

# TCP Sockets Programming

- Creating a passive mode (server) socket.
- Establishing an application-level connection.
- send/receive data.
- Terminating a connection.

# Creating a TCP socket

```
int socket(int family,int type,int proto);
```

```
int sock;
```

```
sock = socket( PF_INET,  
              SOCK_STREAM,  
              0 );
```

```
if (sock<0) { /* ERROR */ }
```

# Binding to well known address

```
int mysock;  
struct sockaddr_in myaddr;  
  
mysock = socket(PF_INET, SOCK_STREAM, 0);  
myaddr.sin_family = AF_INET;  
myaddr.sin_port = htons( 80 );  
myaddr.sin_addr = htonl( INADDR_ANY );  
  
bind(mysock, &myaddr, sizeof(myaddr));
```

# Establishing a passive mode TCP socket

- Passive mode:
  - Address already determined.
  - Tell the kernel to accept incoming connection requests directed at the socket address.
  - Tell the kernel to queue incoming connections for us.

# Listen()

```
int listen( int sockfd, int backlog);
```

**sockfd** is the TCP socket (already bound to an address)

**backlog** is the number of incoming connections the kernel should be able to keep track of.

**Listen()** returns -1 on error (otherwise 0).

# Accept ( )

```
int accept( int sockfd,  
           struct sockaddr* cliaddr,  
           socklen_t *addrlen);
```

**sockfd** is the passive mode TCP socket.

**cliaddr** is a pointer to allocated space.

**addrlen** is a value-result argument, must be set to the size of cliaddr, will be set on return to be the number of bytes in cliaddr set by the call to accept.

## **accept ( ) return value**

**Accept ( )** returns a new socket descriptor (small positive integer) or -1 on error.

After **accept** returns a new socket descriptor I/O can be done using the **read()** and **write()** system calls.

**read()** and **write()** operate a little differently on sockets (vs. file operation)!

# Terminating a TCP connection

- Either end of the connection can call the `close()` system call.
- If the other end had closed the connection and there is no buffered data, reading from a TCP socket returns EOF (0 bytes).

# Client Code

- TCP clients can call `connect()` which:
  - takes care of establishing an endpoint address for the client socket (we don't need to call `bind`).
  - Attempts to establish a connection to the specified server.

# connect ( )

```
int connect( int sockfd,  
            const struct sockaddr *server,  
            socklen_t addrlen);
```

`sockfd` is an already created TCP socket.

`server` contains the address of the server (IP  
Address and TCP port number)

`connect()` returns 0 if OK, -1 on error

# Reading from a TCP socket

```
int read( int fd, char *buf, int max);
```

- By default read() will block until data is available\*
- reading from a TCP socket may return less than max bytes (whatever is available).
- You must be prepared to read data 1 byte at a time!

# Writing to a TCP socket

```
int write( int fd, char *buf, int num);
```

- write might not be able to write all num bytes (on a nonblocking socket).
- The book includes readn(), writen() and readline() function definitions.