Client Server Programming

Srinidhi Varadarajan

Network Applications

- **There are many network applications**
  - **Network applications involve the cooperation of processes running on different hosts connected by a network**
- **Applications may be “standard” or custom applications**
  - **Internet applications are typically defined in one or more Request for Comments (RFCs)**
    - **HTTP defined in RFC 1945**
  - **May be standard, drafts, or informational**

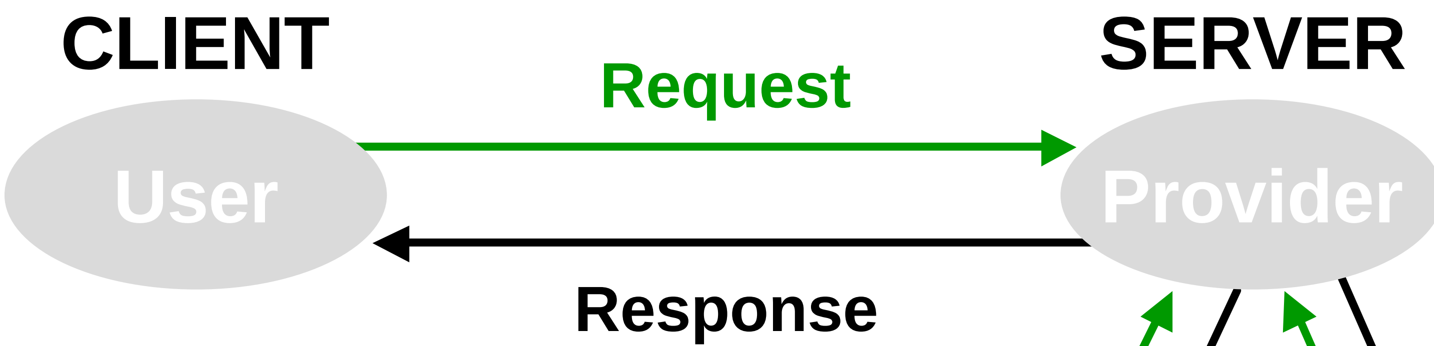
Port Assignment

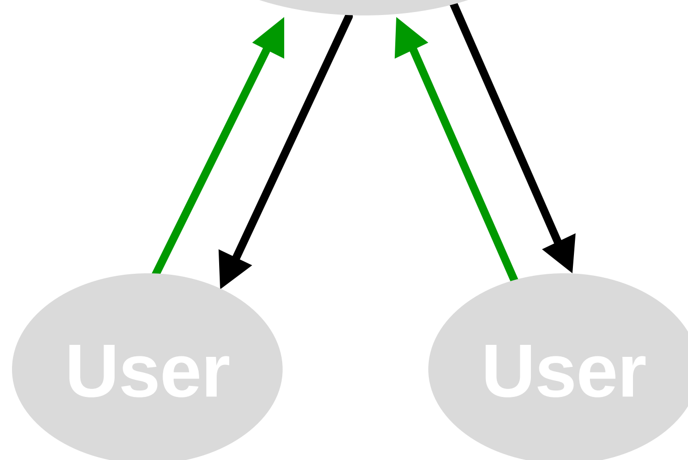
- **UDP and TCP ports are used to distinguish between multiple applications on one host**
- **Standard numbering for “well-known port numbers”**
  - **Defined in RFC 1700 for “standard” Internet applications**
  - **Configured in various places specific to the operating system and in the application itself**
    - **Windows 95/98: \Windows\services**
    - **NT: Systemroot\System32\Drivers\Etc\services**
    - **UNIX: /etc/services**

Sample From /etc/services

```
echo              7/tcp
echo              7/udp
discard           9/tcp    sink null
discard           9/udp    sink null
systat           11/tcp
systat           11/tcp    users
daytime          13/tcp
daytime          13/udp
netstat          15/tcp
qotd             17/tcp    quote
qotd             17/udp    quote
chargen          19/tcp    ttytst source
chargen          19/udp    ttytst source
```

Service User Versus Service Provider





- **Server runs awaiting requests and responds when requests are received**
- **Client issues requests to server and accepts response**

There may be multiple users of one provider

Concurrency at the Server

- **Many servers provide concurrent operation**
  - **Apparent concurrency using asynchronous socket I/O**
  - **True (program-level) concurrency using multithreaded design**
- **Concurrency adds complexity!**
- **When is concurrency justified?**
  - **Need to simultaneously handle multiple requests**
  - **Need to increase performance**

Example Standard Service: TELNET

- TELNET is a standard application protocol for remote login
  - Defines format of data sent by application program to remote machine and by remote machine to the application
  - Defines character encoding
  - Defines special messages to control the session
- telnetd is server running on the remote host (at port 23)
- Client is the application program on the local host, e.g. CRT or other TELNET client

TELNET to Access Alternative Services

- **A TELNET client can be used to access alternative servers**
  - **Simple text transfer -- so can access general text based services**
  - **Typical TELNET clients can be configured to access different remote ports**
  - **Of course, other clients are designed to provide a better user interface**

Peer-to-Peer Communication Model

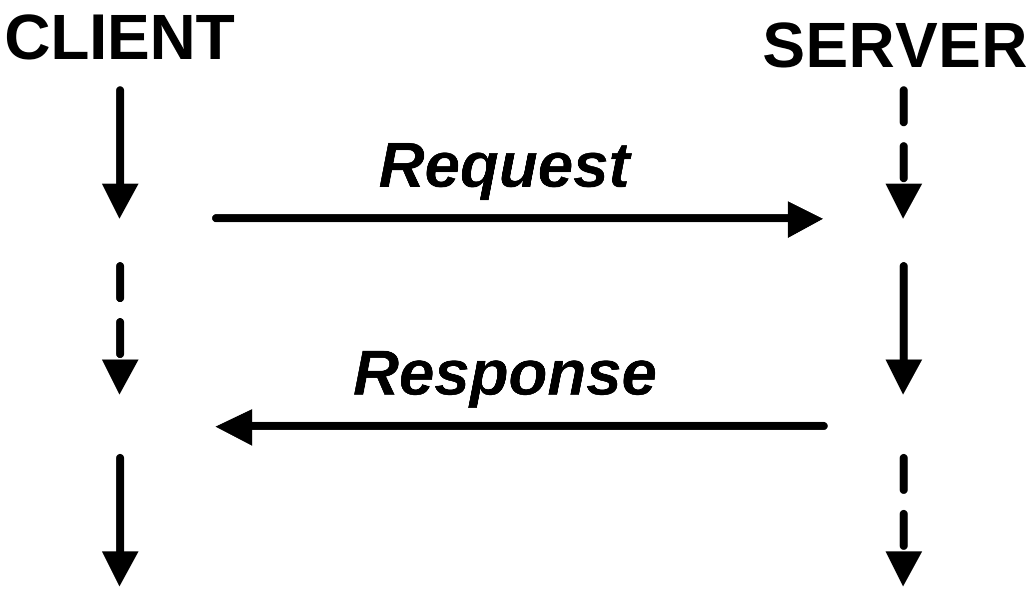
- **TCP/IP suite supports *peer-to-peer* communication**
- **Peer-to-peer communication is *symmetric***
  - **Any node can initiate or terminate communication**
  - **Communication can occur in either direction**
- **There are no implications of …**
  - **When applications should interact**
  - **Meaning of data -- they're just bytes**
  - **Structure of a networked application**

Application-Level Model

- **Higher level model needed to implement networked applications**
- **TCP and UDP require that a program be available to accept a connection request (TCP) or a datagram (UDP)**
- ***Client-server model* is widely used to provide a workable structure**

Client-Server Model

- **Client initiates peer-to-peer communication (at TCP- or UDP-level)**
- **Server waits for incoming request**



Clients Versus Servers

- **Clients**
  - **Relatively simple (with respect to network communication)**
  - **User-level programs that require no special privileges**
- **Servers**
  - **More complex than servers due to performance and security requirements**
  - **Often require special system privileges**
  - **May run all the time or be started on-demand by operating system mechanisms, e.g. inetd in UNIX**

Privilege

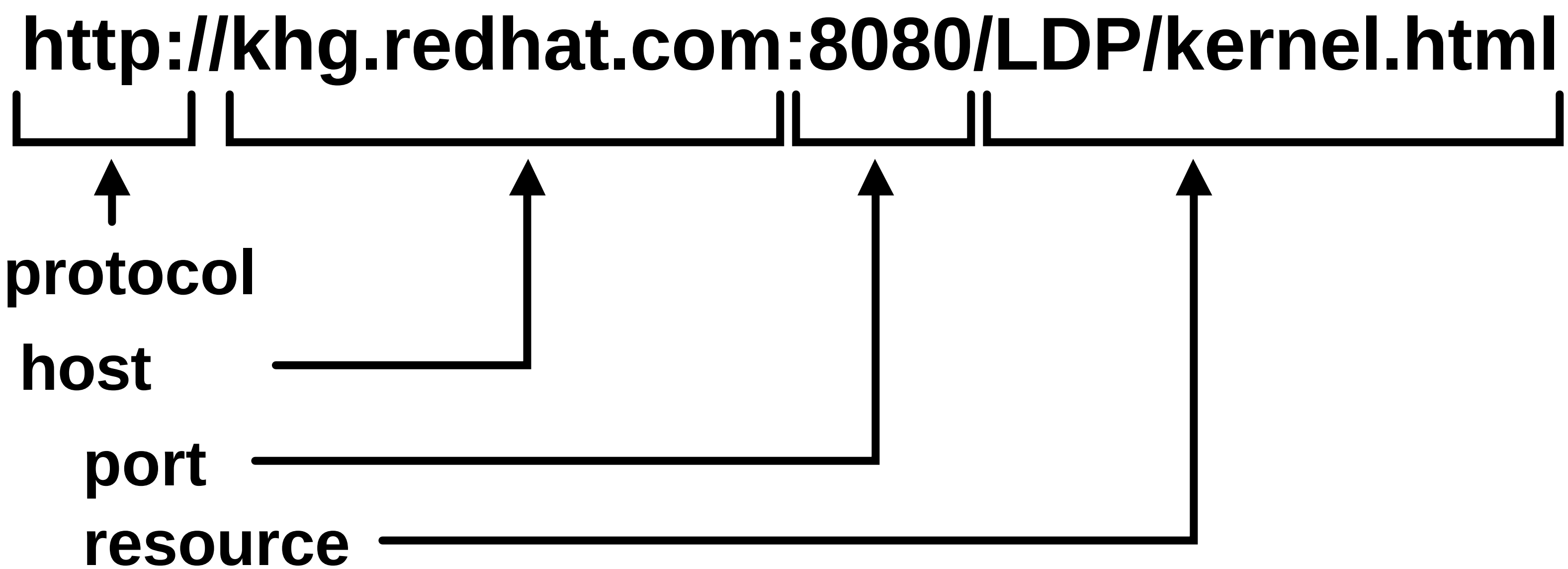
- **Server often runs in a privileged mode, so must protect improper use of privileges by a client**
  - ***Authentication*: verify identity of the client**
  - ***Authorization*: verify permission to access service**
  - ***Data security and privacy*: prevent unauthorized viewing or altering of data**
  - ***Protection*: protect system resources from misuse**

Client Parameterization

- **Parameterized clients lead to generality, e.g. as in TELNET client being able to access other services**
- **Parameters**
  - **Destination host**
    - **Host name: vtopus.cs.vt.edu**
    - **IP address: 128.173.40.24**
  - **Port number (not just default)**
  - **Protocol- or application-specific information, e.g. block size**
  - **Protocol itself, e.g. FTP, HTTP, or Gopher**

Universal Resource Locators (1)

- **URLs integrate many parameters**



Universal Resource Locators (2)

ftp://ftp.cs.purdue.edu/pub/comer/

protocol

host

(default FTP port)

resource

Connectionless/Connection-oriented (1)

- ***Connection-oriented* servers**
  - **Client must first connect to the server prior to any data transfer**
  - **Based on TCP (usually) -- reliable at the expense of connection overhead**
    - **Data arrives correctly**
    - **Data ordering is maintained**
    - **Data is not duplicated**

Connectionless/Connection-oriented (2)

- ***Connectionless* servers**
  - **Data can be sent by clients immediately**
  - **Based on UDP (usually) -- no connection overhead, but no benefits**
    - **Data may not arrive**
    - **Data may be incorrect, although unlikely**
    - **Duplicates may arrive, although unlikely**
    - **May arrive out of order, although unlikely**

Connectionless/Connection-oriented (3)

- **Connectionless vs. connection-oriented design issues**
  - **Inherent reliability?**
  - **Reliability needed?**
  - **Reliability is already very high (LAN vs. WAN)?**
  - **Real-time operation gives no time for error correction (retransmission)?**
  - **Need for broadcast or multicast?**
- **Need to test in a variety of environments**
  - **Packet delay**
  - **Packet loss**

Stateless/Stateful

- **State information is any information about ongoing interactions**
- ***Stateful servers* maintain state information**
- ***Stateless servers* keep no state information**
- **Examples -- stateful or stateless?**
  - **Finger?**
  - **TELNET?**
  - **HTTP?**
  - **FTP?**
  - **NFS?**

File Server Example

- **Consider a file server that supports four operations**
  - **OPEN -- identify file and operation, e.g. read or write**
  - **READ -- identify file, location in file, number of bytes to read**
  - **WRITE -- identify file, location in file, number of bytes, data to write**
  - **CLOSE -- identify file**

File Server Example: Stateless

- **Stateless version -- identify all information with each request**
- **Example**
  - **OPEN(/tmp/test.txt, “r”)**
  - **READ(/tmp/test.txt, 0, 200)**
  - **READ(/tmp/test.txt, 200, 200)**
- **Redundant information is provided with subsequent requests**
  - **Inefficient with respect to information transfer**
  - **Server operation is simplified**

File Server Example: Stateful (1)

- **Stateful version -- server provides *handle* to access state at the server**
- **File open**
  - **Request: OPEN(/tmp/test.txt, “r”)**
  - **Reply: OPEN(ok, 32) -- handle = 32**
  - **State: 32: /tmp/test.txt, 0, read**
- **File read**
  - **Request: READ(32, 200)**
  - **Reply: READ(ok, data)**
  - **State: 32: /tmp/test.txt, 200, read**

File Server Example: Stateful (2)

- **What if there is a duplicate request?**
  - **READ(32, 200) sent once, but received twice**
  - **Client and server lose synchronization -- server thinks that 400 bytes have been read, client thinks it has read just 200 bytes**
- **Stateful servers are more complex than stateless servers since they must deal with synchronization**
- **State is implied by the protocol, not the implementation**
  - **TCP is a stateful protocol**
  - **Synchronization required with byte numbers**

Stateful Protocol Design Issues

- **Time-outs**
- **Duplicate requests and replies**
- **System crashes (at one end)**
- **Multiple clients**
- **File locking**

Concurrency in Network Applications

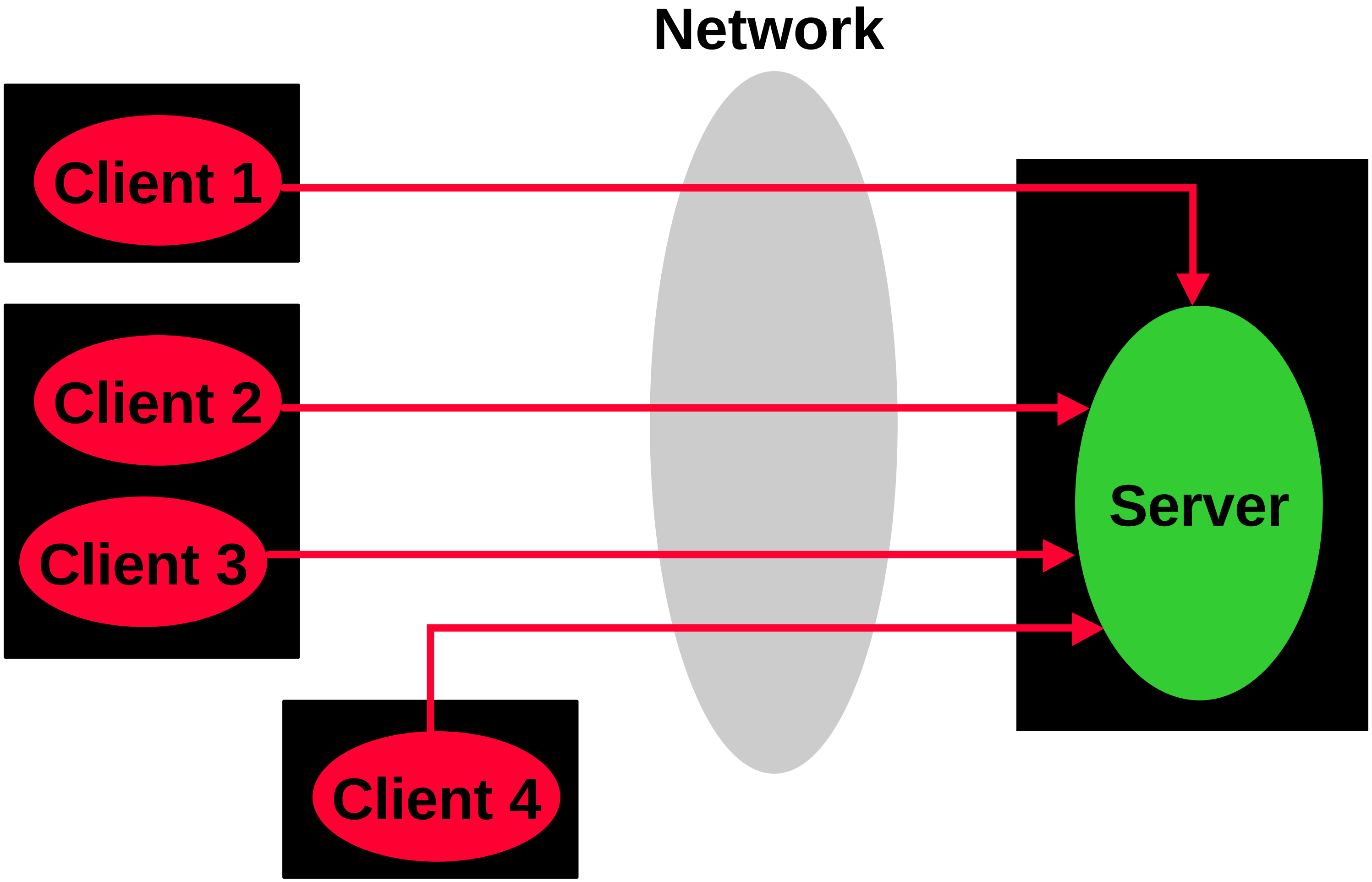
- ***Concurrency* is real or apparent simultaneous computing**
  - **Real in a multiprocessor**
  - **Apparent in a time-shared uniprocessor (apparent concurrency provided by OS)**

- **Networks are inherently concurrent -- multiple hosts have the appearance of simultaneously transferring data**
  - **Real, to some extent, in switched networks**
  - **Apparent in shared media networks (apparent concurrency provided by MAC protocol)**

Client Concurrency

- Clients usually make use of concurrency in a trivial way
  - Multiple clients can run on a single processor
- Such concurrency is provided by the operating system, not by any programmed features of the client
- Note that complex clients can use concurrency, e.g. modern Web browser
  - Simultaneous requests and receipt of multiple files
  - Overlapping communication with graphical rendering or other processing

Server Concurrency (1)



Server Concurrency (2)

- **Servers use concurrency to achieve functionality and performance**
- **Concurrency is inherent in the server -- must be explicitly considered in server design**
- **Exact design and mechanisms depend on support provided by the underlying operating system**
- **Achieved through**
  - **Concurrent processes**
  - **Concurrent threads**

Processes

- ***Process*: fundamental unit of computation**
  - **Per process information:**
    - **Owner of process**
    - **Program being executed**
    - **Program and data memory areas**
    - **Run-time stack for procedure activation**
    - **Instruction pointer**
    - **Allocated resources, e.g. file and socket descriptors**
- **A *program* implies just the code, a *process* includes the concept of the active execution of the code**

Concurrent Execution

- ***Concurrent execution*: executing a piece of code more than once at apparently the same time**
- **If a program is executed multiple times at apparently the same time**
  - **Each invocation is a unique process**
  - **Each invocation has its own unique per process information, such as distinct instruction pointer, program and data memory, resources, etc.**

Threads

- **Threads are another form of concurrent execution *within* a process**
  - **Each thread has its own:**
    - **Instruction pointer**
    - **Copy of *local* variables**
    - **Run-time stack for procedure activation**
  - **Multiple threads can be associated with a single process**
  - **All threads within a process share:**
    - **Process owner**
    - **Program being executed**
    - **Program and *global* data memory**
    - **Allocated resources**

Processes Versus Threads

- Both provide mechanisms for concurrent execution
- Advantages of threads
  - Allocated resources and global data are easily shared
  - Typically lower overhead for creation and switching (but not zero overhead)
- Advantages of multiple processes
  - Inherent separation (isolation) makes interaction clearer
  - More widely supported on different operating systems; common mechanisms

Context Switching

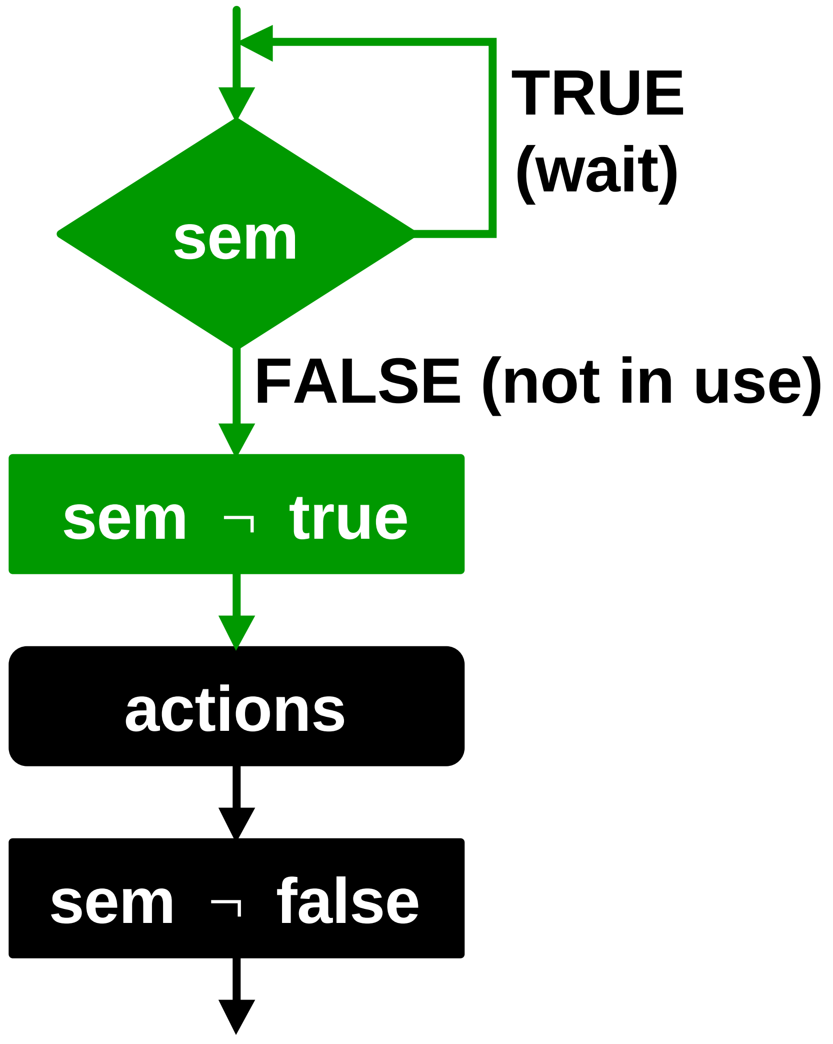
- ***Context switching*** **is the operation of changing from the execution of one process or thread to another**
  - **Overhead incurred with each context switch**
  - **Context switch for threads requires less overhead than for processes**
    - **Threads are "lightweight processes"**

Mutual Exclusion

- **Threads do share allocated resources (files, sockets, etc.) and global memory**
- **So, some form of *mutual exclusion* is needed to ensure that only a single thread has use of a particular resource at any given time**
- **Mutual exclusion can be implemented using a “test and set” operation on a true-false value**

Semaphore Operation

- Semaphore is variable *sem*
  - TRUE Þ in use
  - FALSE Þ not in use
- Semaphore (sem) is first initialized to FALSE
- Test-and-set must be an “indivisible” or “atomic” operation



Apparent Concurrency (1)

- **Threads allow concurrency to be implemented at the application level**
- **Apparent concurrency is also possible where server appears to be simultaneously serving requests, but is doing this with a single thread**
- **Based on asynchronous I/O**
  - ***Synchronous I/O* is blocking -- a call blocks until the source is ready**
  - ***Asynchronous I/O* is non-blocking**

Apparent Concurrency (2)

- **select() call**
  - **Allows a program to select between multiple services and returns when one becomes active**
  - **Basis for apparent concurrency**

You should now be able to … (1)

- **Specify general design requirements for clients and servers**
- **Characterize application protocols with respect to**
  - **Connection versus connection-less**
  - **Stateful versus stateless**
- **Identify design issues related to use of stateful and stateless protocols**
- **Identify the need for concurrent execution**

You should now be able to … (2)

- **Identify the properties of threads and processes**
- **Identify design issues related to the use of threads versus processes**
- **Identify the difference between concurrent execution with threads and apparent concurrency**