Introduction to Unix Network Programming

Reference: Stevens Unix Network Programming

Network Programming

- Key Components:
  - Internet protocols
    - IP, TCP, UDP, etc
  - Sockets
    - API - application programming interface
- Why focus on the Internet?
  - Internet Protocol (IP)
    - IP is standard
    - allows a common namespace across most of Internet
    - reduces number of translations, which incur overhead
  - Sockets
    - reasonably simple and elegant, Unix interface

Network Programming with Sockets

- Socket
  - Host-local, application-created, OS-controlled Application process can both send and receive messages to/from another application process
- Sockets API
  - A transport layer service interface
    - Introduced in 1981 by BSD 4.1
    - Implemented as library and/or system calls
    - Similar interfaces to TCP and UDP
    - Also interface to IP (for super-user); “raw sockets”

Beej's Guide

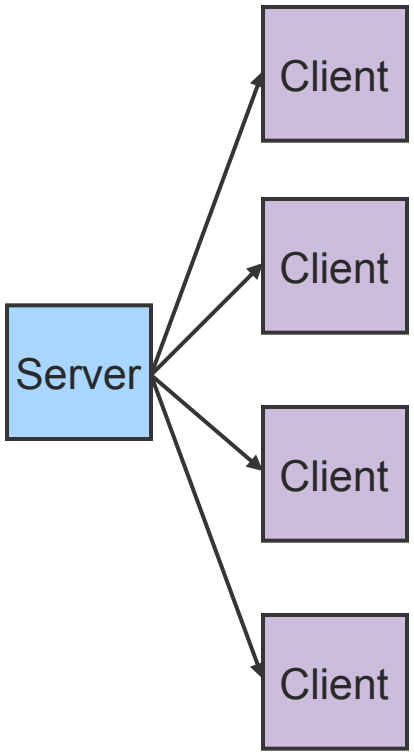
- How-to guide on network programming using Internet sockets, or "sockets programming"

  http://beej.us/guide/bgnet/

Outline

- Client-Sever Model
- TCP Connection
- UDP Services
- Sockets API
- Example

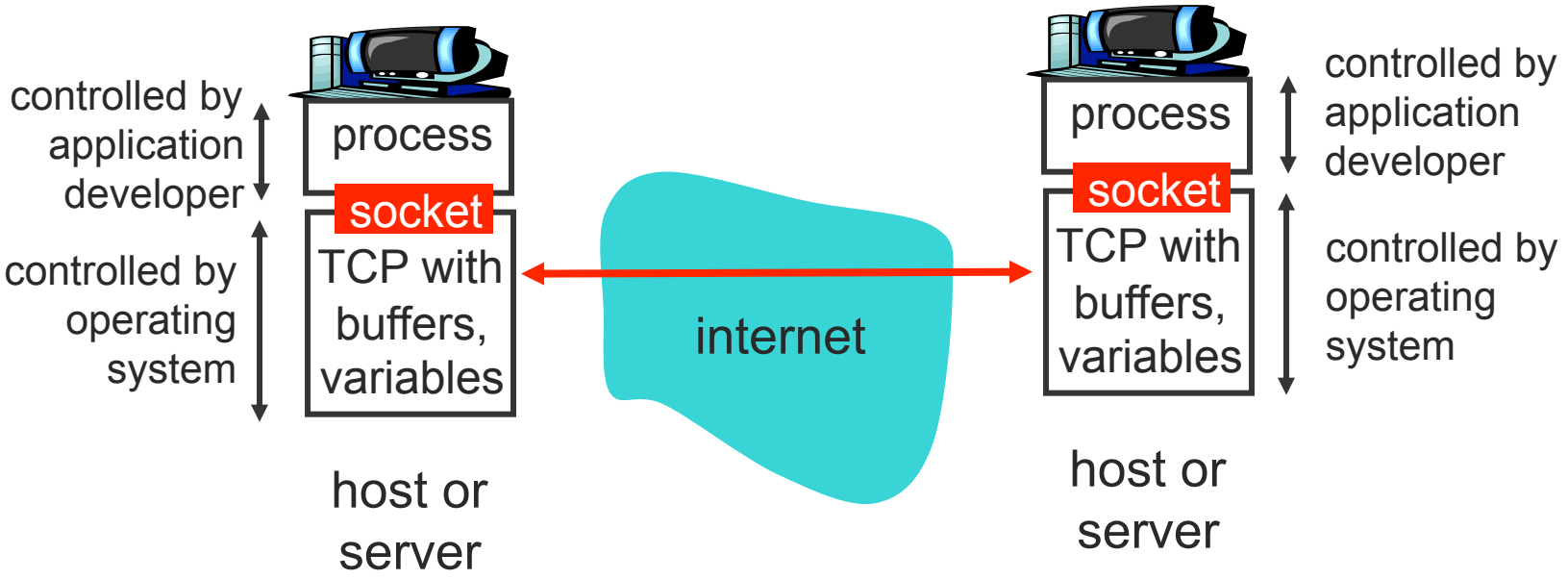
Client-Server Model



- Asymmetric Communication
  - Client sends requests
  - Server sends replies
- Server/Daemon
  - Well-known name
  - Waits for contact
  - Processes requests, sends replies
- Client
  - Initiates contact
  - Waits for response

TCP Connections

- Transmission Control Protocol (TCP) Service
  - OSI Transport Layer



TCP Connections

- Transmission Control Protocol (TCP) Service
  - OSI Transport Layer
  - Service Model
    - Byte stream (interpreted by application)
    - 16-bit port space allows multiple connections on a single host
    - Connection-oriented
      - Set up connection before communicating
      - Tear down connection when done

TCP Service

- Reliable Data Transfer
  - Guaranteed delivery
  - Exactly once if no catastrophic failures
- Sequenced Data Transfer
  - In-order delivery
- Regulated Data Flow
  - Monitors network and adjusts transmission appropriately
- Data Transmission
  - Full-Duplex byte stream

- Telephone Call
  - Guaranteed delivery
  - In-order delivery
  - Connection-oriented
  - Setup connection followed by conversation

UDP Services

- User Datagram Protocol Service
  - OSI Transport Layer
  - Provides a thin layer over IP
  - 16-bit port space (distinct from TCP ports) allows multiple recipients on a single host

UDP Services

- Unit of Transfer
  - Datagram (variable length packet)
- Unreliable
  - No guaranteed delivery
  - Drops packets silently
- Unordered
  - No guarantee of maintained order of delivery
- Unlimited Transmission
  - No flow control

- Postal Mail
  - Single mailbox to receive all letters
  - Unreliable
  - Not necessarily in-order
  - Letters sent independently
  - Must address each reply

Choose between TCP and UDP for each of these apps

- File downloads (e.g., Web)
- Sensor readings
- Robot control
- Nanny cam
- Peer-to-peer video distribution

Addresses and Data

- Goal: naming for machines on the Internet
- Internet domain names
  - Human readable
  - Variable length
  - Ex: `sal.cs.uiuc.edu`
- IP addresses
  - Each attachment point on Internet is given unique address
  - Easily handled by routers/computers
  - Fixed length
  - Somewhat geographical
  - Ex: `128.174.252.217`

Byte Ordering

- Big Endian vs. Little Endian
  - Little Endian (Intel, Arm):
    - Least significant byte of word is stored in the lowest memory address
  - Big Endian (Sun, SGI, HP):
    - Most significant byte of word is stored in the lowest memory address
  - Example: **`128.2.194.95`**

Big Endian	128	2	194	95
Little Endian	95	194	2	128

Byte Ordering

- Big Endian vs. Little Endian
  - Little Endian (Intel, Arm):
    - Least significant byte of word is stored in the lowest memory address
  - Big Endian (Sun, SGI, HP):
    - Most significant byte of word is stored in the lowest memory address
  - Network Byte Order = Big Endian
    - Must be used for some data (i.e. IP Addresses)
  - For your app, be consistent
    - Key to transmitting binary data

Byte Ordering Functions

- 16- and 32-bit conversion functions (for platform independence)
- Examples:

```
int m, n;
short int s,t;

m = ntohl (n)    // net-to-host long (32-bit) translation
s = ntohs (t)    // net-to-host short (16-bit) translation
n = htonl (m)    // host-to-net long (32-bit) translation
t = htons (s)    // host-to-net short (16-bit) translation
```


Reserved Ports

Keyword	Decimal	Description	Keyword	Decimal	Description
	0/tcp	Reserved	time	37/tcp	Time
	0/udp	Reserved	time	37/udp	Time
tcpmux	1/tcp	TCP Port Service	name	42/tcp	Host Name Server
tcpmux	1/udp	TCP Port Service	name	42/udp	Host Name Server
echo	7/tcp	Echo	nameserver	42/tcp	Host Name Server
echo	7/udp	Echo	nameserver	42/udp	Host Name Server
systat	11/tcp	Active Users	nicname	43/tcp	Who Is
systat	11/udp	Active Users	nicname	43/udp	Who Is
daytime	13/tcp	Daytime (RFC 867)	domain	53/tcp	Domain Name Server
daytime	13/udp	Daytime (RFC 867)	domain	53/udp	Domain Name Server
qotd	17/tcp	Quote of the Day	whois++	63/tcp	whois++
qotd	17/udp	Quote of the Day	whois++	63/udp	whois++
chargen	19/tcp	Character Generator	gopher	70/tcp	Gopher
chargen	19/udp	Character Generator	gopher	70/udp	Gopher
ftp-data	20/tcp	File Transfer Data	finger	79/tcp	Finger
ftp-data	20/udp	File Transfer Data	finger	79/udp	Finger
ftp	21/tcp	File Transfer Ctl	http	80/tcp	World Wide Web HTTP
ftp	21/udp	File Transfer Ctl	http	80/udp	World Wide Web HTTP
ssh	22/tcp	SSH Remote Login	www	80/tcp	World Wide Web HTTP
ssh	22/udp	SSH Remote Login	www	80/udp	World Wide Web HTTP
telnet	23/tcp	Telnet	www-http	80/tcp	World Wide Web HTTP
telnet	23/udp	Telnet	www-http	80/udp	World Wide Web HTTP
smtp	25/tcp	Simple Mail Transfer	kerberos	88/tcp	Kerberos
smtp	25/udp	Simple Mail Transfer	kerberos	88/udp	Kerberos

Socket interface

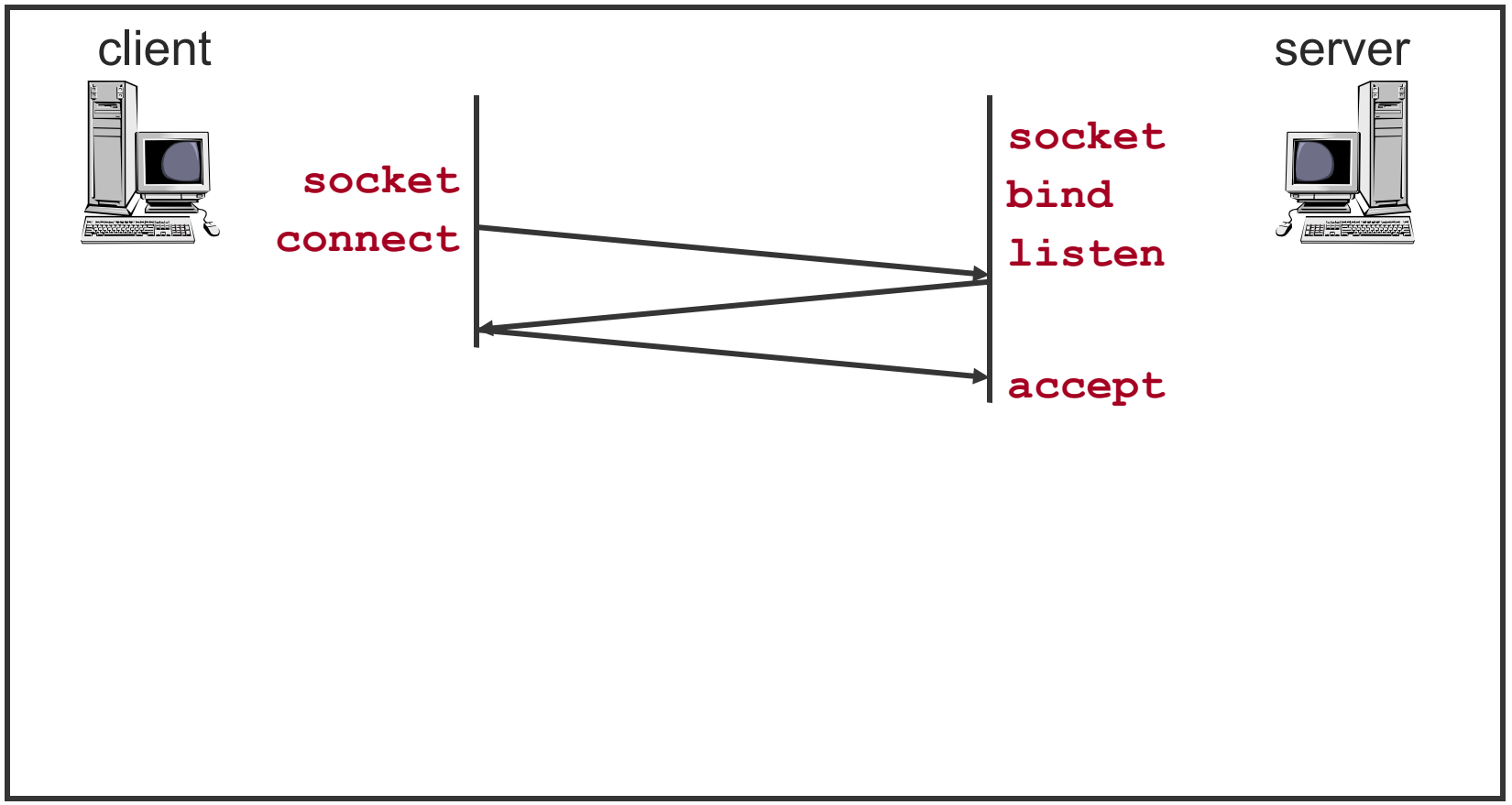
- A simplified API for accessing sockets
  - Similar to the interface from Java sockets
  - Function calls not found in Unix systems
    - But do convey concepts
- Will go over the gory details in disc.

- Programming questions on final will be at this level of abstraction

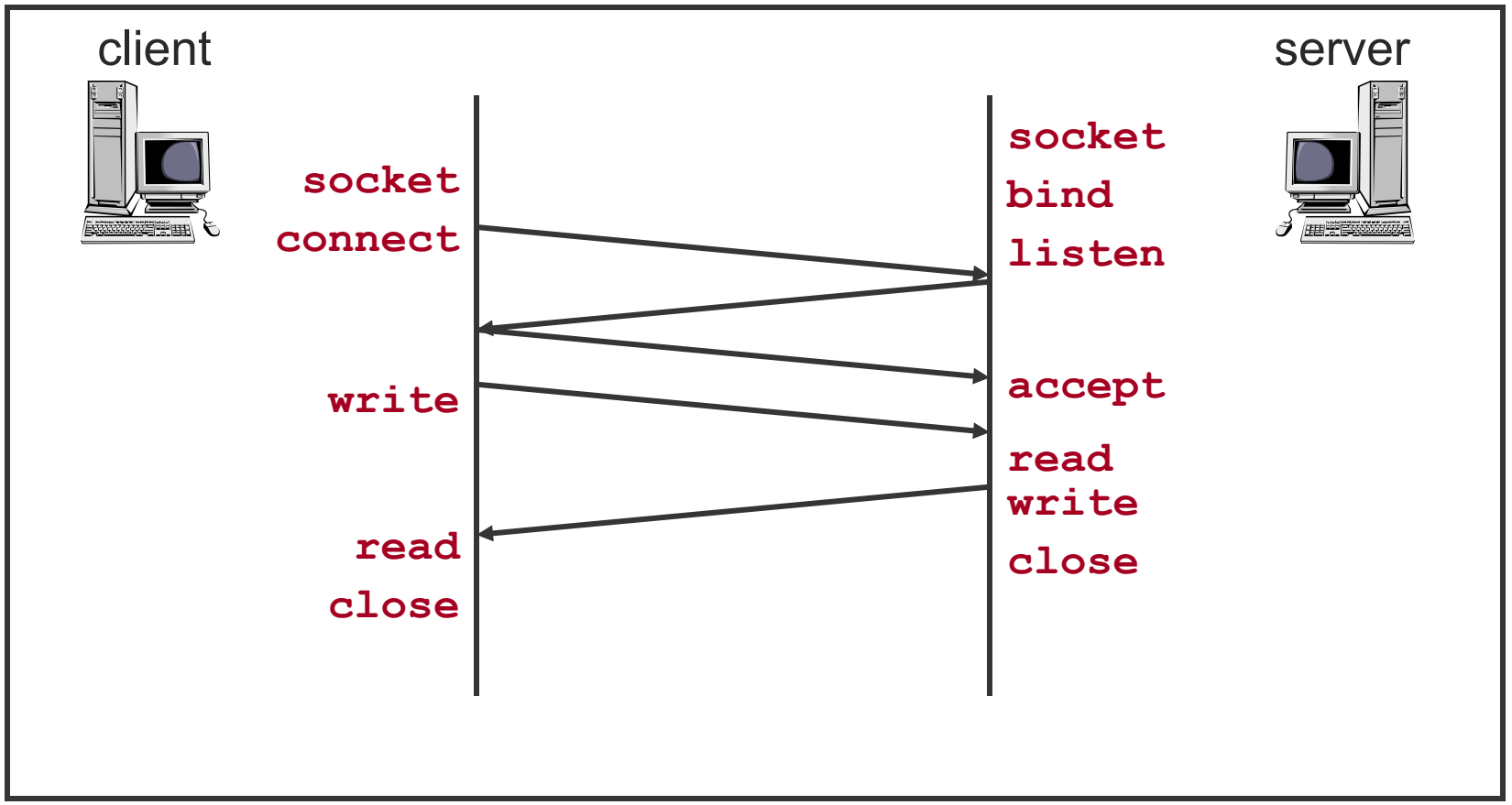
Basic Unix Concepts

- Input/Output – I/O
  - Per-process table of I/O channels
  - Table entries describe files, sockets, devices, pipes, etc.
  - Unifies I/O interface
  - Table entry/index into table called “file descriptor”
- Error Model
  - Return value
    - 0 on success, num bytes for file descriptors
    - -1 on failure
    - NULL on failure for routines returning pointers
  - `errno` variable

TCP Connection Example



TCP Connection Example



Socket TCP client

```
int NewConnection(char *inetAddr,
                  int port)
```

inetAddr – DNS name ("google.com")

port – TCP port of server (e.g., 80)

Returns a file descriptor for new socket

NewConnection behind the scenes

- Create a new socket file descriptor
- Resolve hostname into IP addr
- Connect to server

Read/write

int read(int sock, uchar *buf, uint len)

int write(int sock, uchar *buf, uint len)

buf – where data is stored

len – max bytes to read/write

returns num bytes read/write, 0 when socket closed, -1 on error

Read/write

- You should assume that read/write will process less data than you give it for sockets
  - Can play a little fast and loose with files, not with sockets

Closing connection

close(int sock)

sends any buffered bytes

disables further reads/writes on socket

notifies remote host

Socket TCP server

- int NewServerSocket(int port)
  - Port – local port to use for server
  - Returns socket file descriptor

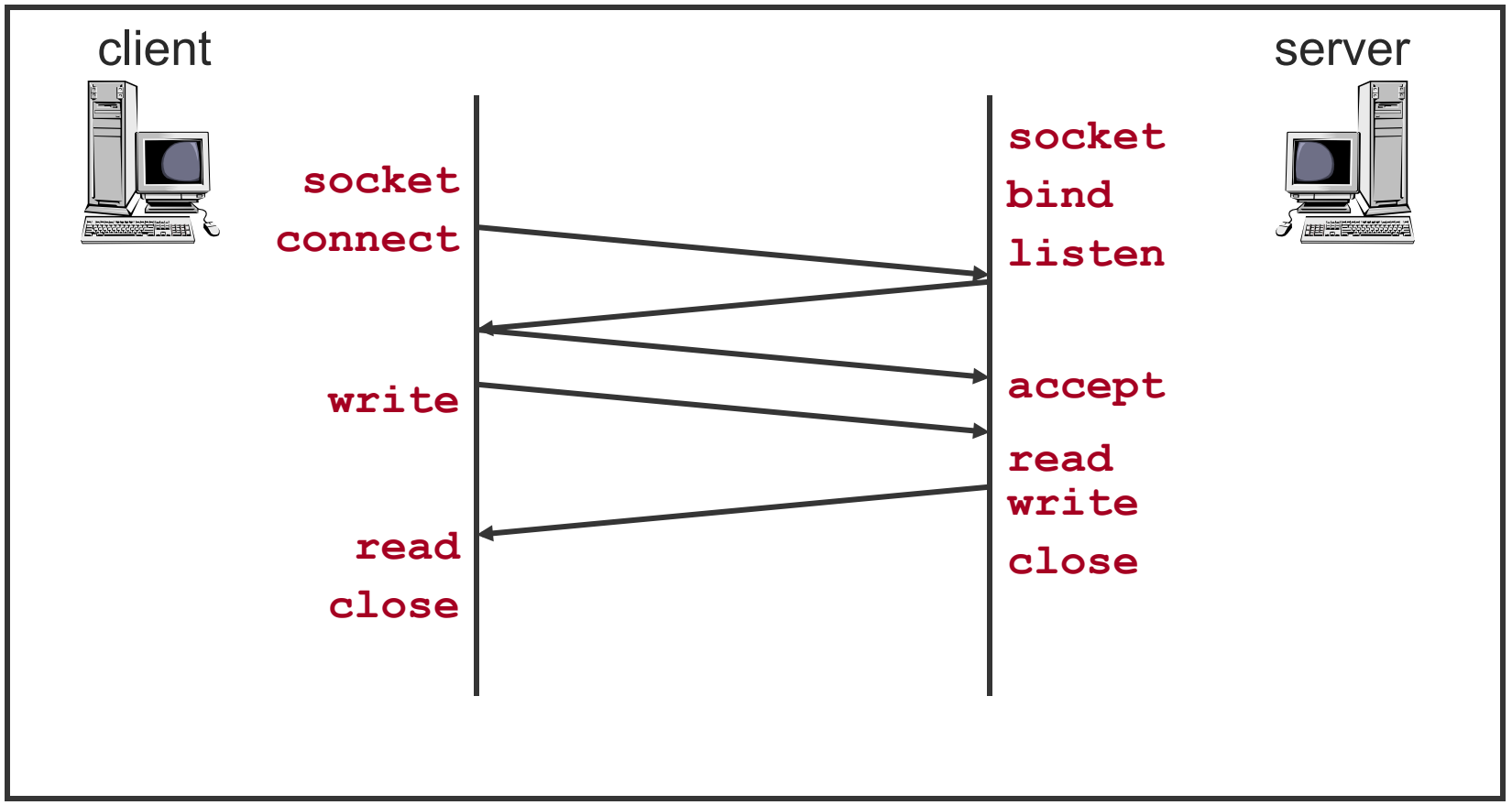
NewServerSocket

- Create a new socket fd
- “Bind” to a local port
- Setup a listen queue
  - Way of queuing up new client connections

Accept

- int accept(int serverSockFd)
  - Accept new connections on server sock
  - Returns a new sock fd when client makes new connection
  - That new sock fd is different from serverSockFd and used for client comm.

TCP Connection Example



Client

```
#define BUF_SIZE 4096
char msg[] = “hello”;
char buffer[BUF_SIZE];
int ret, bytesWritten, bytesRead;
int len = strlen(msg) + 1;
int sock;
int bufSize = BUF_SIZE;

sock = NewConnection(“localhost”, 8080);
assert(sock >= 0);
```


Client

```
while(bytesWritten < len) {
    ret = write(sock, msg + bytesWritten,
                len – bytesWritten);
    assert(ret > 0);
    bytesWritten += ret;
}

while((ret = read(sock, buffer+bytesRead,
                  bufSize – bytesRead)) > 0) {
    bytesRead += ret;
}
assert(ret == 0);
```


Server

```
#define BUF_SIZE 4096
char msg[] = “world”;
char buffer[BUF_SIZE];
int ret, bytesWritten, bytesRead;
int len = strlen(msg) + 1;
int servSock, cliSock;
int bufSize = BUF_SIZE;

servSock = NewServerSocket(8080);
assert(servSock >= 0);

cliSock = accept(servSock);
```


Server

```
while((ret = read(cliSock, buffer+bytesRead,
                  bufSize – bytesRead)) > 0) {
    bytesRead += ret;
    if(buffer[bytesRead-1] == '\0') break;
}
assert(ret > 0);

while(bytesWritten < len) {
    ret = write(cliSock, msg + bytesWritten,
                len – bytesWritten);
    assert(ret > 0);
    bytesWritten += ret;
}
```


UDP Connection Example