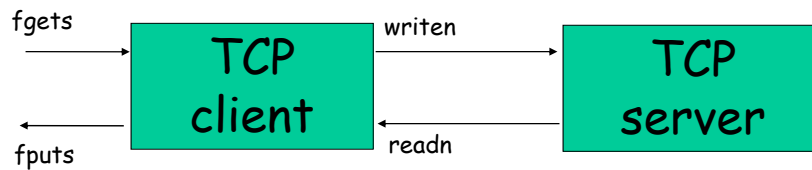
## Optional Slides

## Asynchronous Network Programming

(C/C++)

41

### A Relay TCP Client: telnet-like Program



<http://zoo.cs.yale.edu/classes/cs433/programming/examples-c-socket/tcpclient>

42

## Method 1: Process and Thread

- process
  - fork()
  - waitpid()
  
- Thread: light weight process
  - pthread\_create()
  - pthread\_exit()

43

## pthread

```
Void main() {
char  recvline[MAXLINE + 1];
ss = new socketstream(sockfd);

pthread_t tid;
if (pthread_create(&tid, NULL, copy_to, NULL)) {
    err_quit("pthread_creat()");
}

while (ss->read_line(recvline, MAXLINE) > 0) {
    fprintf(stdout, "%s\n", recvline);
}

void *copy_to(void *arg) {
char  sendline[MAXLINE];

if (debug) cout << "Thread create()!" << endl;
while (fgets(sendline, sizeof(sendline), stdin))
    ss->writen_socket(sendline, strlen(sendline));

shutdown(sockfd, SHUT_WR);
if (debug) cout << "Thread done!" << endl;

pthread_exit(0);
}
```

44

## Method 2: Asynchronous I/O (Select)

- select: deal with blocking system call  
int select(int n, fd\_set \*readfds, fd\_set \*writefds, fd\_set \*exceptfds, struct timeval \*timeout);

FD\_CLR(int fd, fd\_set \*set);

FD\_ZERO(fd\_set \*set);

FD\_ISSET(int fd, fd\_set \*set);

FD\_SET(int fd, fd\_set \*set);

45

## Method 3: Signal and Select

- signal: events such as timeout

46

## Examples of Network Programming

- ❑ Library to make life easier
- ❑ Four design examples
  - TCP Client
  - TCP server using select
  - TCP server using process and thread
  - Reliable UDP
- ❑ Warning: It will be hard to listen to me reading through the code. Read the code.

47

## Example 2: A Concurrent TCP Server Using Process or Thread

- ❑ Get a line, and echo it back
- ❑ Use select()
- ❑ For how to use process or thread, see later
- ❑ Check the code at:  
<http://zoo.cs.yale.edu/classes/cs433/programming/examples-c-socket/tcpserver>
- ❑ Are there potential denial of service problems with the code?

48

## Example 3: A Concurrent HTTP TCP Server Using Process/Thread

- ❑ Use process-per-request or thread-per-request

- ❑ Check the code at:

[http://zoo.cs.yale.edu/classes/cs433/programming/examples-c-socket/simple\\_httpd](http://zoo.cs.yale.edu/classes/cs433/programming/examples-c-socket/simple_httpd)