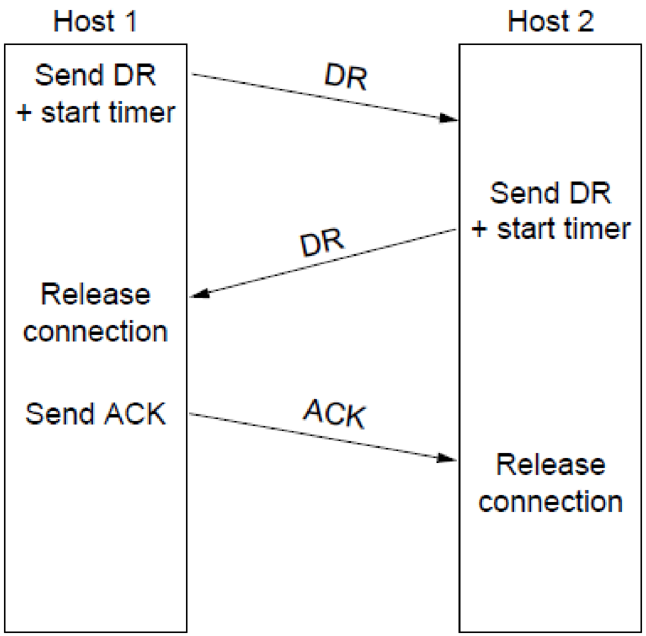
Transport Layer Contd

Internet Technologies
COMP90007

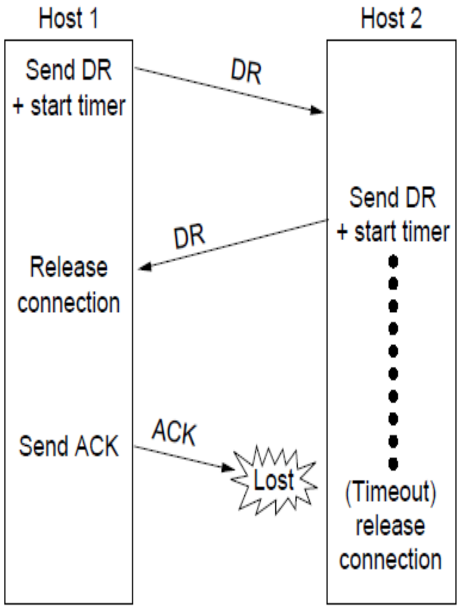
Strategies for Connection Release

- No perfect solution but there is a practical/usable solution that works most of the time
- 3 way handshake
- Finite retry
- Timeouts
- Normal release sequence, initiated by transport user on Host 1
  - DR=Disconnect Request
  - Both DRs are ACKed by the other side

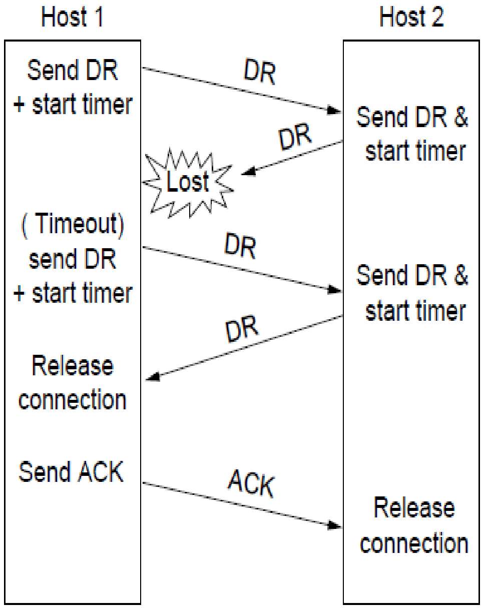


Connection Release (Error Cases)

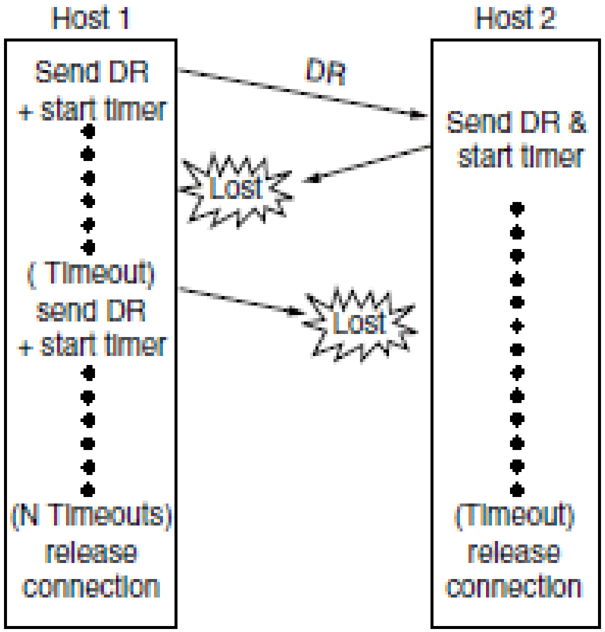
- Error cases are handled with timers and retransmission



Final ACK lost, Host 2 times out



Lost DR causes retransmissions

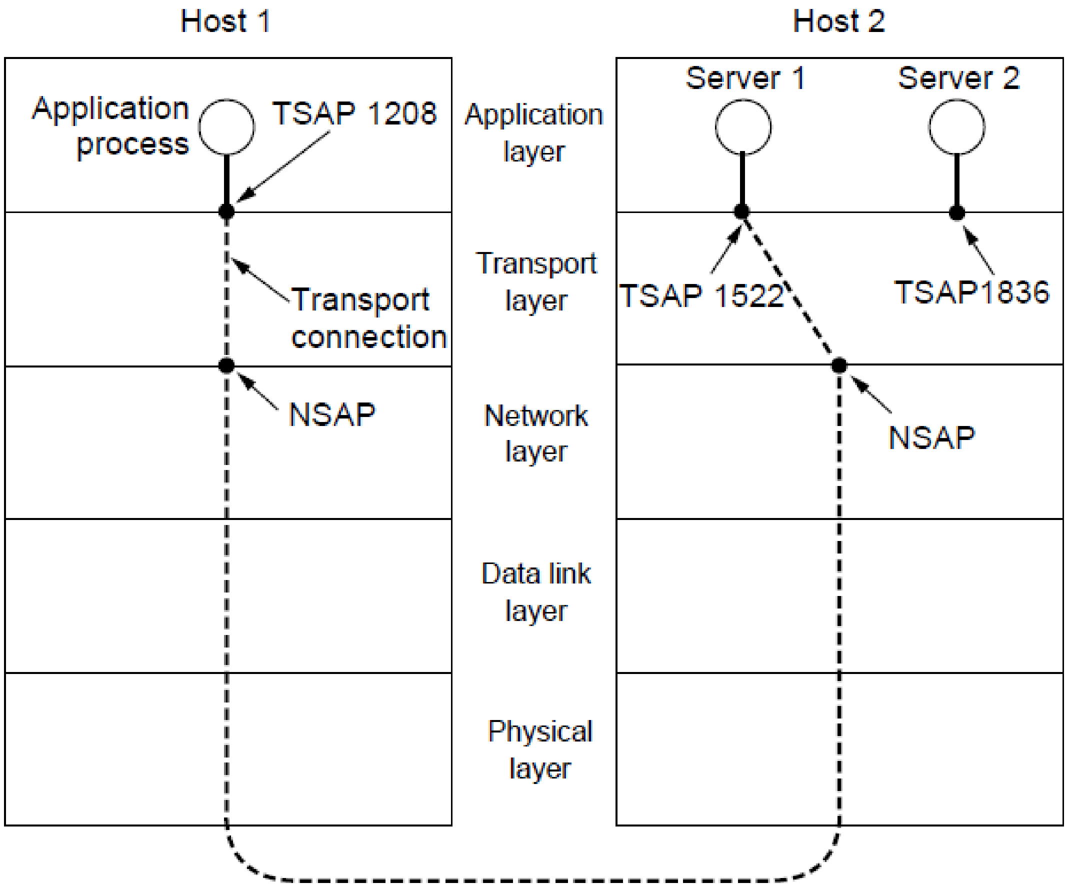


Extreme: Many lost DRs cause both hosts to timeout

Transport Layer Addressing

- Specification of **remote process to connect to** is required at application and transport layers
- Addressing in transport layer is typically done using **Transport Service Access Points** (TSAPs)
  - on the Internet, a TSAP is commonly referred to as a port (e.g. ***port*** 80)
- Addressing in the network layer is typically done using **Network Service Access Points** (NSAPs)
  - on the Internet, the concept of an NSAP is commonly interpreted as simply an ***IP address***

TSAPs, NSAPs and Transport Layer Connections Illustrated

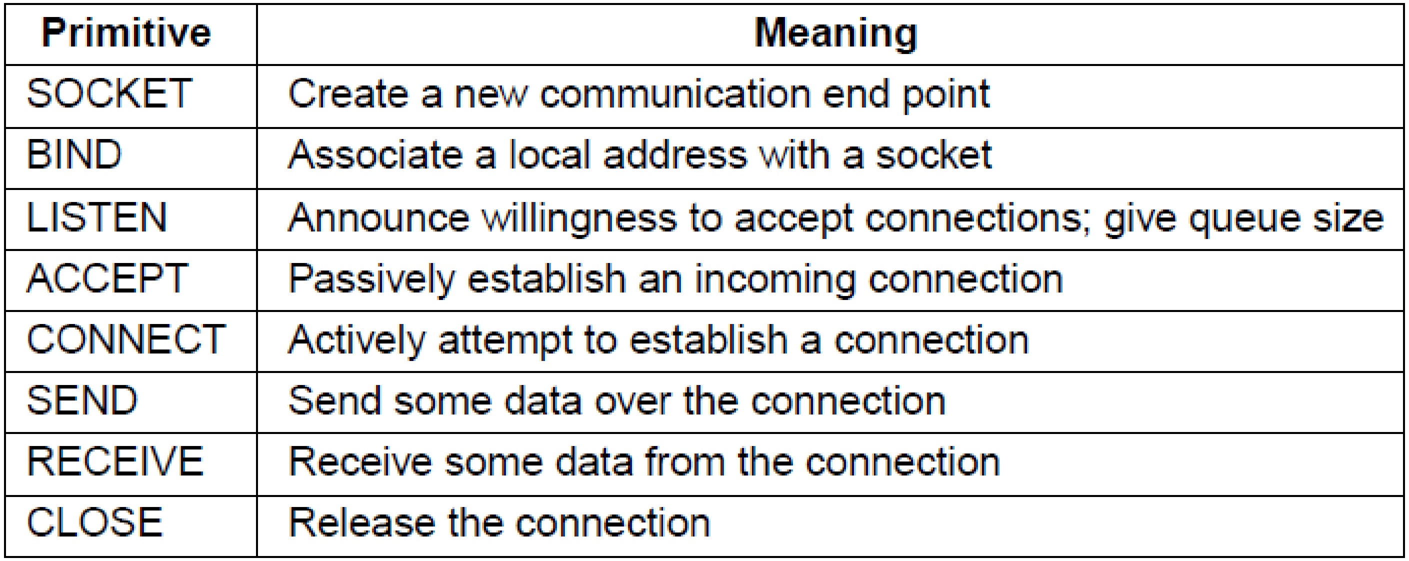


Types of TSAP Allocation

1. Static
   - Well known services have standard allocated TSAPs/ports, which are embedded in OS
2. Directory Assistance – Port-mapper
   - A new service must register itself with the portmapper, giving both its service name and TSAP
3. Mediated
   - A process server intercepts inbound connections and spawns requested server and attaches inbound connection

Programming Basics with Sockets

- Sockets widely used for interconnections
  - Berkeley sockets are predominant in Internet applications today
  - Sockets are transport endpoints that help with programming etc
  - Recall the simple set we saw earlier for this, now add to that SOCKET, BIND, and ACCEPT
  - These are at OS level and are called from a programming API

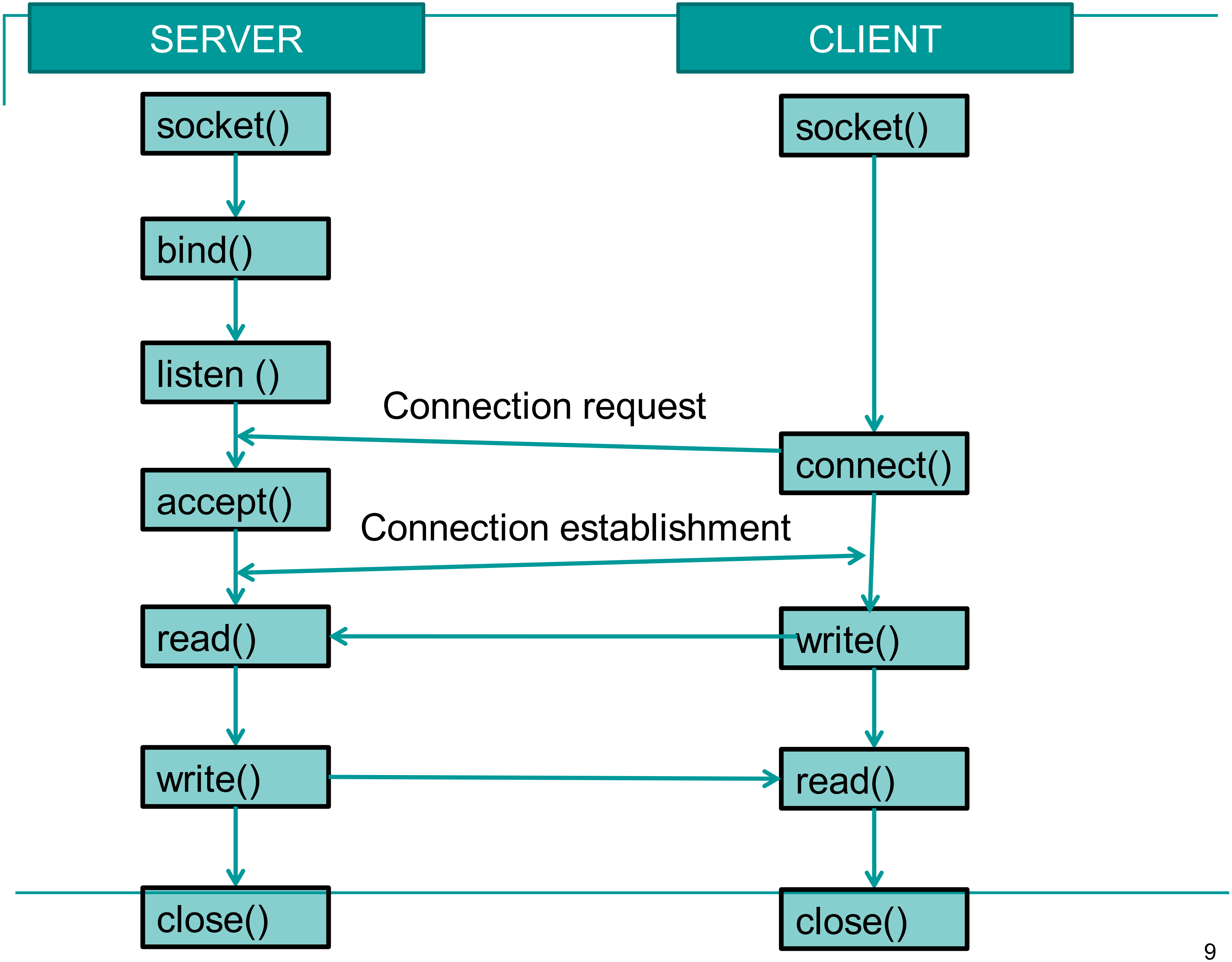


Primitive	Meaning
SOCKET	Create a new communication end point
BIND	Associate a local address with a socket
LISTEN	Announce willingness to accept connections; give queue size
ACCEPT	Passively establish an incoming connection
CONNECT	Actively attempt to establish a connection
SEND	Send some data over the connection
RECEIVE	Receive some data from the connection
CLOSE	Release the connection

Recall Our Example Code

```
Socket A_Socket = createSocket("TCP");

connect(A_Socket, 128.255.16.0, 80);

send(A_socket, “My first message!”);

disconnect(A_socket);
```

- *These call OS services defined in the table*
- *There is also a server component for this client that runs on another host…*



SERVER

socket()

bind()

listen ()

accept()

read()

write()

close()

CLIENT

socket()

connect()

write()

read()

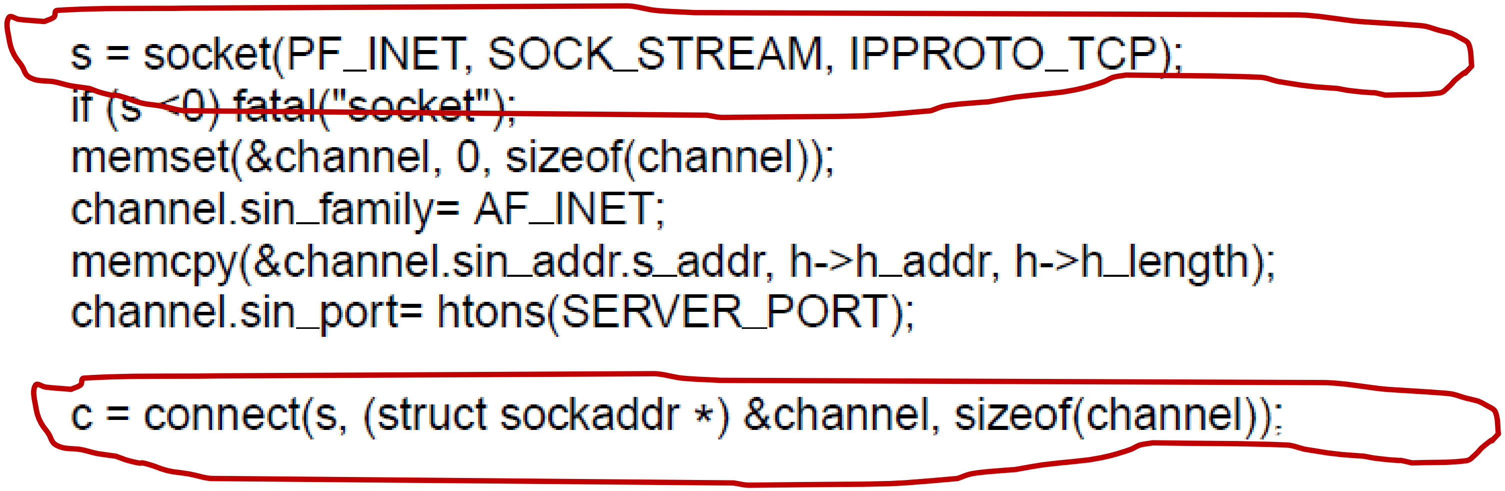
close()

Connection request

Connection establishment

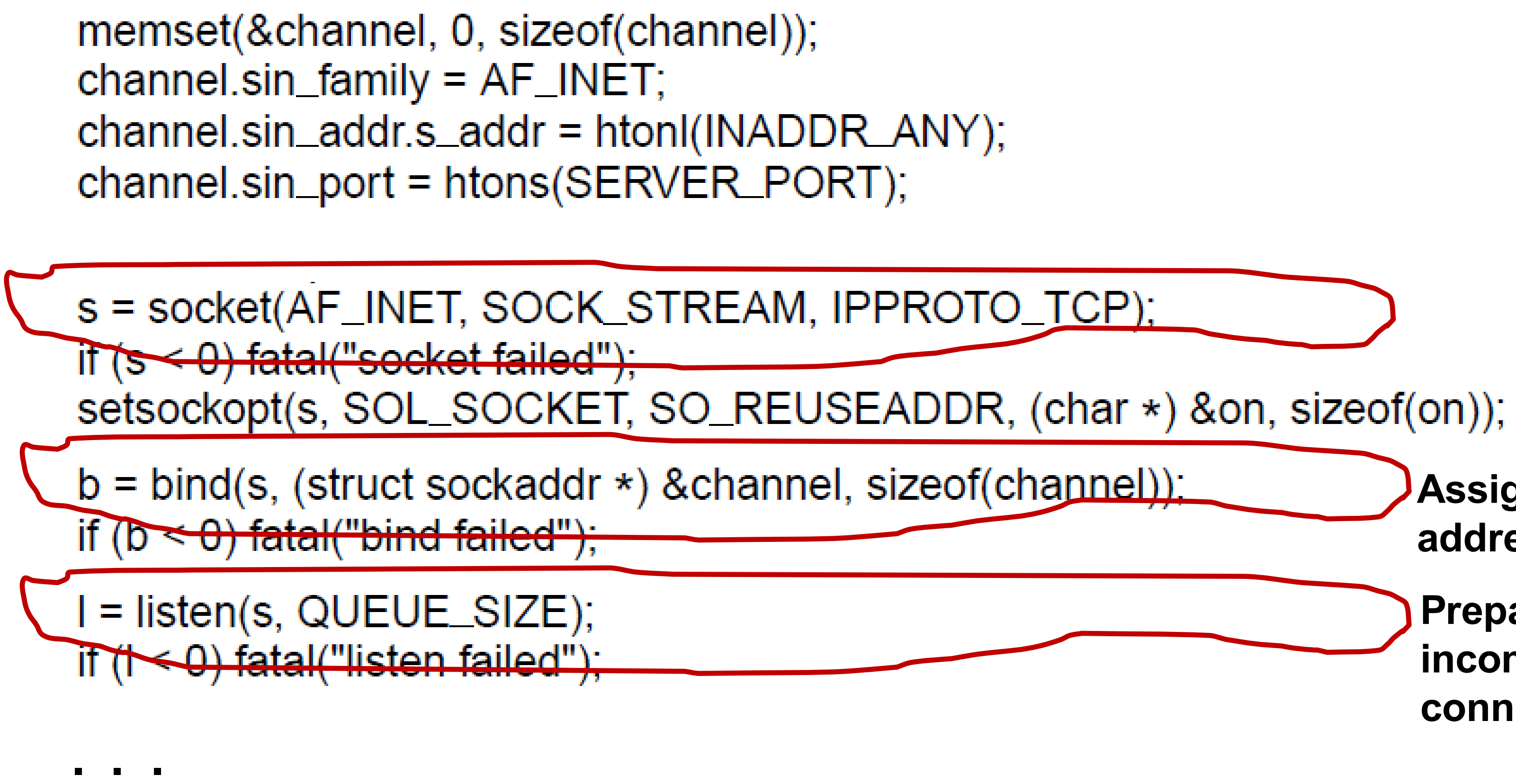
Lets Look at the Code from the book as well

Example from the book has more details and the code is written in a different language but the essence is the same… This is way we program with Sockets in most languages…



```
s = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
if (s <0) fatal("socket");
memset(&channel, 0, sizeof(channel));
channel.sin_family= AF_INET;
memcpy(&channel.sin_addr.s_addr, h->h_addr, h->h_length);
channel.sin_port= htons(SERVER_PORT);

c = connect(s, (struct sockaddr *) &channel, sizeof(channel));
```

Example Contd. with Your First Server Code Example

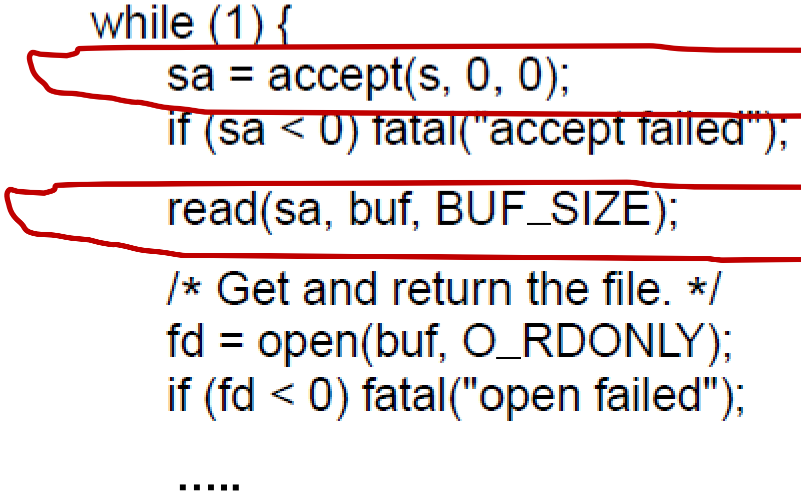


```
memset(&channel, 0, sizeof(channel));
channel.sin_family = AF_INET;
channel.sin_addr.s_addr = htonl(INADDR_ANY);
channel.sin_port = htons(SERVER_PORT);

s = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
if (s < 0) fatal("socket failed");
setsockopt(s, SOL_SOCKET, SO_REUSEADDR, (char *) &on, sizeof(on));

b = bind(s, (struct sockaddr *) &channel, sizeof(channel));
if (b < 0) fatal("bind failed");

l = listen(s, QUEUE_SIZE);
if (l < 0) fatal("listen failed");

. . .
```

Assign address

Prepare for incoming connections

Server Code Contd



```
while (1) {
        sa = accept(s, 0, 0);
        if (sa < 0) fatal("accept failed");
        read(sa, buf, BUF_SIZE);
        /* Get and return the file. */
        fd = open(buf, O_RDONLY);
        if (fd < 0) fatal("open failed");

         …..
```

Block waiting for the next connection

Read (receive) request

But the server can also create a new thread to handle the connection on a new socket and go back to waiting for the next connection on the original socket…

An Example on Multi-Threading

```
ServerSocket serverSocket = new ServerSocket([parameters]);

While (true) {
          Socket socket = serverSocket.accept();
          MultiThreadMyServer server = new MultiThreadMyServer();
          server.setMyService([some more parameters]);
          server.setSocket(socket);
          new Thread(server).start();
       ....
```

- *(Code from OO Programming with Java; Chp. 14)*

More background on threading

```
class MultiThreadMyServer extends Thread {
        int somedata;
        MultiThreadMyServer() {
            this.somedata = …;
            …
        }

            … more methods here

        public void run() {
             . . .
        }
    }
```

Service Without Connection

```
public static void main(String args[]) {
        ….
        DatagramSocket mySocket = new
                DatagramSocket();
        mySocket.send([data,address, etc
                parameters]);
        …
}
```

Server Side in This Case

```
public static void main(String args[]) {
        ….
        DatagramSocket server = new
                DatagramSocket(port);
        …
        server.receive([parameters]);
        …
}
```