

## TCP Sockets Programming

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

Netprog 2001 TCP Sockets

1

---

---

---

---

---

---

---

---

## 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 */ }
```

Netprog 2001 TCP Sockets

2

---

---

---

---

---

---

---

---

## 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, (sockaddr *) &myaddr,
      sizeof(myaddr));
```

Netprog 2001 TCP Sockets

3

---

---

---

---

---

---

---

---

## Establishing a passive mode TCP socket

Passive mode:

- Address already determined.
- Tell the kernel to accept incoming connection requests directed at the socket address.
  - **3-way handshake**
- Tell the kernel to queue incoming connections for us.

Netprog 2001 TCP Sockets

4

---

---

---

---

---

---

---

---

## 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 (queue for us).

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

Netprog 2001 TCP Sockets

5

---

---

---

---

---

---

---

---

## Accepting an incoming connection.

- Once we call **listen()**, the O.S. will queue incoming connections
  - Handles the 3-way handshake
  - Queues up multiple connections.
- When our application is ready to handle a new connection, we need to ask the O.S. for the next connection.

Netprog 2001 TCP Sockets

6

---

---

---

---

---

---

---

---

## 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**
- on return, will be set to be the number of used bytes in **cliaddr**.

Netprog 2001 TCP Sockets

7

---

---

---

---

---

---

---

---

## 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)!

Netprog 2001 TCP Sockets

8

---

---

---

---

---

---

---

---

## Terminating a TCP connection

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

Netprog 2001 TCP Sockets

9

---

---

---

---

---

---

---

---

## Client Code

- TCP clients can call `connect()` which:
  - takes care of establishing an endpoint address for the client socket.
    - don't need to call `bind` first, the O.S. will take care of assigning the local endpoint address (TCP port number, IP address).
  - Attempts to establish a connection to the specified server.
    - **3-way handshake**

Netprog 2001 TCP Sockets

10

---

---

---

---

---

---

---

---

## `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

Netprog 2001 TCP Sockets

11

---

---

---

---

---

---

---

---

## 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!

Netprog 2001 TCP Sockets

12

---

---

---

---

---

---

---

---

## 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.

Netprog 2001 TCP Sockets

13

---

---

---

---

---

---

---

---

## Metaphor for Good Relationships

Copyright Dr. Laura's Network Programming Corp.

To succeed in relationships:

- you need to establish your own identity. *bind()*
- you need to be open & accepting. *accept()*
- you need to establish contacts. *connect()*
- you need to take things as they come, not as you expect them. *read might return 1 byte*
- you need to handle problems as they arise. *check for errors*

Netprog 2001 TCP Sockets

14

---

---

---

---

---

---

---

---