Elementary TCP Sockets

Chapter 4

UNIX Network Programming
Vol. 1, Second Ed. Stevens
Figure 2.16

Networks: TCP/IP sockets
TCP socket calls

Server
- `socket()`
- `bind()`
- `listen()`
- `accept()` blocks until server receives a connect request from client

Client
- `socket()`
- `connect()` connect negotiation
- `write()`
- `read()` data
- `write()`
- `read()` data
- `close()`
UDP socket calls

Server

1. **socket()**
2. **bind()**
3. **recvfrom()**
   - Blocks until server receives data from client
4. **sendto()**
5. **close()**

Client

1. **socket()**
2. **bind()**
3. **sendto()**
4. **recvfrom()**
5. **close()**

---

**Figure 2.18**

Leon-Garcia & Widjaja: *Communication Networks*

Copyright ©2000 The McGraw Hill Companies

Networks: TCP/IP sockets
System Calls for Elementary TCP Sockets

```c
#include <sys/types.h>
#include <sys/socket.h>

socket Function
int socket ( int family, int type, int protocol );
```

family: specifies the protocol family {AF_INET for TCP/IP}
type: indicates communications semantics

- SOCK_STREAM  stream socket  TCP
- SOCK_DGRAM   datagram socket  UDP
- SOCK_RAW     raw socket

protocol: set to 0 except for raw sockets

returns on success:  socket descriptor  {a small nonnegative integer}
on error: -1

Example:
If (( sockfd = socket (AF_INET, SOCK_STREAM, 0)) < 0)
    err_sys ("socket call error");
**connect Function**

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

- **sockfd**: a socket descriptor returned by the socket function
- ***servaddr**: a pointer to a socket address structure
- **addrlen**: the size of the socket address structure

The socket address structure must contain the *IP address* and the *port number* for the connection wanted.

In TCP **connect** initiates a three-way handshake. **connect** returns when the connection is established or when an error occurs.

**returns on success**: 0

**on error**: -1

**Example:**

```c
if ( connect (sockfd, (struct sockaddr *) &servaddr, sizeof (servaddr)) != 0)
    err_sys("connect call error");
```
**bind Function**

`int bind (int sockfd, const struct sockaddr *myaddr, socklen_t addrlen);`

*bind* assigns a local protocol address to a socket.

*protocol address:* a 32 bit IPv4 address + a 16 bit TCP or UDP port number.

*sockfd:* a socket descriptor returned by the socket function.

*myaddr:* a pointer to a protocol-specific address.

*addrlen:* the size of the socket address structure.

*Servers bind* their “well-known port” when they start.

**returns**

*on success:* 0

*on error:* -1

**Example:**

If `(bind (sd, (struct sockaddr *) &servaddr, sizeof (servaddr)) != 0)`

`errsys ("bind call error");`
listen Function

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

Listen is called **only** by a TCP server and performs two actions:
1. Converts an unconnected socket into a passive socket.
2. Specifies the maximum number of connections that the kernel should queue for this socket.

**returns on success:** 0
**on error:** -1

**Example:**
If (listen (sd, 2) != 0)
    errsys ("listen call error");
accept Function

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

accept is called by the TCP server to return the next completed connection from the front of the completed connection queue.

sockfd: this is the same socket descriptor as in listen call.

*cliaddr: used to return the protocol address of the connected peer process (i.e., the client process).

*addrlen: {this is a value-result argument}

before the accept call: we set the integer value pointed to by *addrlen to the size of the socket address structure pointed to by cliaddr;

on return from accept call: this integer value contains the actual number of bytes stored in the socket address structure.

returns on success: a new socket descriptor

on error: -1
For `accept` the first argument `sockfd` is the **listening socket** and the returned value is the **connected socket**.

The server will have one connected socket for each client connection accepted.

When the server is finished with a client, the connected socket **must** be closed.

**Example:**
```c
sfd = accept (s, NULL, NULL);
if (sfd == -1) err_sys ("accept error");
```
close Function

```c
int close (int sockfd);
```

close marks the socket as closed and returns to the process immediately.

sockfd this socket descriptor is no longer useable.

Note – TCP will try to send any data already queued to the other end before the normal connection termination sequence.

Returns on success: 0
on error: -1

Example:

close (s);