TCP

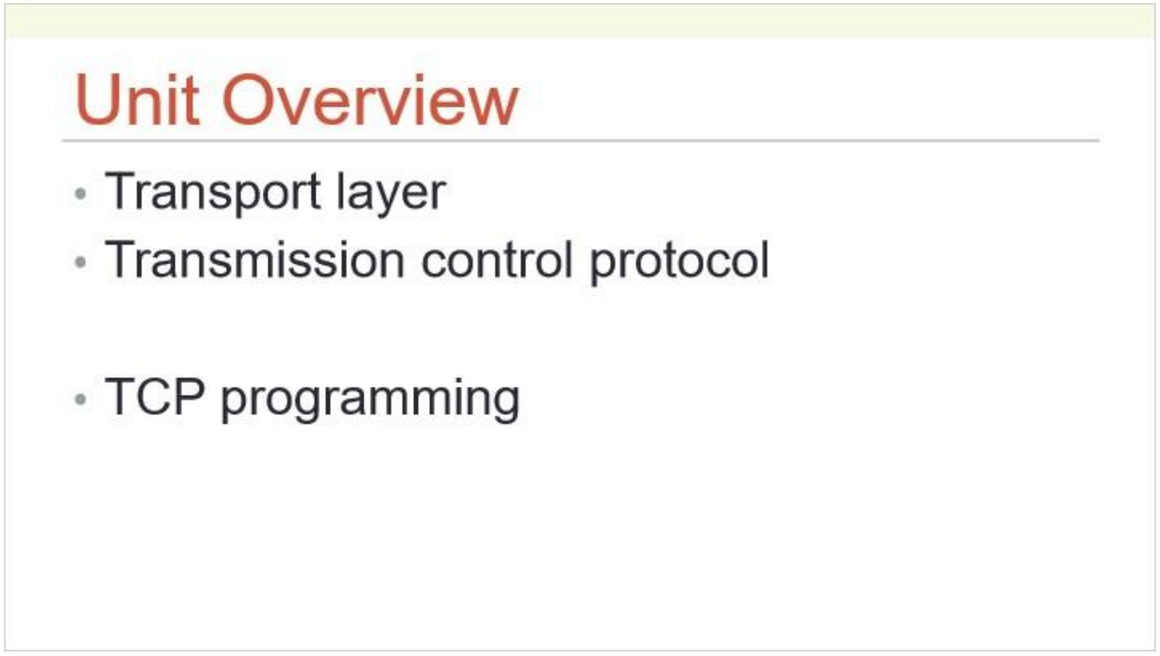
1. Transmission Control Protocol

1.1 CSCI 330

CSCI 330
UNIX and Network Programming

TCP
Transmission Control Protocol

1.2 Unit Overview

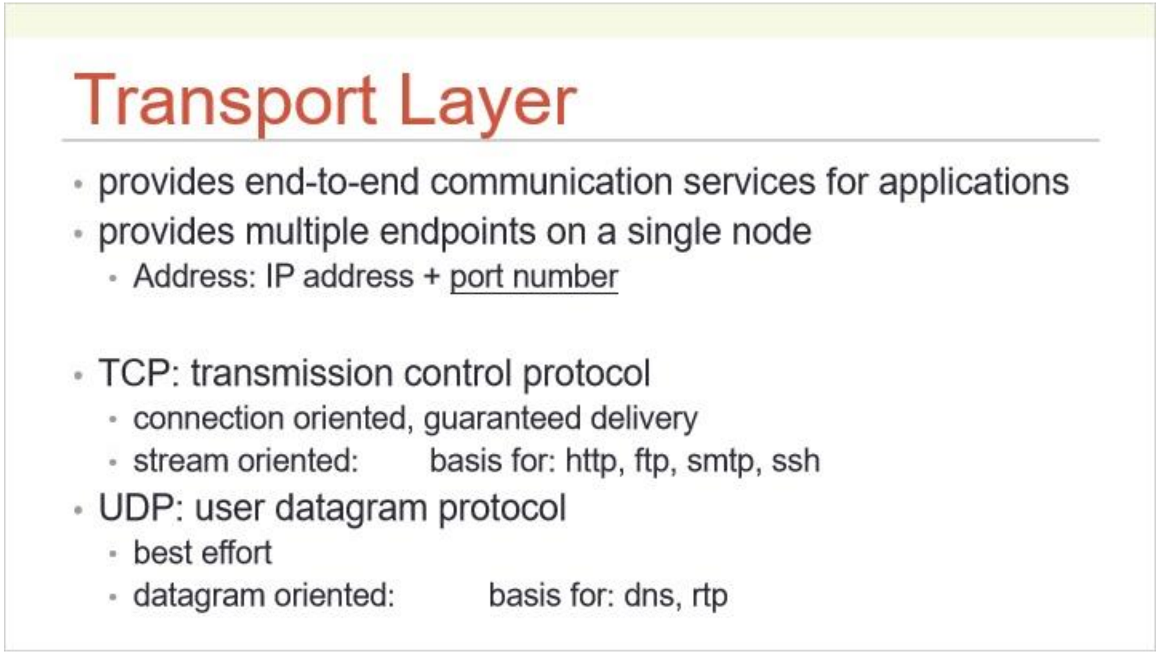


Unit Overview

- Transport layer
- Transmission control protocol

- TCP programming

1.3 Transport Layer

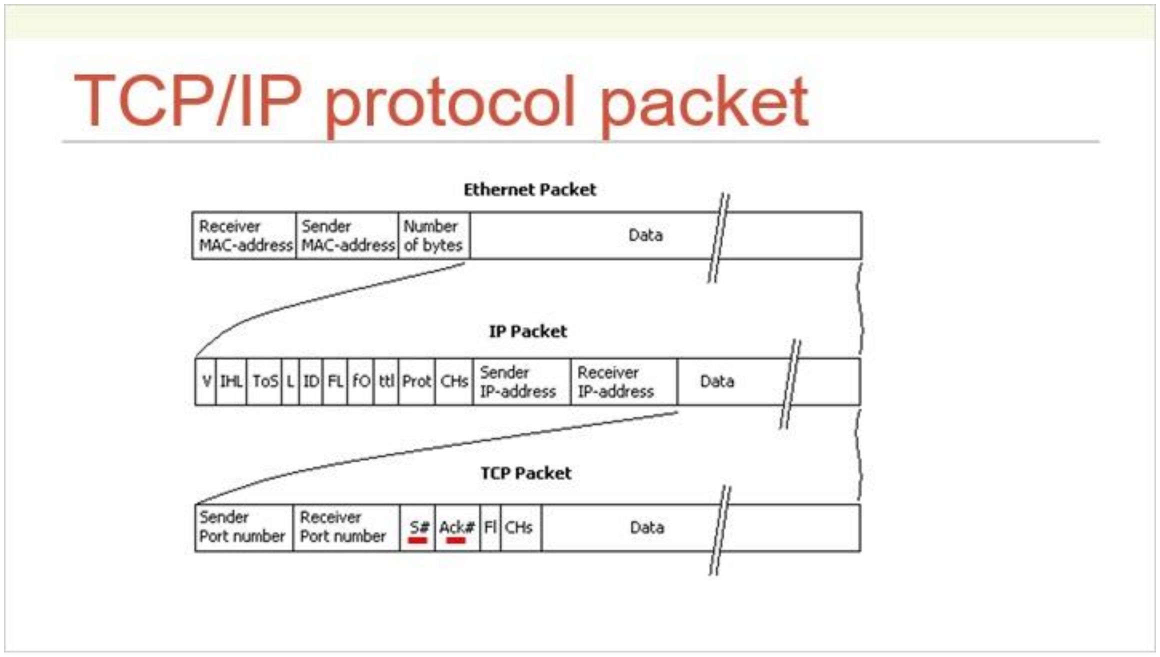


Transport Layer

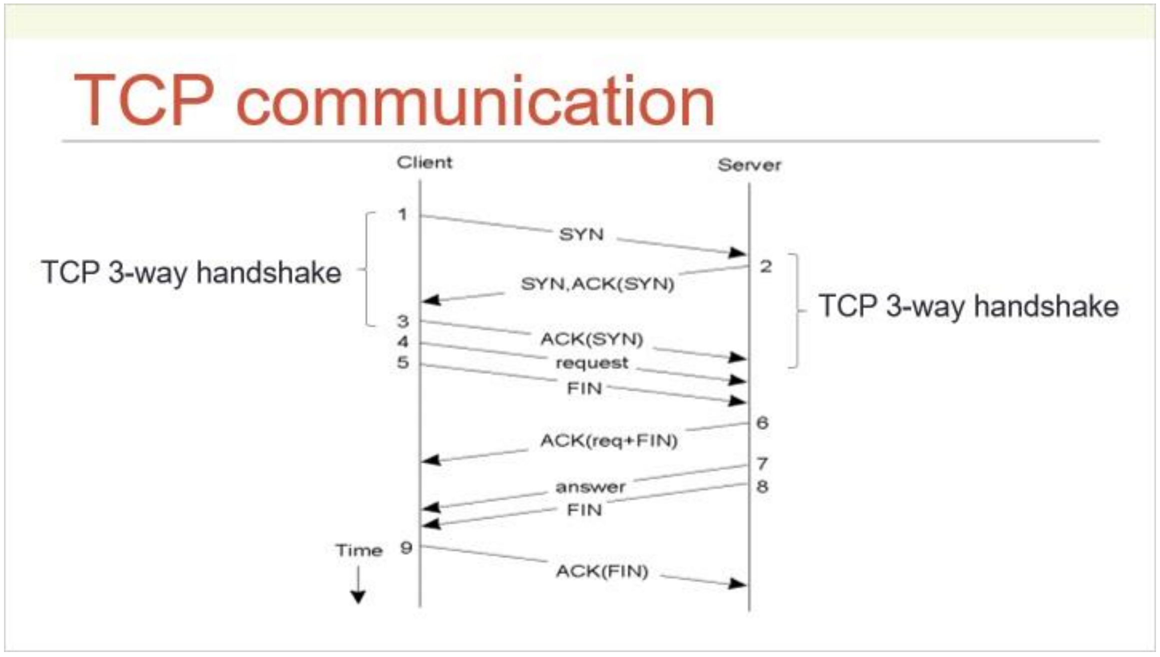
- provides end-to-end communication services for applications
- provides multiple endpoints on a single node
  - Address: IP address + port number

- TCP: transmission control protocol
  - connection oriented, guaranteed delivery
  - stream oriented: basis for: http, ftp, smtp, ssh
- UDP: user datagram protocol
  - best effort
  - datagram oriented: basis for: dns, rtp

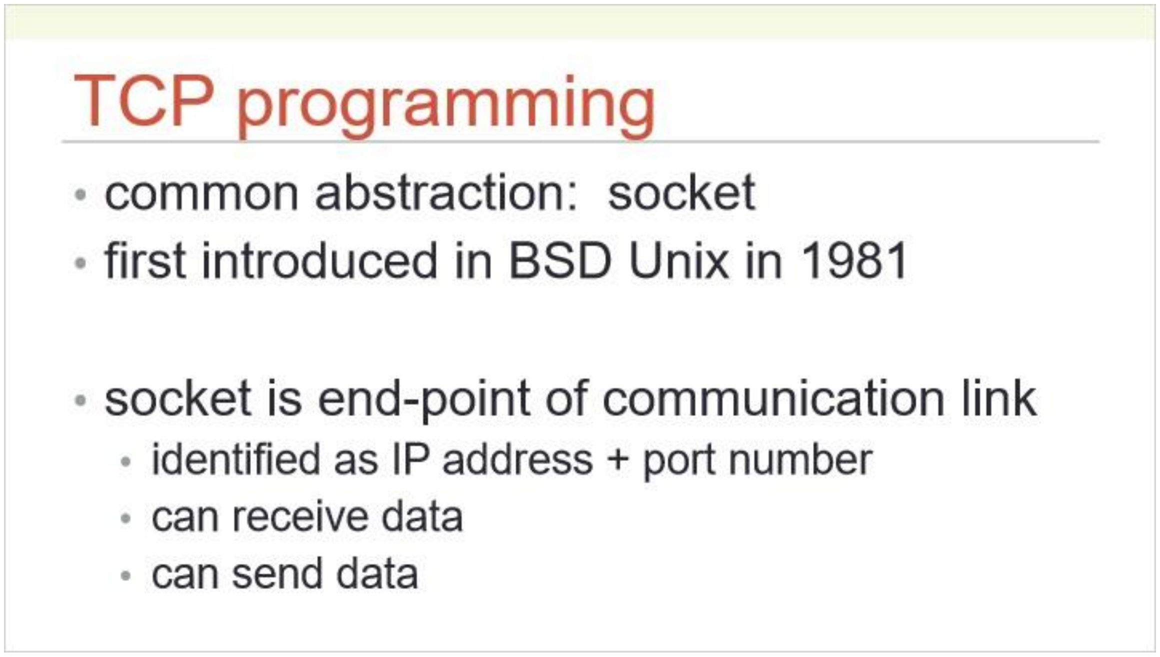
1.4 TCP/IP protocol packet


TCP/IP protocol packet

1.5 TCP communication


TCP communication

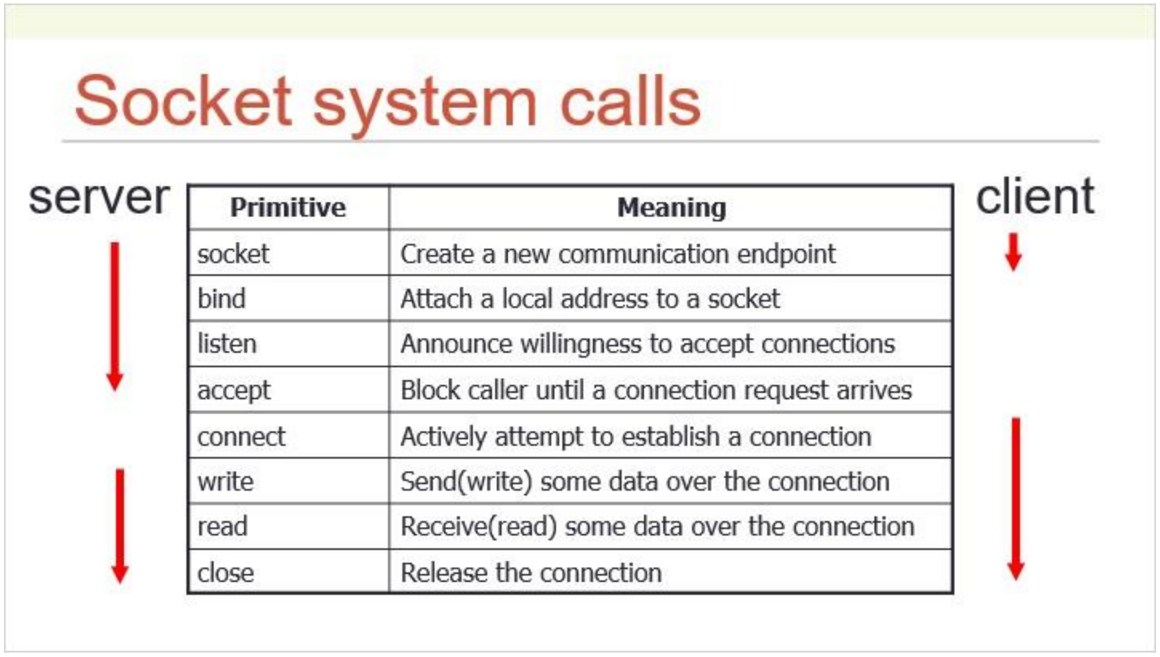
1.6 TCP programming



TCP programming

- common abstraction: socket
- first introduced in BSD Unix in 1981
- socket is end-point of communication link
  - identified as IP address + port number
  - can receive data
  - can send data

1.7 Socket system calls

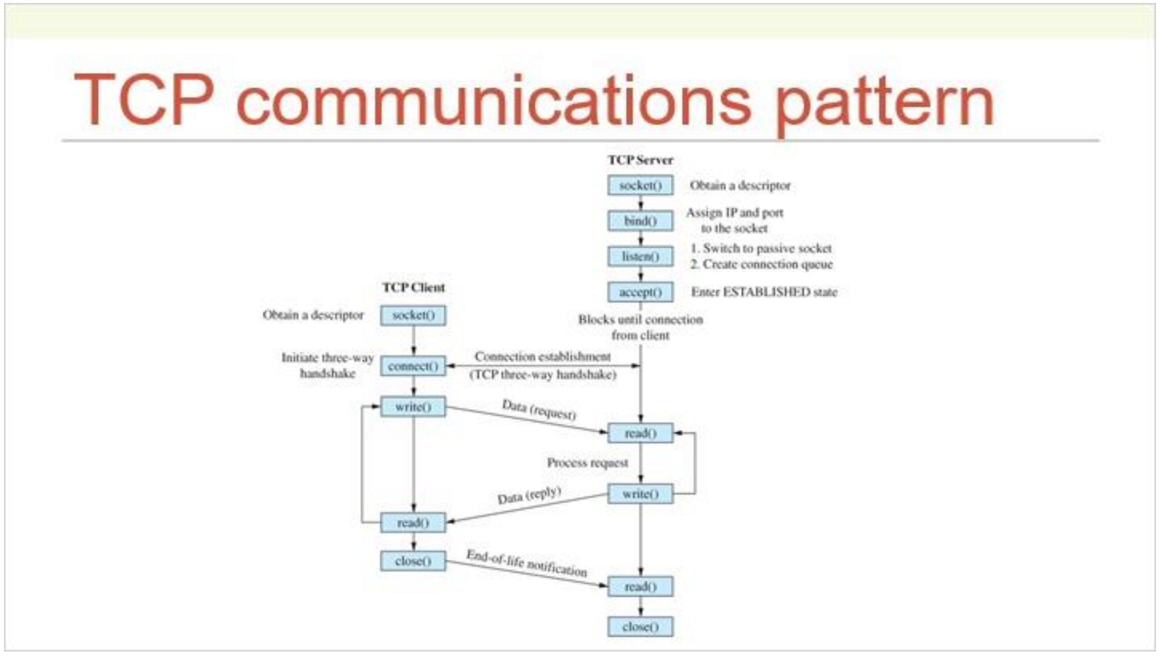


Socket system calls

server
client

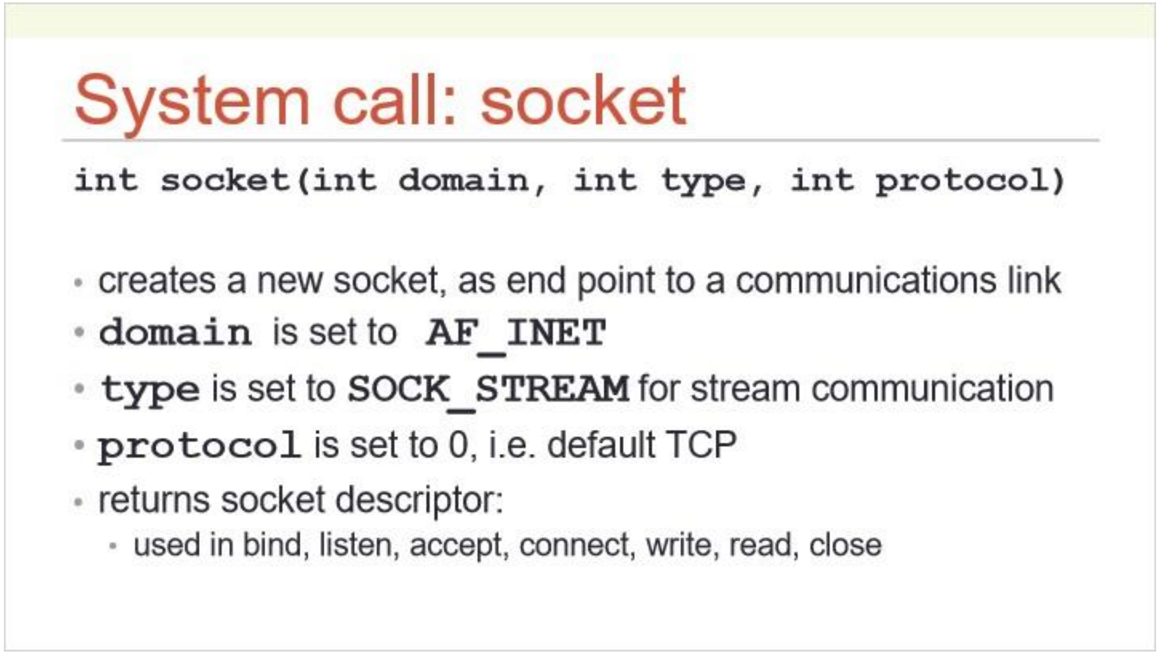
Primitive	Meaning
socket	Create a new communication endpoint
bind	Attach a local address to a socket
listen	Announce willingness to accept connections
accept	Block caller until a connection request arrives
connect	Actively attempt to establish a connection
write	Send(write) some data over the connection
read	Receive(read) some data over the connection
close	Release the connection

1.8 TCP communications pattern



TCP communications pattern

1.9 System call: socket

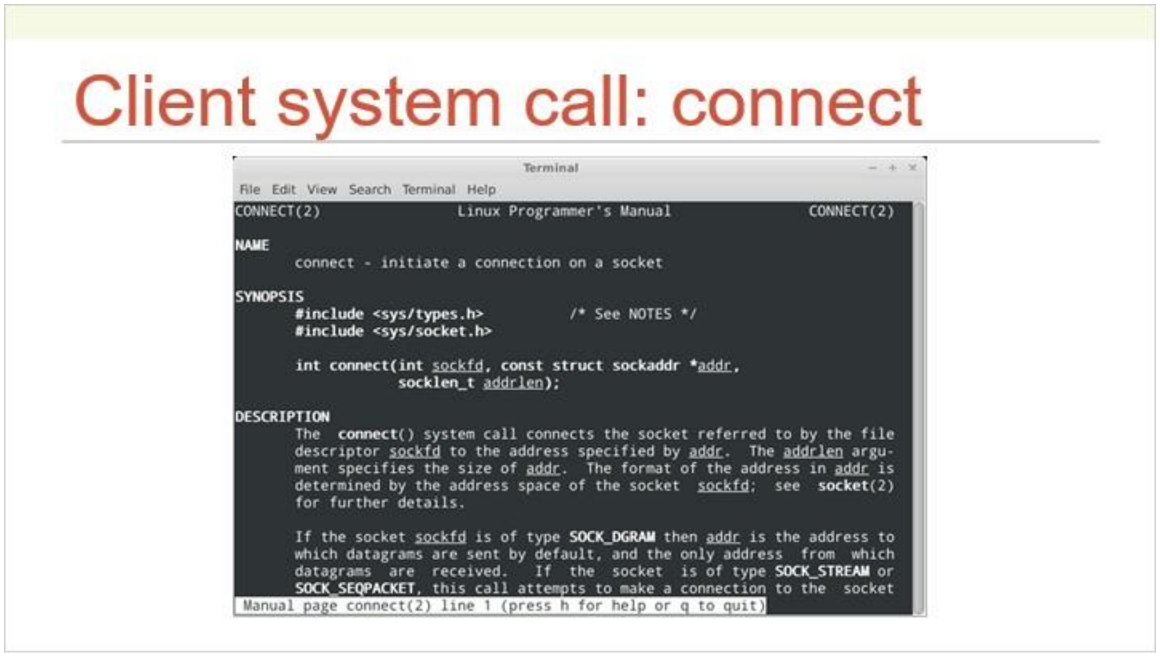


System call: socket

```
int socket(int domain, int type, int protocol)
```

- creates a new socket, as end point to a communications link
- **`domain`** is set to **`AF_INET`**
- **`type`** is set to **`SOCK_STREAM`** for stream communication
- **`protocol`** is set to 0, i.e. default TCP
- returns socket descriptor:
  - used in bind, listen, accept, connect, write, read, close

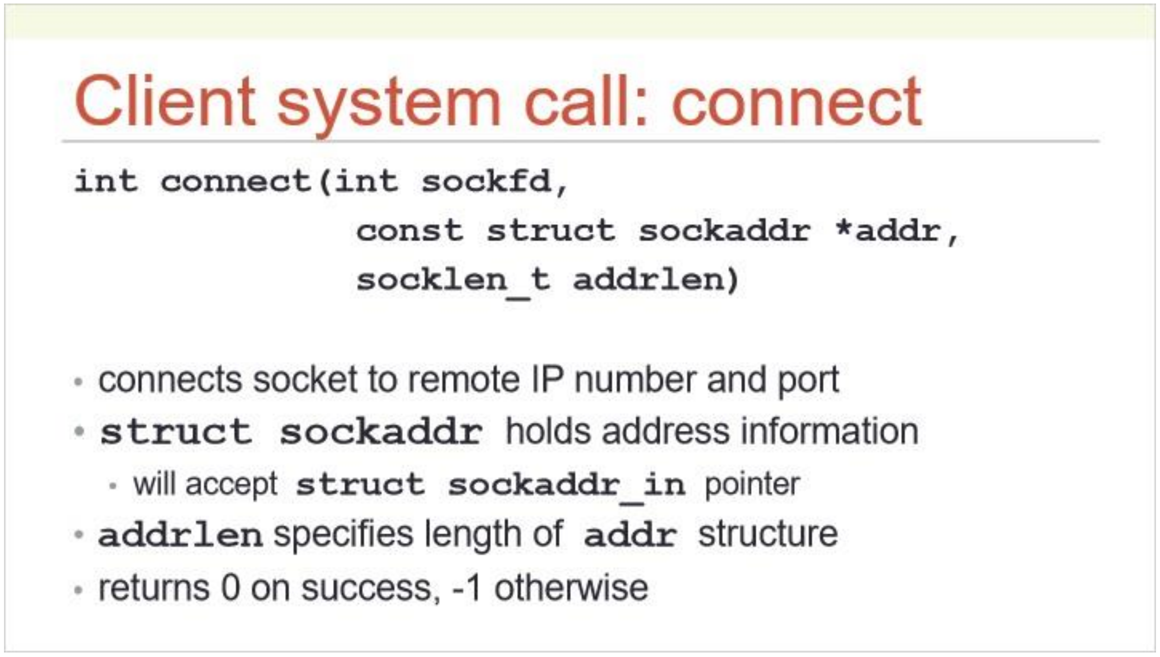
1.10 Client system call: connect



Client system call: connect

```
CONNECT(2)              Linux Programmer's Manual             CONNECT(2)

NAME
       connect - initiate a connection on a socket

SYNOPSIS
       #include <sys/types.h>          /* See NOTES */
       #include <sys/socket.h>

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

DESCRIPTION
       The  connect() system call connects the socket referred to by the file
       descriptor sockfd to the address specified by addr.  The addrlen argu-
       ment specifies the size of addr.  The format of the address in addr is
       determined by the address space of the socket  sockfd;  see  socket(2)
       for further details.

       If the socket sockfd is of type SOCK_DGRAM then addr is the address to
       which datagrams are sent by default, and the only address  from  which
       datagrams  are  received.   If  the  socket  is of type SOCK_STREAM or
       SOCK_SEQPACKET, this call attempts to make a connection to the  socket
 Manual page connect(2) line 1 (press h for help or q to quit)
```

1.11 Client system call: connect

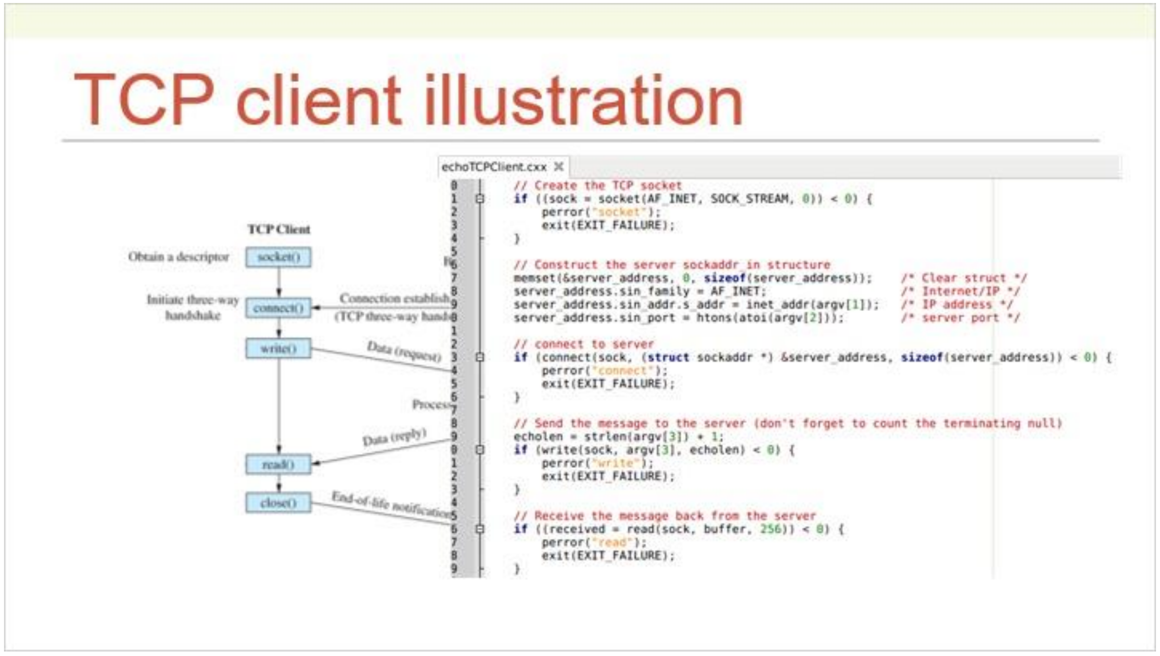


Client system call: connect

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

- connects socket to remote IP number and port
- `struct sockaddr` holds address information
  - will accept `struct sockaddr_in` pointer
- `addrlen` specifies length of `addr` structure
- returns 0 on success, -1 otherwise

1.12 TCP client illustration

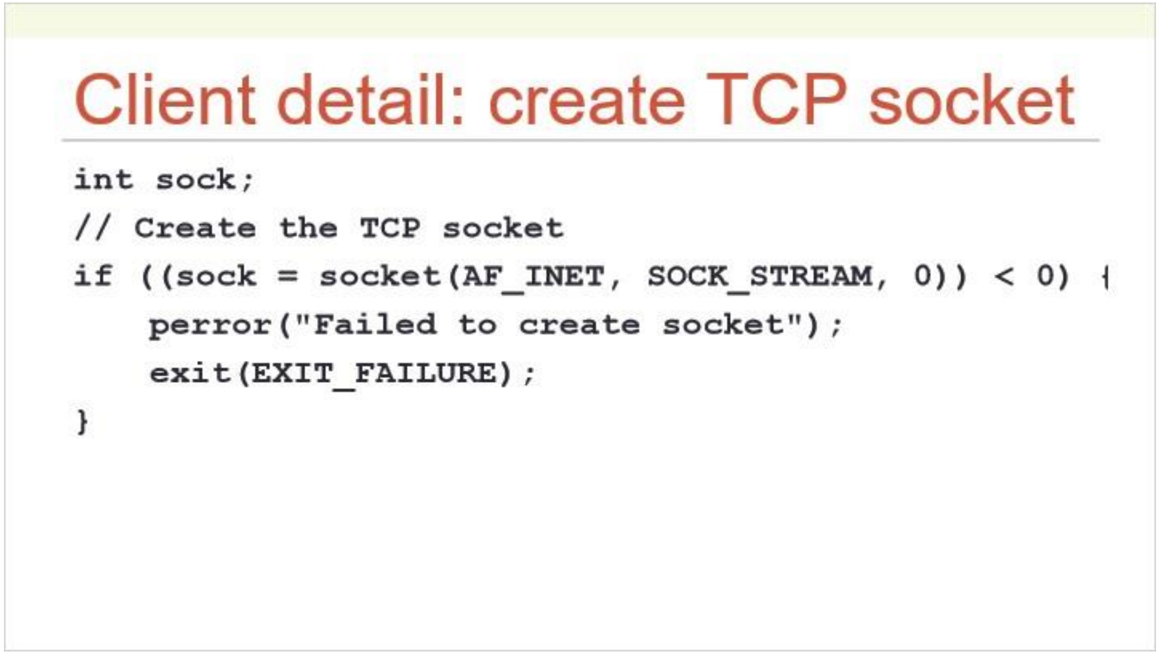


TCP client illustration

echoTCPClient.cxx

```
// Create the TCP socket
if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
    perror("socket");
    exit(EXIT_FAILURE);
}

// Construct the server sockaddr_in structure
memset(&server_address, 0, sizeof(server_address));     /* Clear struct */
server_address.sin_family = AF_INET;                    /* Internet/IP */
server_address.sin_addr.s_addr = inet_addr(argv[1]);   /* IP address */
server_address.sin_port = htons(atoi(argv[2]));         /* server port */

// connect to server
if (connect(sock, (struct sockaddr *) &server_address, sizeof(server_address)) < 0) {
    perror("connect");
    exit(EXIT_FAILURE);
}

// Send the message to the server (don't forget to count the terminating null)
echolen = strlen(argv[3]) + 1;
if (write(sock, argv[3], echolen) < 0) {
    perror("write");
    exit(EXIT_FAILURE);
}

// Receive the message back from the server
if ((received = read(sock, buffer, 256)) < 0) {
    perror("read");
    exit(EXIT_FAILURE);
}
```

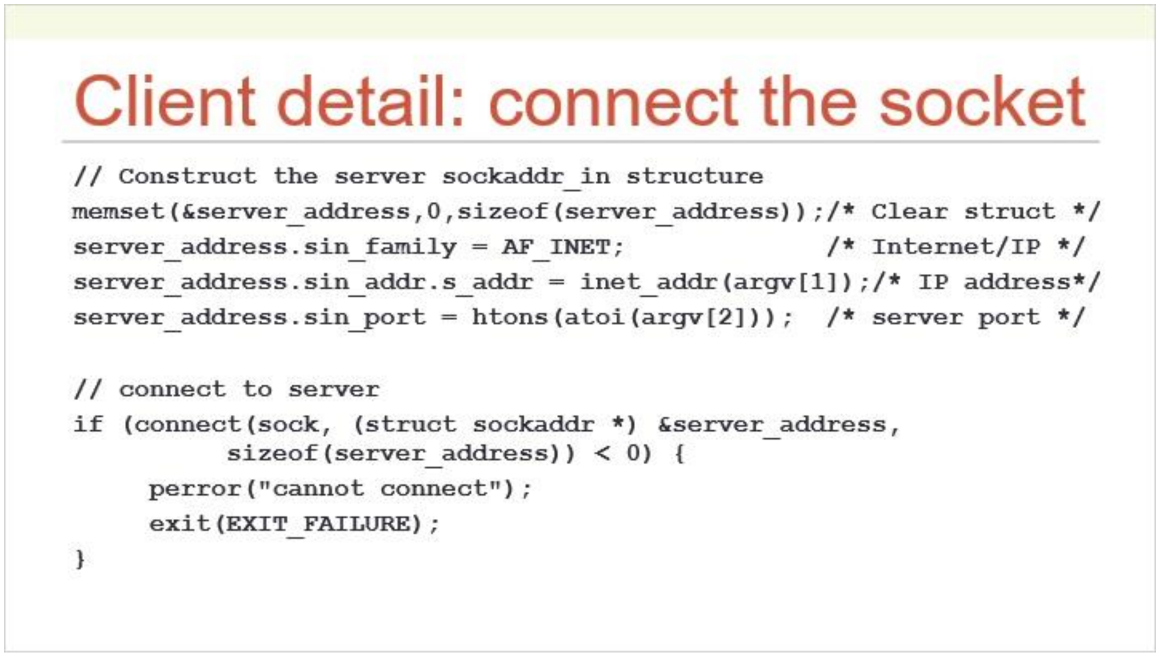
1.13 Client detail: create TCP socket



Client detail: create TCP socket

```
int sock;
// Create the TCP socket
if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
    perror("Failed to create socket");
    exit(EXIT_FAILURE);
}
```

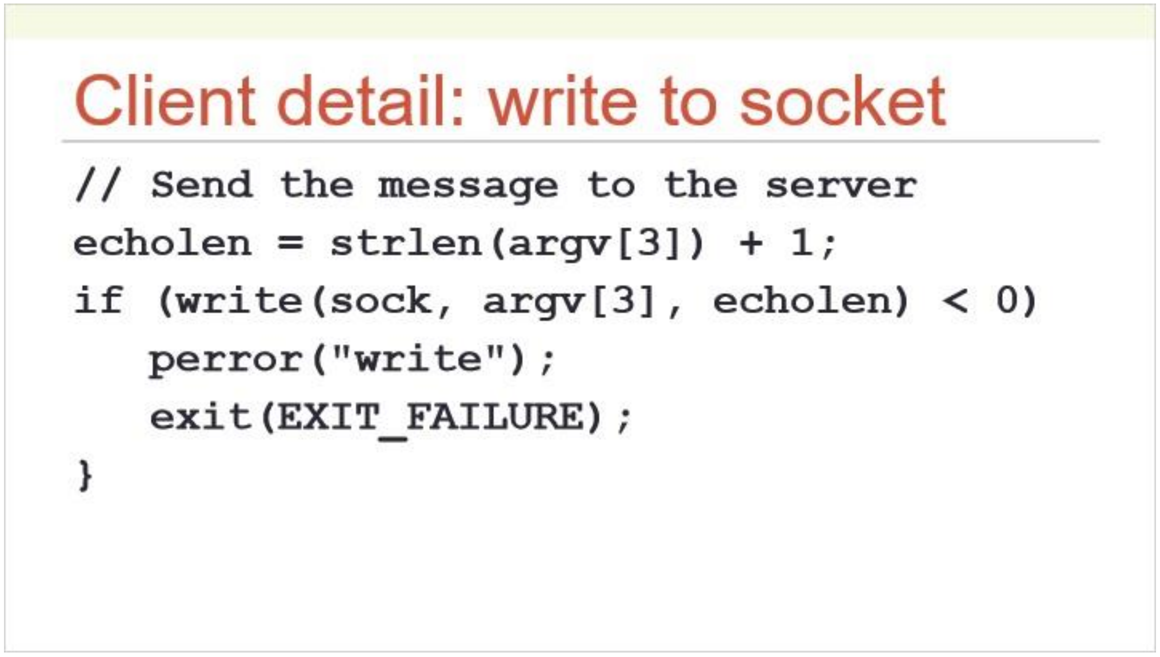
1.14 Client detail: connect the socket



Client detail: connect the socket

```
// Construct the server sockaddr_in structure
memset(&server_address,0,sizeof(server_address));/* Clear struct */
server_address.sin_family = AF_INET;            /* Internet/IP */
server_address.sin_addr.s_addr = inet_addr(argv[1]);/* IP address*/
server_address.sin_port = htons(atoi(argv[2]));  /* server port */

// connect to server
if (connect(sock, (struct sockaddr *) &server_address,
          sizeof(server_address)) < 0) {
     perror("cannot connect");
     exit(EXIT_FAILURE);
}
```

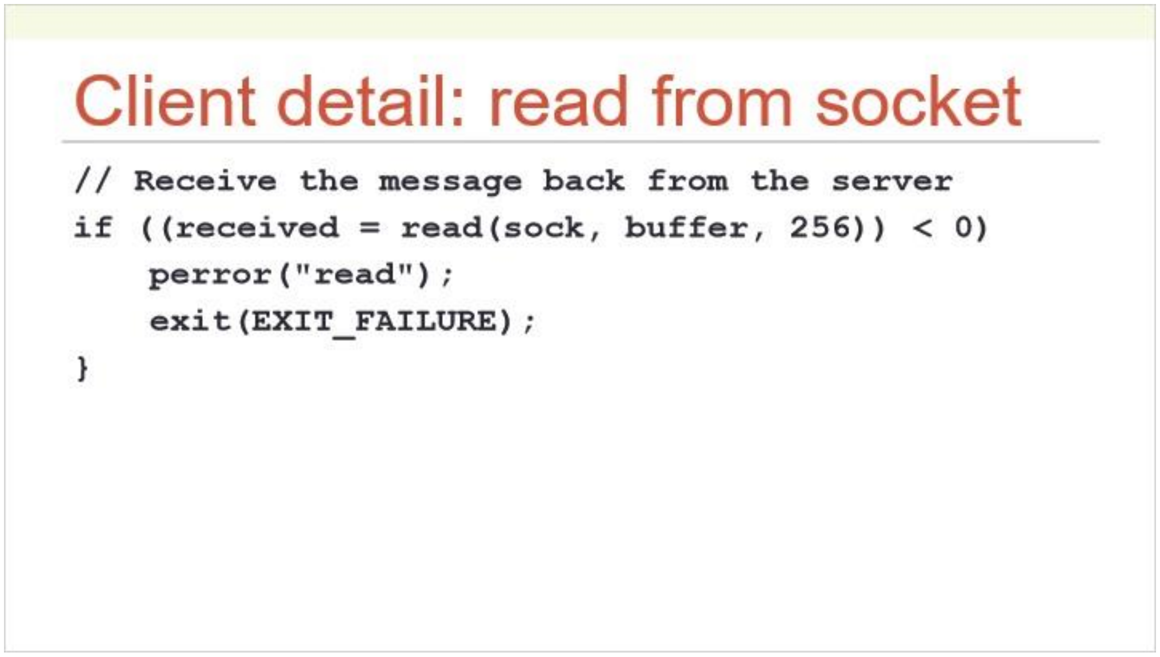
1.15 Client detail: write to socket



Client detail: write to socket

```
// Send the message to the server
echolen = strlen(argv[3]) + 1;
if (write(sock, argv[3], echolen) < 0)
   perror("write");
   exit(EXIT_FAILURE);
}
```

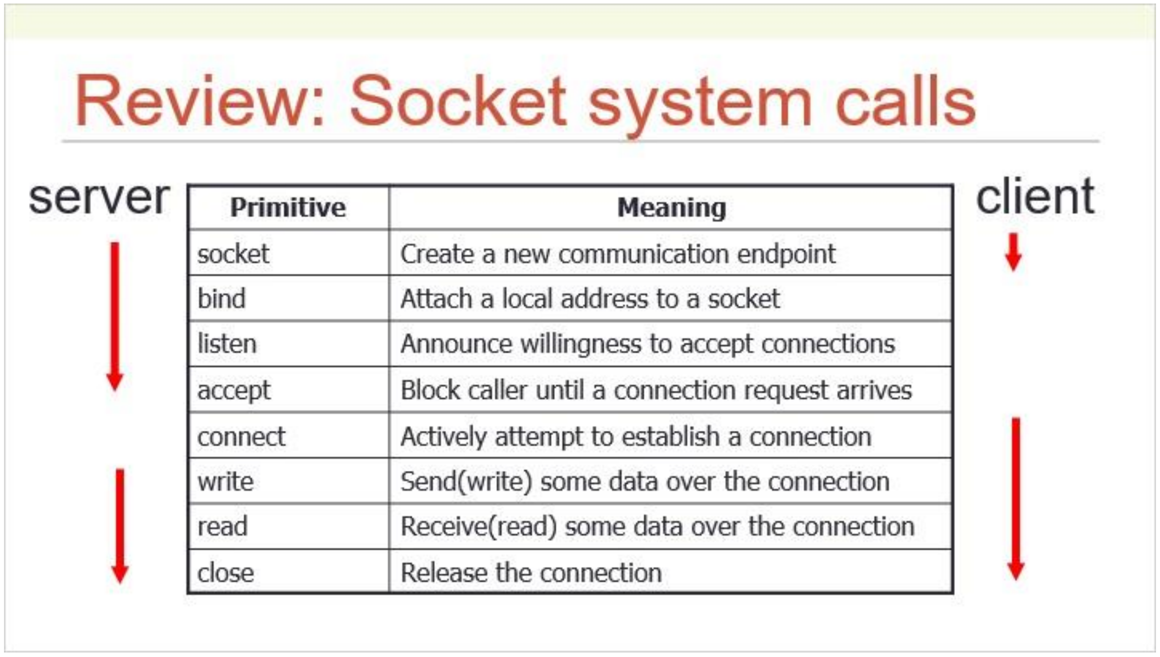
1.16 Client detail: read from socket



Client detail: read from socket

```
// Receive the message back from the server
if ((received = read(sock, buffer, 256)) < 0)
    perror("read");
    exit(EXIT_FAILURE);
}
```

1.17 Socket system calls

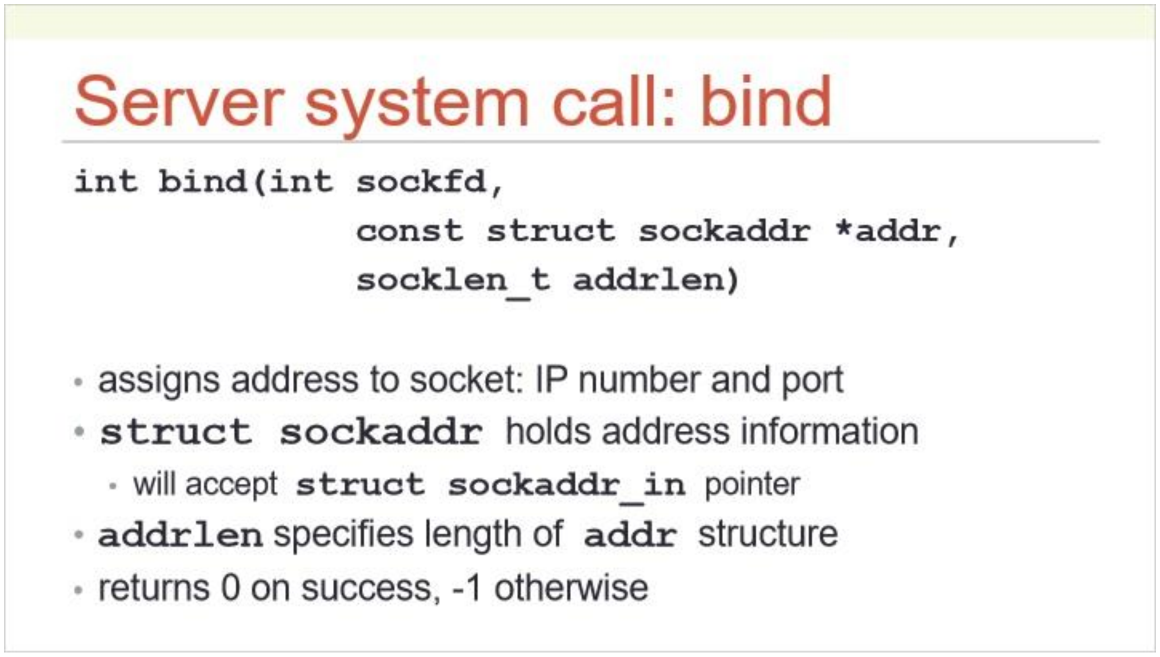


Review: Socket system calls

server
client

Primitive	Meaning
socket	Create a new communication endpoint
bind	Attach a local address to a socket
listen	Announce willingness to accept connections
accept	Block caller until a connection request arrives
connect	Actively attempt to establish a connection
write	Send(write) some data over the connection
read	Receive(read) some data over the connection
close	Release the connection

1.18 Server system call: bind

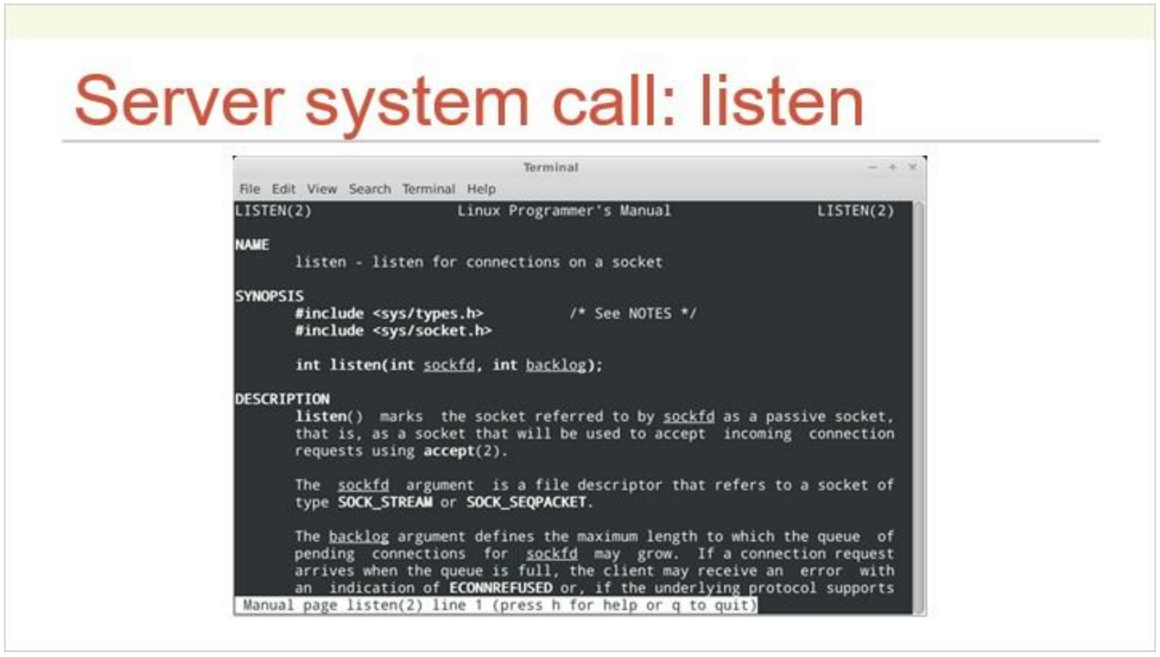


Server system call: bind

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

- assigns address to socket: IP number and port
- `struct sockaddr` holds address information
  - will accept `struct sockaddr_in` pointer
- `addrlen` specifies length of `addr` structure
- returns 0 on success, -1 otherwise

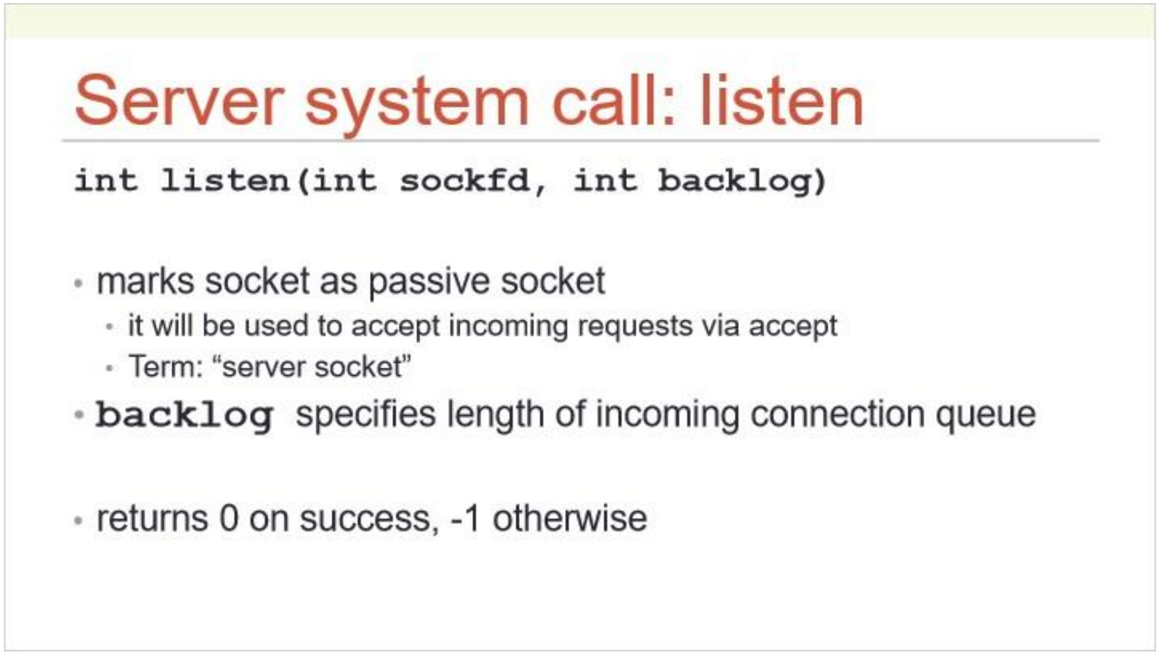
1.19 Server system call: listen



Server system call: listen

```
LISTEN(2)                  Linux Programmer's Manual                 LISTEN(2)

NAME
       listen - listen for connections on a socket

SYNOPSIS
       #include <sys/types.h>          /* See NOTES */
       #include <sys/socket.h>

       int listen(int sockfd, int backlog);

DESCRIPTION
       listen()  marks  the socket referred to by sockfd as a passive socket,
       that is, as a socket that will be used to accept  incoming  connection
       requests using accept(2).

       The  sockfd  argument  is a file descriptor that refers to a socket of
       type SOCK_STREAM or SOCK_SEQPACKET.

       The backlog argument defines the maximum length to which the queue  of
       pending  connections  for  sockfd  may  grow.  If a connection request
       arrives when the queue is full, the client may receive an  error  with
       an  indication of ECONNREFUSED or, if the underlying protocol supports
 Manual page listen(2) line 1 (press h for help or q to quit)
```

1.20 Server system call: listen

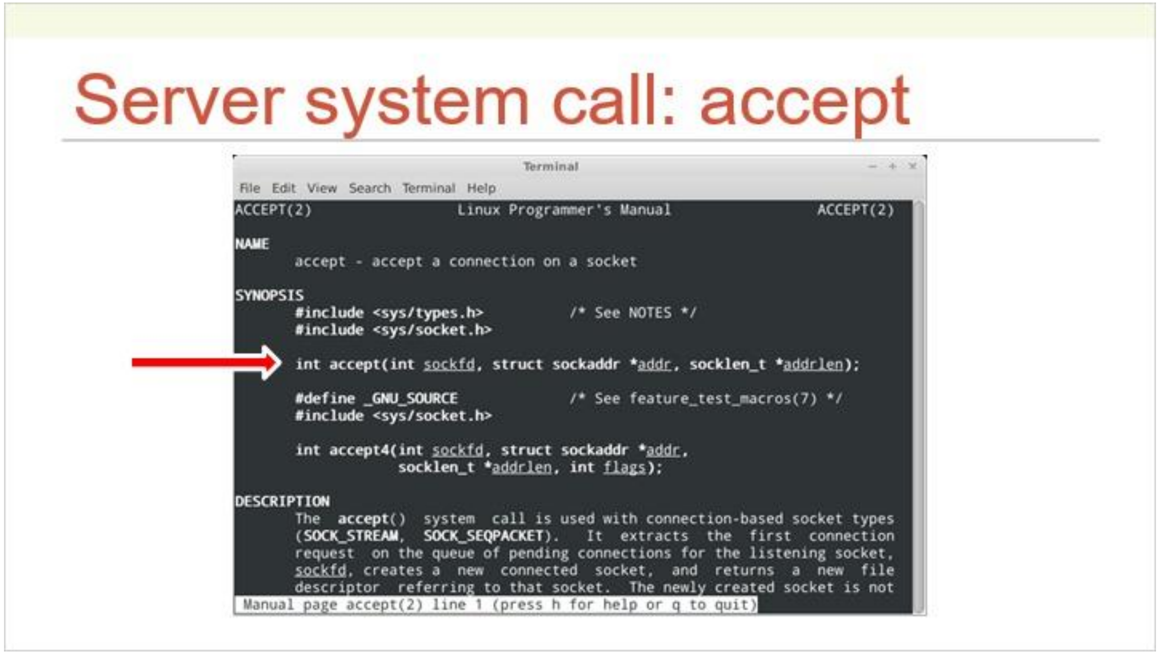


Server system call: listen

`int listen(int sockfd, int backlog)`

- marks socket as passive socket
  - it will be used to accept incoming requests via accept
  - Term: “server socket”
- **`backlog`** specifies length of incoming connection queue
- returns 0 on success, -1 otherwise

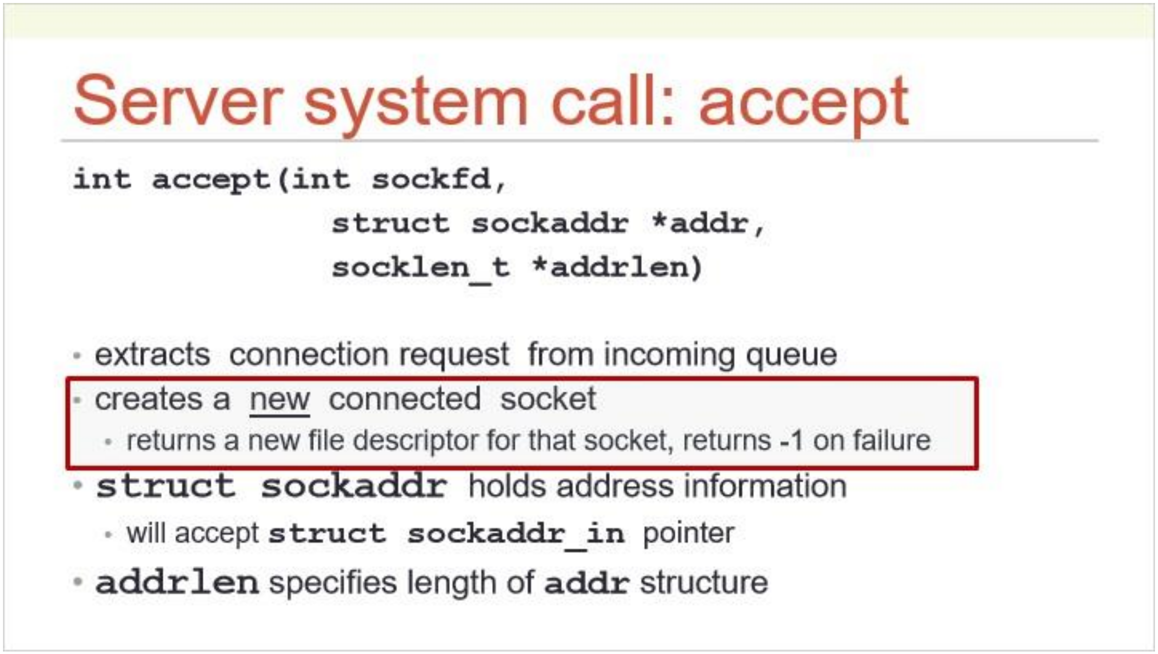
1.21 Server system call: accept



Server system call: accept

```
ACCEPT(2)                 Linux Programmer's Manual                ACCEPT(2)

NAME
       accept - accept a connection on a socket

SYNOPSIS
       #include <sys/types.h>          /* See NOTES */
       #include <sys/socket.h>

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

       #define _GNU_SOURCE             /* See feature_test_macros(7) */
       #include <sys/socket.h>

       int accept4(int sockfd, struct sockaddr *addr,
                   socklen_t *addrlen, int flags);

DESCRIPTION
       The  accept()  system  call is used with connection-based socket types
       (SOCK_STREAM,  SOCK_SEQPACKET).   It  extracts  the  first  connection
       request  on the queue of pending connections for the listening socket,
       sockfd, creates a  new  connected  socket,  and  returns  a  new  file
       descriptor  referring to that socket.  The newly created socket is not
 Manual page accept(2) line 1 (press h for help or q to quit)
```

1.22 Server system call: accept

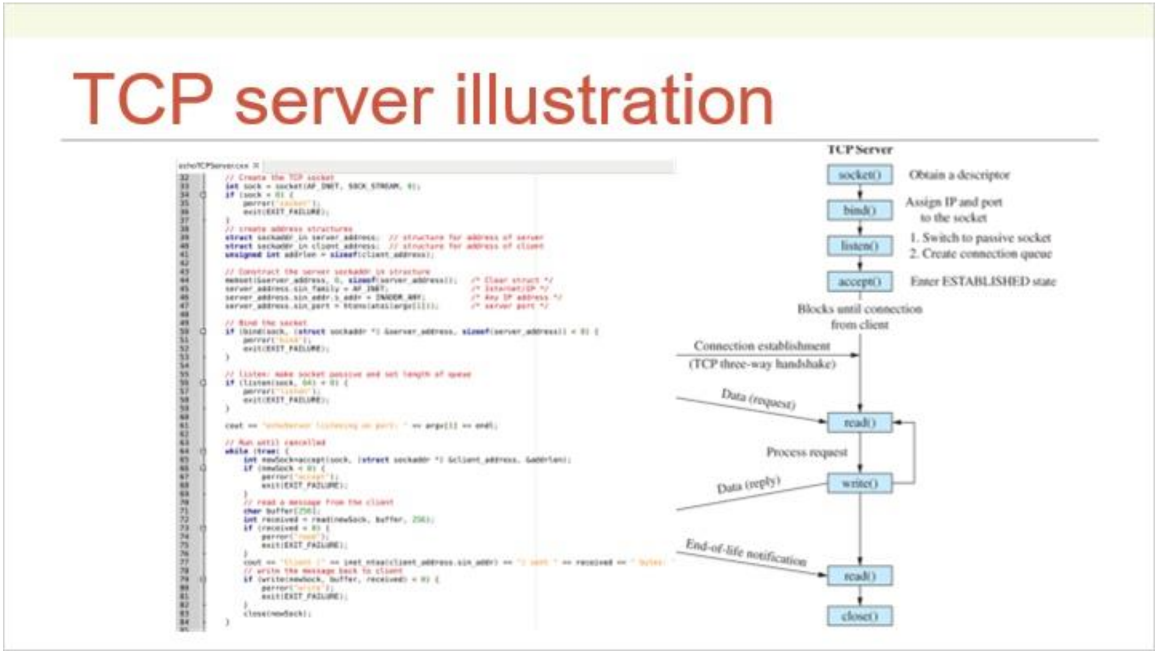


Server system call: accept

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

- extracts connection request from incoming queue
- creates a new connected socket
  - returns a new file descriptor for that socket, returns -1 on failure
- **`struct sockaddr`** holds address information
  - will accept **`struct sockaddr_in`** pointer
- **`addrlen`** specifies length of **`addr`** structure

1.23 TCP server illustration

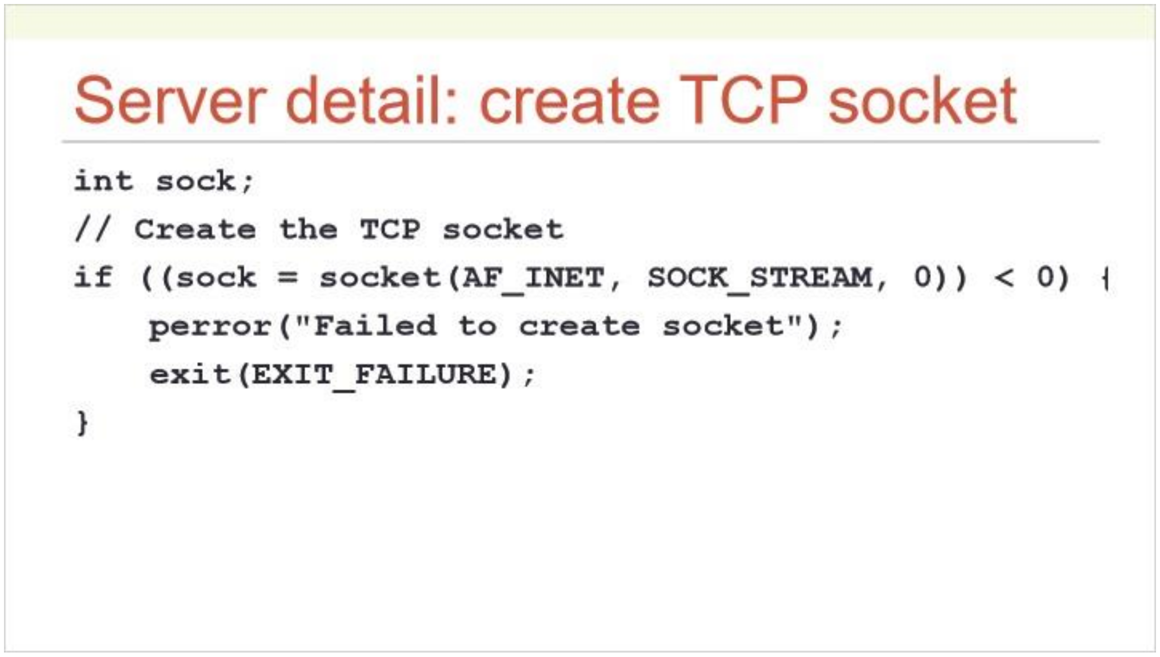


TCP server illustration

echoTCPServer.cxx

```
    // Create the TCP socket
    int sock = socket(AF_INET, SOCK_STREAM, 0);
    if (sock < 0) {
        perror("socket");
        exit(EXIT_FAILURE);
    }
    // create address structures
    struct sockaddr_in server_address;  // structure for address of server
    struct sockaddr_in client_address;  // structure for address of client
    unsigned int addrlen = sizeof(client_address);

    // Construct the server sockaddr_in structure
    memset(&server_address, 0, sizeof(server_address));   /* Clear struct */
    server_address.sin_family = AF_INET;                   /* Internet/IP */
    server_address.sin_addr.s_addr = INADDR_ANY;           /* Any IP address */
    server_address.sin_port = htons(atoi(argv[1]));        /* server port */

    // Bind the socket
    if (bind(sock, (struct sockaddr *) &server_address, sizeof(server_address)) < 0) {
        perror("bind");
        exit(EXIT_FAILURE);
    }

    // listen: make socket passive and set length of queue
    if (listen(sock, 64) < 0) {
        perror("listen");
        exit(EXIT_FAILURE);
    }

    cout << "echoServer listening on port: " << argv[1] << endl;

    // Run until cancelled
    while (true) {
        int newSock=accept(sock, (struct sockaddr *) &client_address, &addrlen);
        if (newSock < 0) {
            perror("accept");
            exit(EXIT_FAILURE);
        }
        // read a message from the client
        char buffer[256];
        int received = read(newSock, buffer, 256);
        if (received < 0) {
            perror("read");
            exit(EXIT_FAILURE);
        }
        cout << "Client [" << inet_ntoa(client_address.sin_addr) << "] sent " << received << " bytes: "
        // write the message back to client
        if (write(newSock, buffer, received) < 0) {
            perror("write");
            exit(EXIT_FAILURE);
        }
        close(newSock);
    }
```

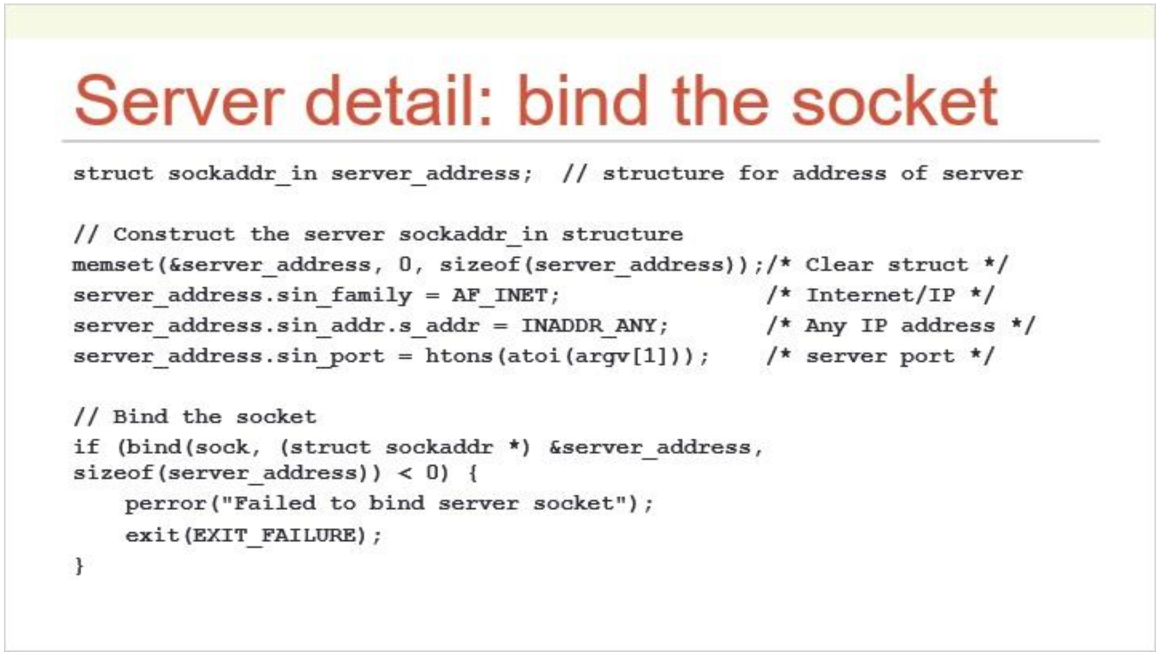
1.24 Server detail: create TCP socket



Server detail: create TCP socket

```
int sock;
// Create the TCP socket
if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
    perror("Failed to create socket");
    exit(EXIT_FAILURE);
}
```

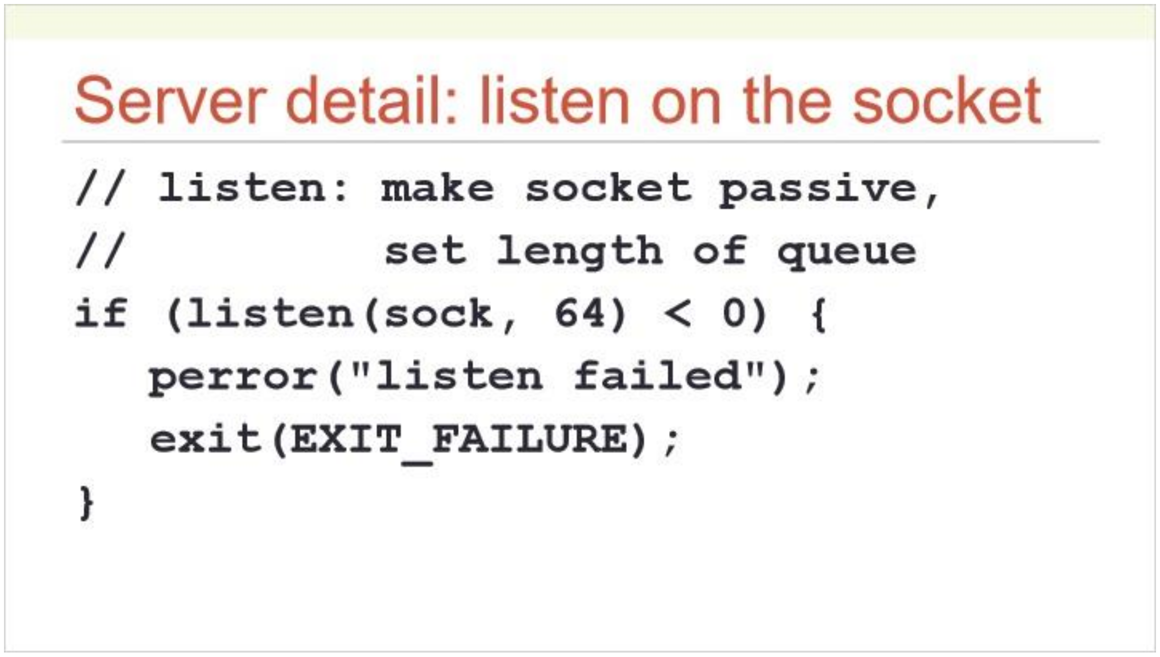
1.25 Server detail: bind the socket



Server detail: bind the socket

```
struct sockaddr_in server_address;  // structure for address of server

// Construct the server sockaddr_in structure
memset(&server_address, 0, sizeof(server_address));/* Clear struct */
server_address.sin_family = AF_INET;                /* Internet/IP */
server_address.sin_addr.s_addr = INADDR_ANY;        /* Any IP address */
server_address.sin_port = htons(atoi(argv[1]));     /* server port */

// Bind the socket
if (bind(sock, (struct sockaddr *) &server_address,
sizeof(server_address)) < 0) {
    perror("Failed to bind server socket");
    exit(EXIT_FAILURE);
}
```

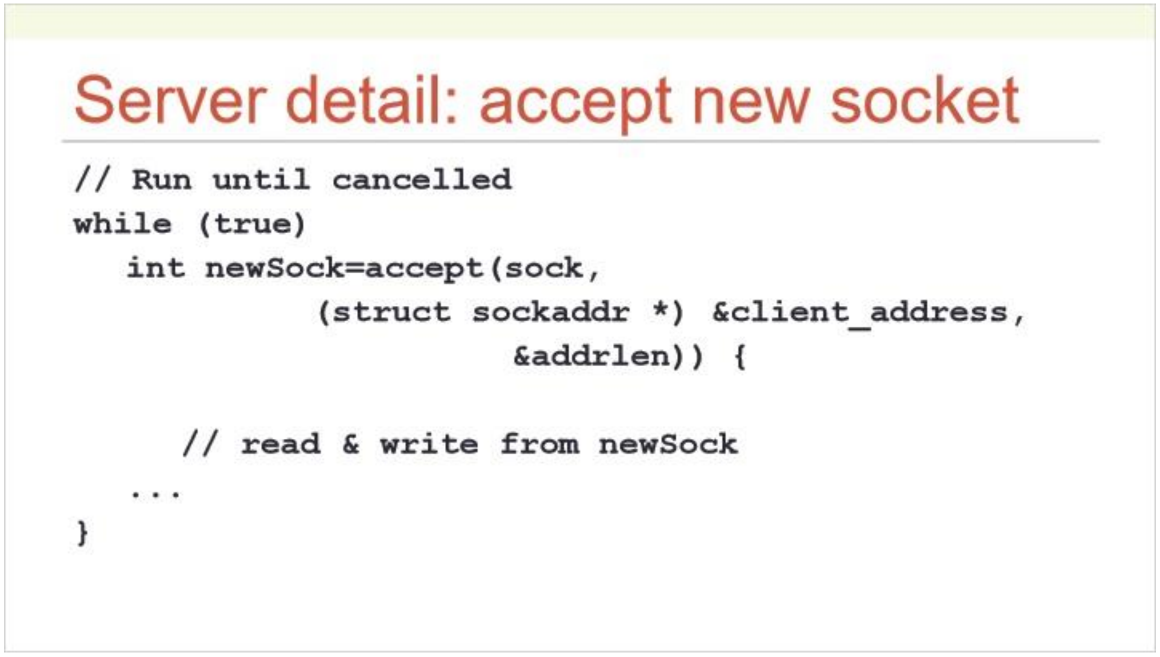
1.26 Server detail: listen on the socket



Server detail: listen on the socket

```
// listen: make socket passive,
//         set length of queue
if (listen(sock, 64) < 0) {
   perror("listen failed");
   exit(EXIT_FAILURE);
}
```

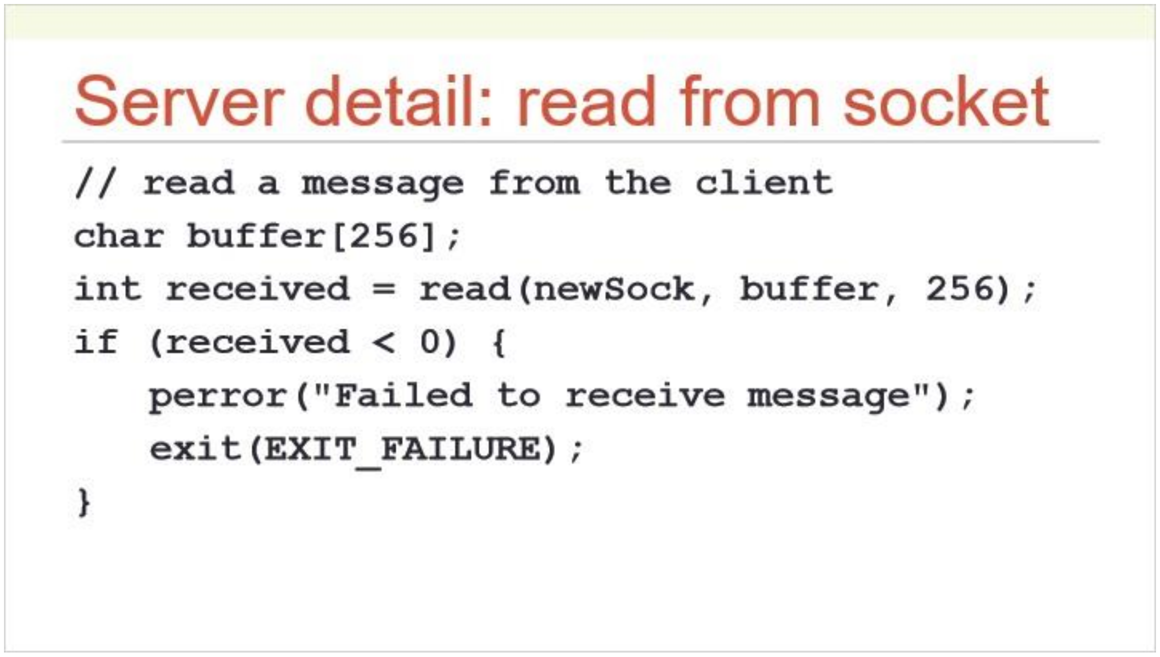
1.27 Server detail: accept new socket



Server detail: accept new socket

```
// Run until cancelled
while (true)
   int newSock=accept(sock,
            (struct sockaddr *) &client_address,
                     &addrlen)) {

      // read & write from newSock
   ...
}
```

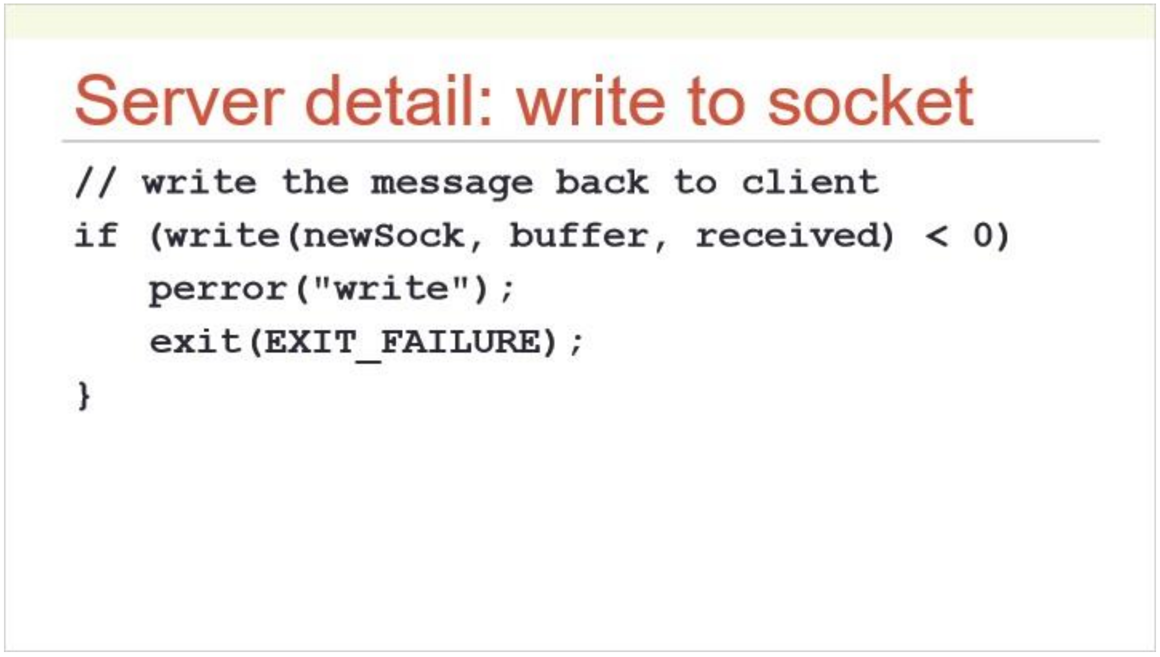
1.28 Server detail: read from socket



Server detail: read from socket

```
// read a message from the client
char buffer[256];
int received = read(newSock, buffer, 256);
if (received < 0) {
    perror("Failed to receive message");
    exit(EXIT_FAILURE);
}
```

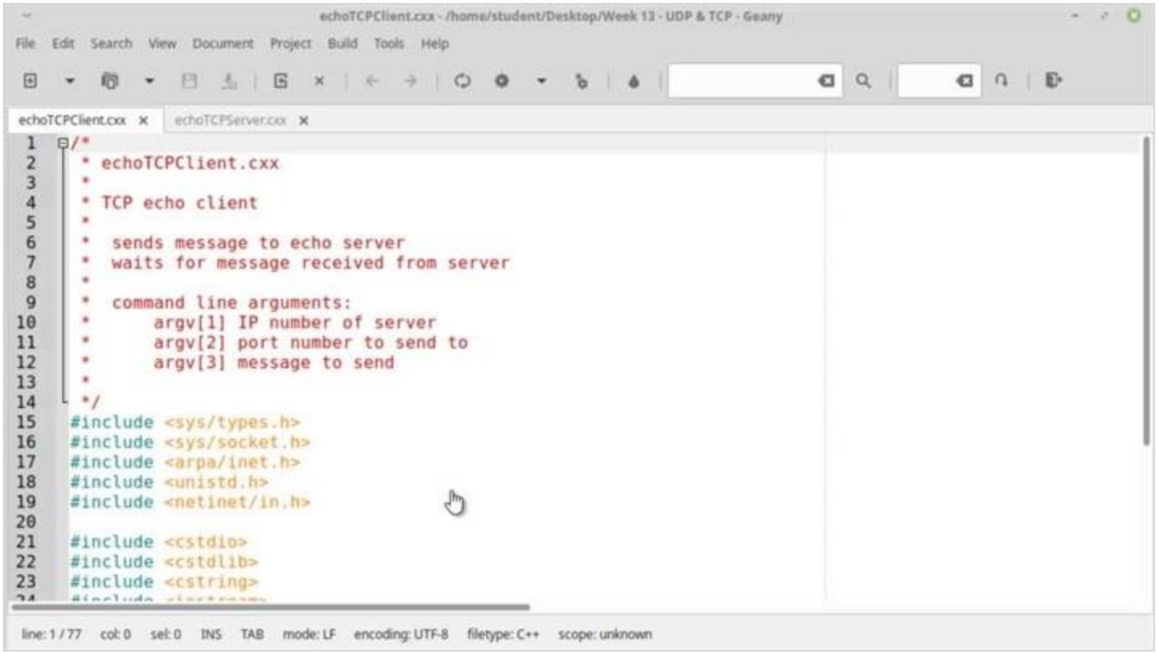
1.29 Server detail: write to socket



Server detail: write to socket

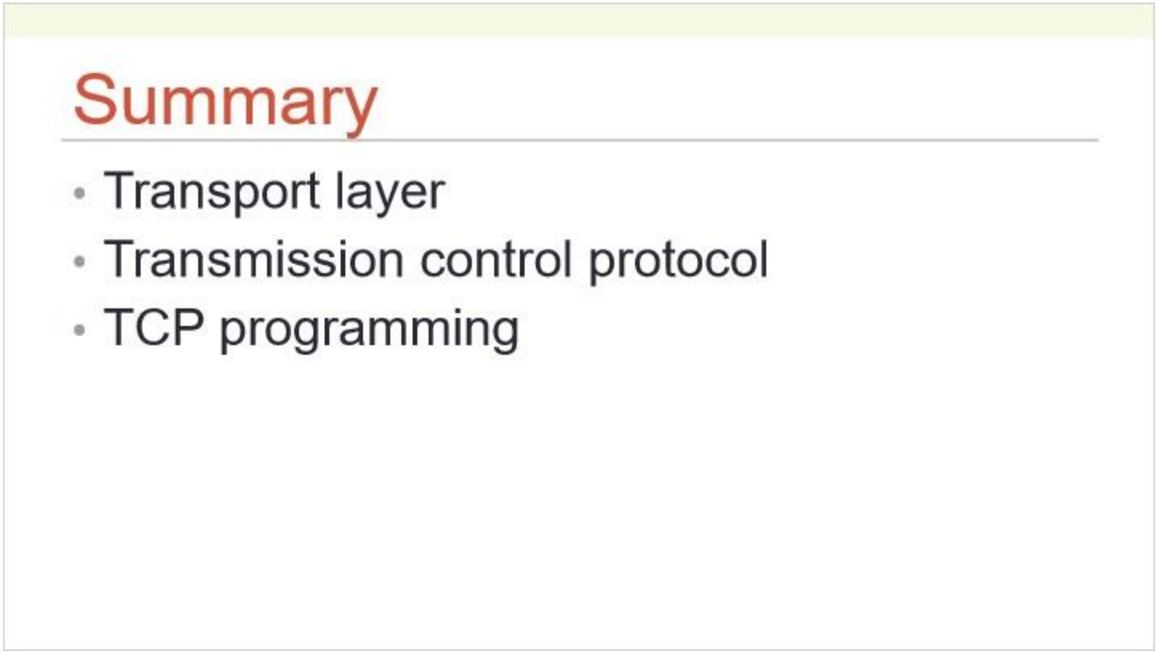
```
// write the message back to client
if (write(newSock, buffer, received) < 0)
    perror("write");
    exit(EXIT_FAILURE);
}
```

1.30 Echo Example


echoTCPClient.cxx - /home/student/Desktop/Week 13 - UDP & TCP - Geany
File Edit Search View Document Project Build Tools Help
echoTCPClient.cxx
echoTCPServer.cxx
line: 1 / 77 col: 0 sel: 0 INS TAB mode: LF encoding: UTF-8 filetype: C++ scope: unknown

```
/*
 * echoTCPClient.cxx
 *
 * TCP echo client
 *
 *  sends message to echo server
 *  waits for message received from server
 *
 *  command line arguments:
 *      argv[1] IP number of server
 *      argv[2] port number to send to
 *      argv[3] message to send
 *
 */
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <netinet/in.h>

#include <cstdio>
#include <cstdlib>
#include <cstring>
```

1.31 Summary



Summary

- Transport layer
- Transmission control protocol
- TCP programming