CSCI 330
The UNIX System

Transmission Control Protocol

Unit Overview

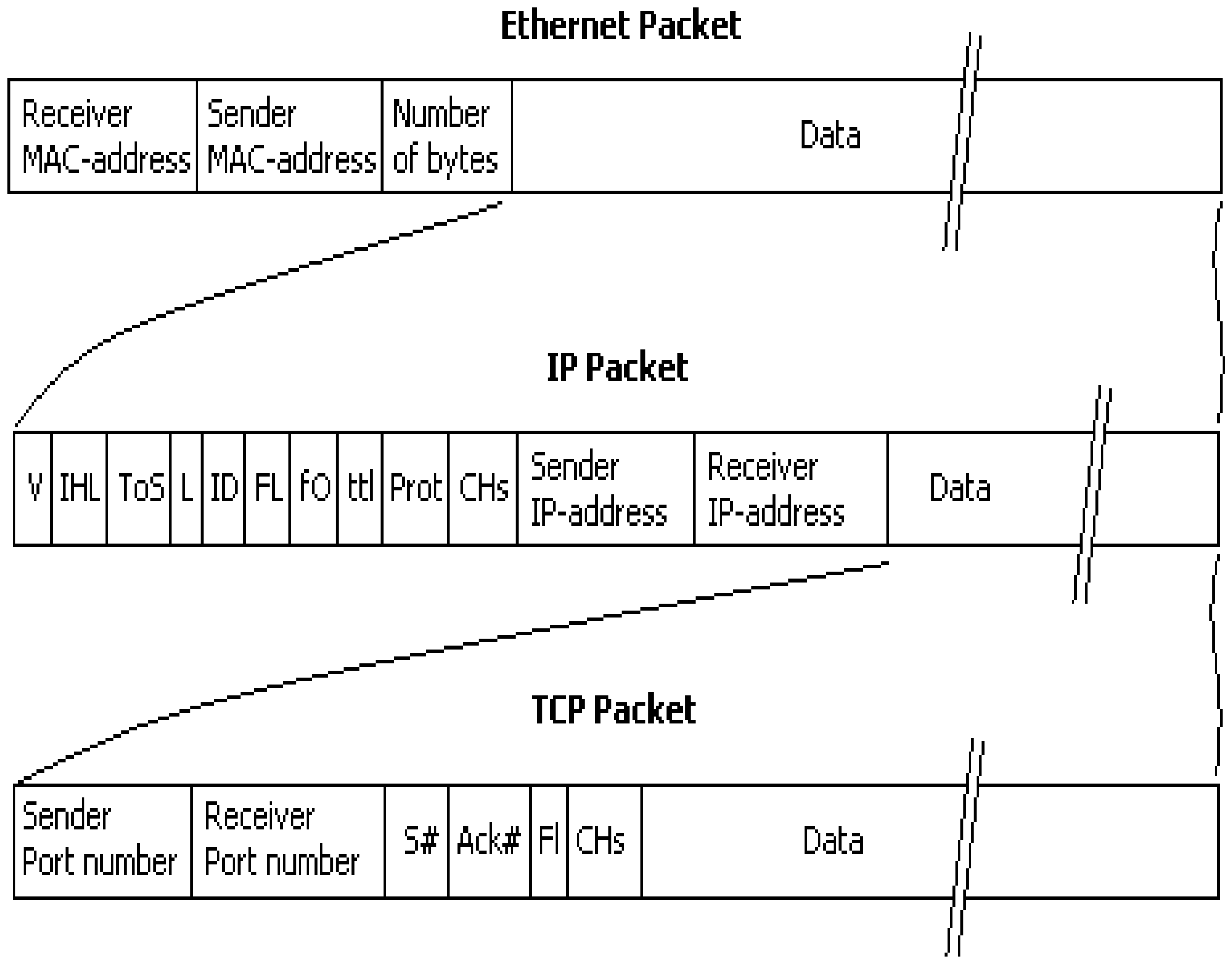
- Transport layer
- Transmission control protocol
- TCP programming

Transport Layer

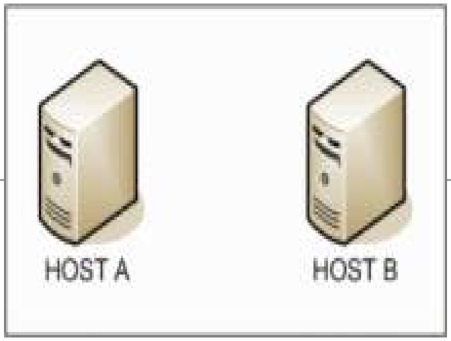
- provides end-to-end communication services for applications
- provides multiple endpoints on a single node: <u>port</u>

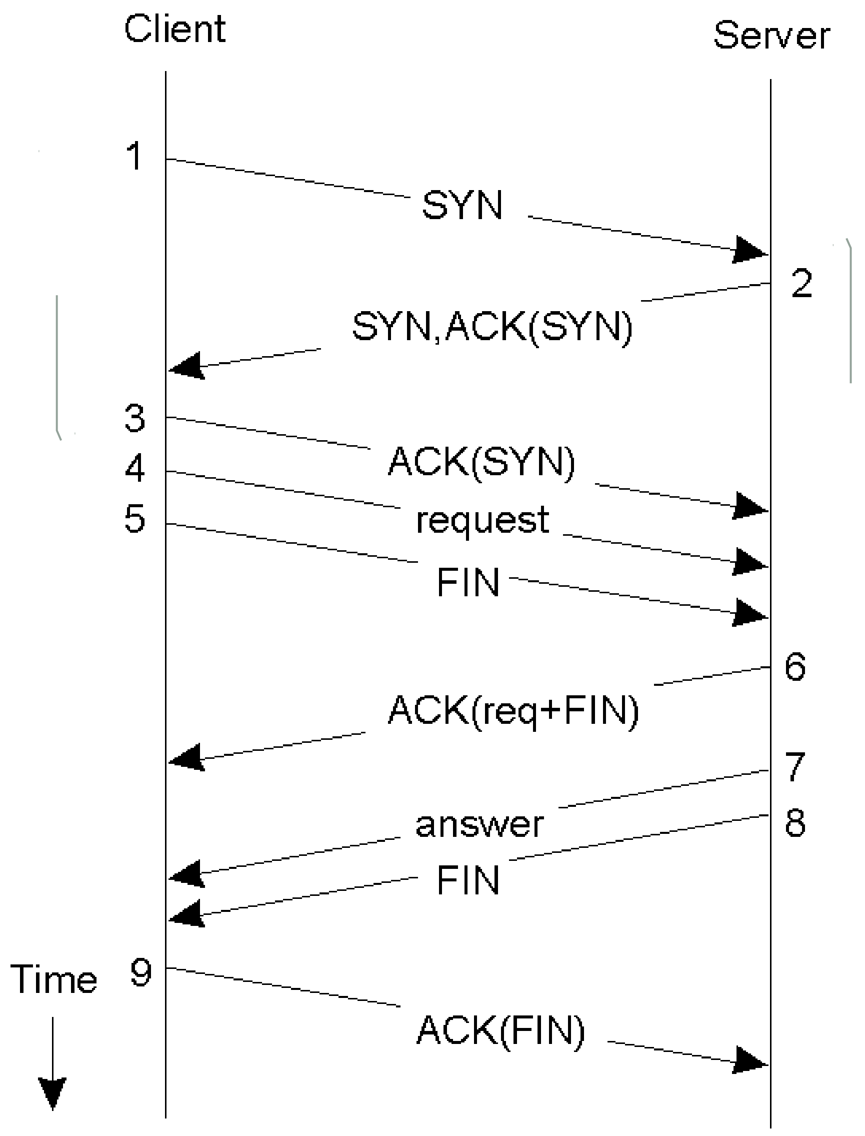
- TCP: transmission control protocol
  - connection oriented, guaranteed delivery
  - stream oriented: basis for: http, ftp, smtp, ssh
- UDP: user datagram protocol
  - best effort
  - datagram oriented: basis for: dns, rtp

TCP/IP protocol packet



TCP communication





TCP 3-way handshake

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

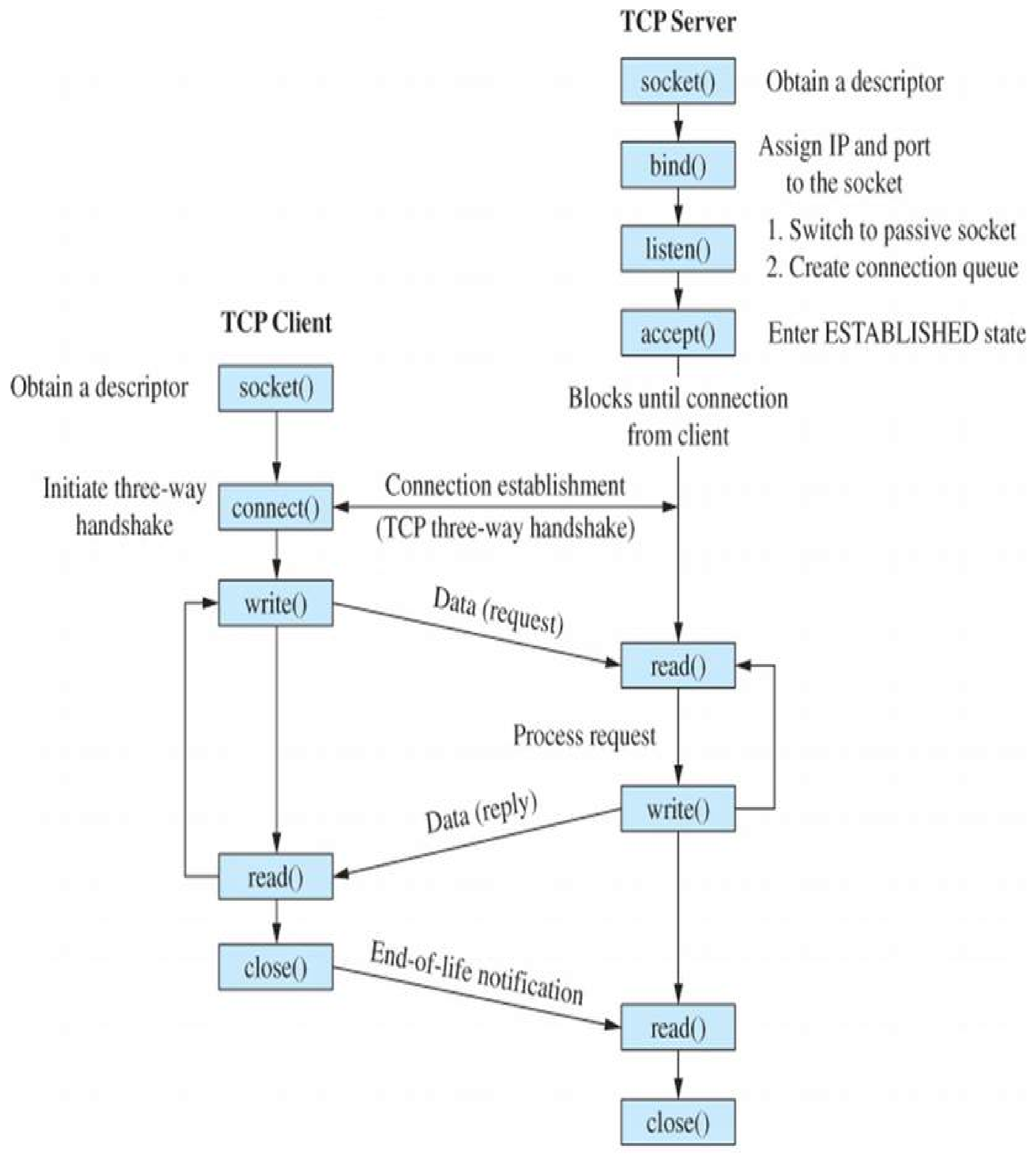
Socket system calls

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

TCP communications pattern

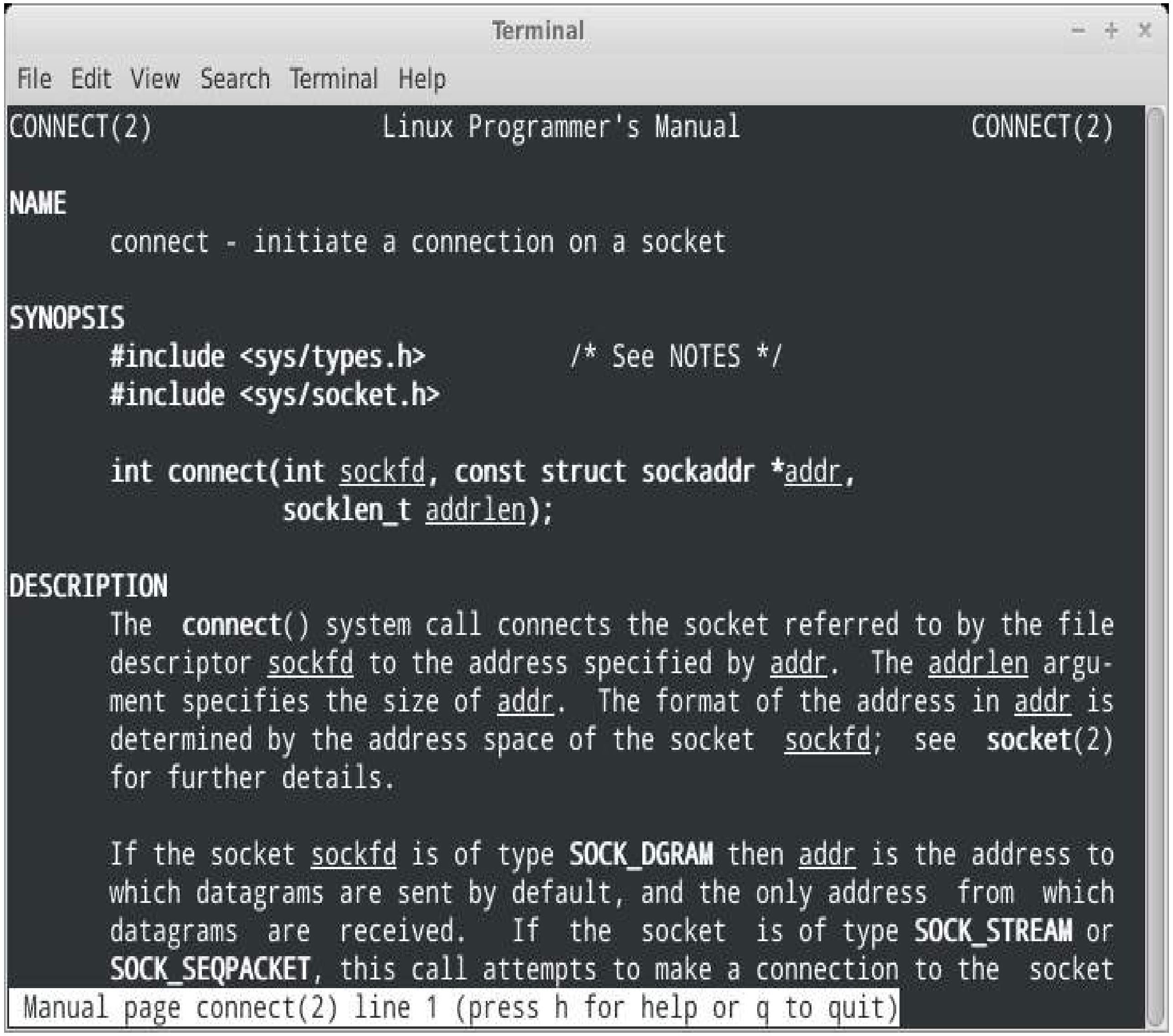


System call: socket

```
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 datagrams
- **`protocol`** is set to 0, i.e. default TCP
- returns socket descriptor:
  - used in bind, listen, accept, connect, write, read, close

Client system call: connect



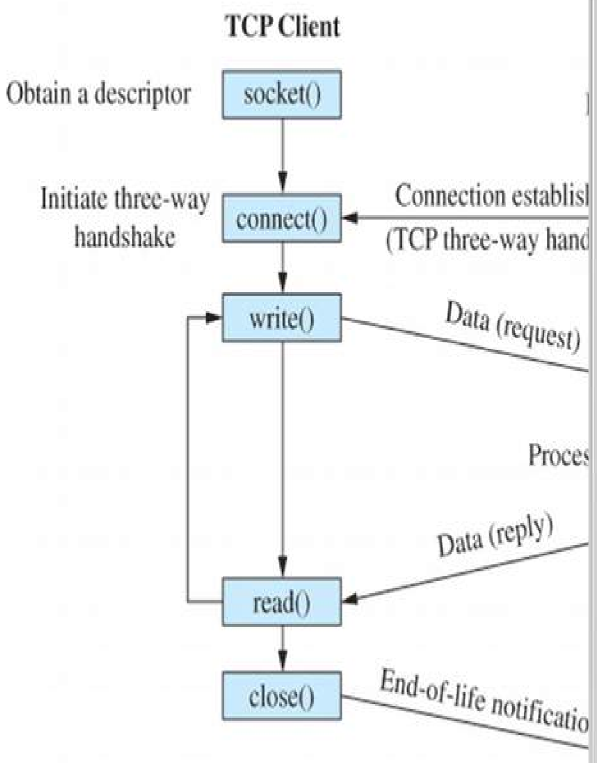
```
CONNECT(2)                 Linux Programmer's Manual                CONNECT(2)

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)
```

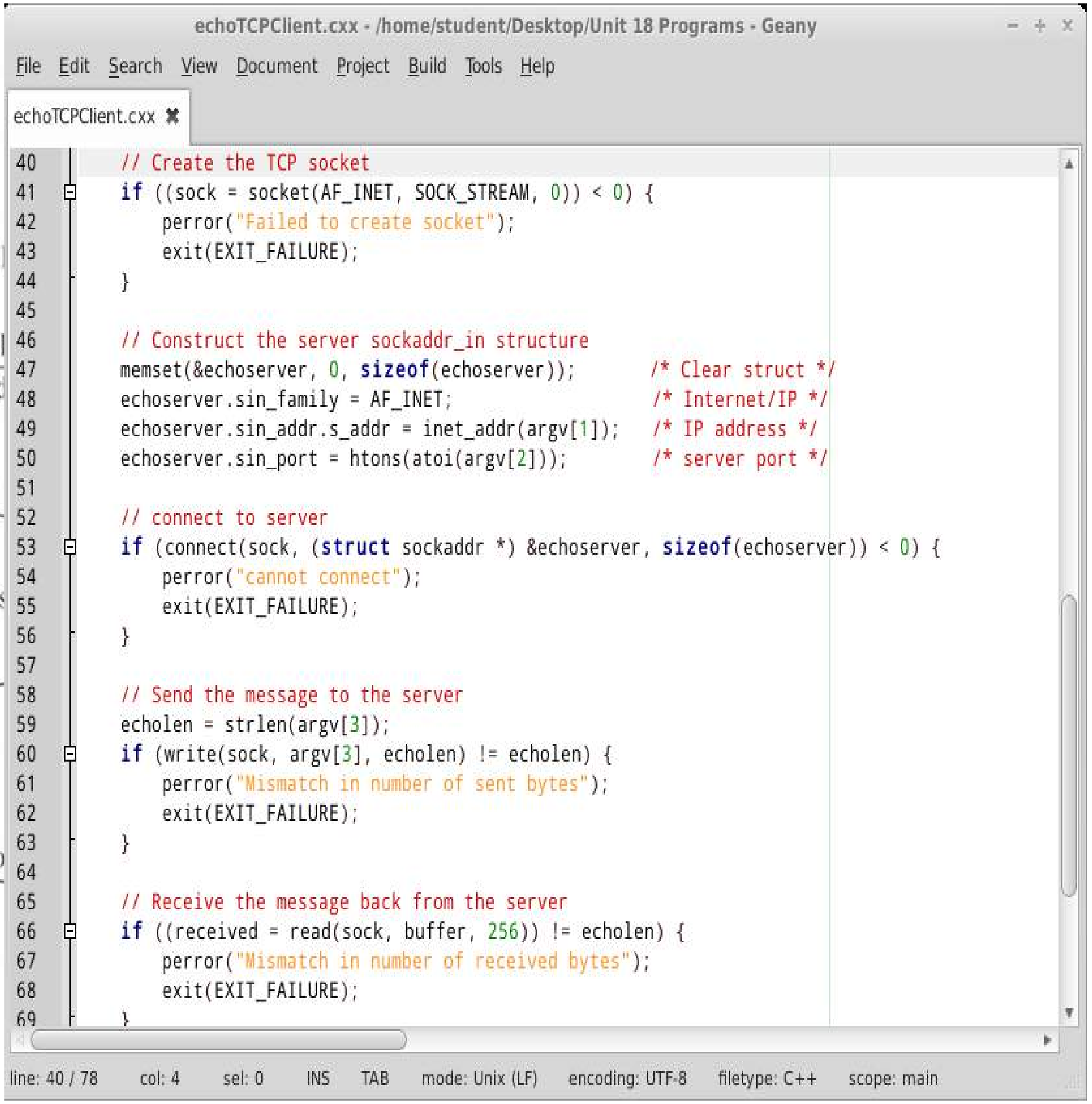
Client system call: connect

```
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

TCP client illustration




echoTCPClient.cxx - /home/student/Desktop/Unit 18 Programs - Geany
File Edit Search View Document Project Build Tools Help
echoTCPClient.cxx

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

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

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

    // Send the message to the server
    echolen = strlen(argv[3]);
    if (write(sock, argv[3], echolen) != echolen) {
        perror("Mismatch in number of sent bytes");
        exit(EXIT_FAILURE);
    }

    // Receive the message back from the server
    if ((received = read(sock, buffer, 256)) != echolen) {
        perror("Mismatch in number of received bytes");
        exit(EXIT_FAILURE);
    }
```

line: 40 / 78   col: 4   sel: 0   INS   TAB   mode: Unix (LF)   encoding: UTF-8   filetype: C++   scope: main

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);
}
```

Client detail: connect the socket

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


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

Client detail: write to socket

```
// Send the message to the server
echolen = strlen(argv[3]);
if (write(sock, argv[3], echolen) != echolen) {
   perror("Mismatch in number of sent bytes");
   exit(EXIT_FAILURE);
}
```

Client detail: read from socket

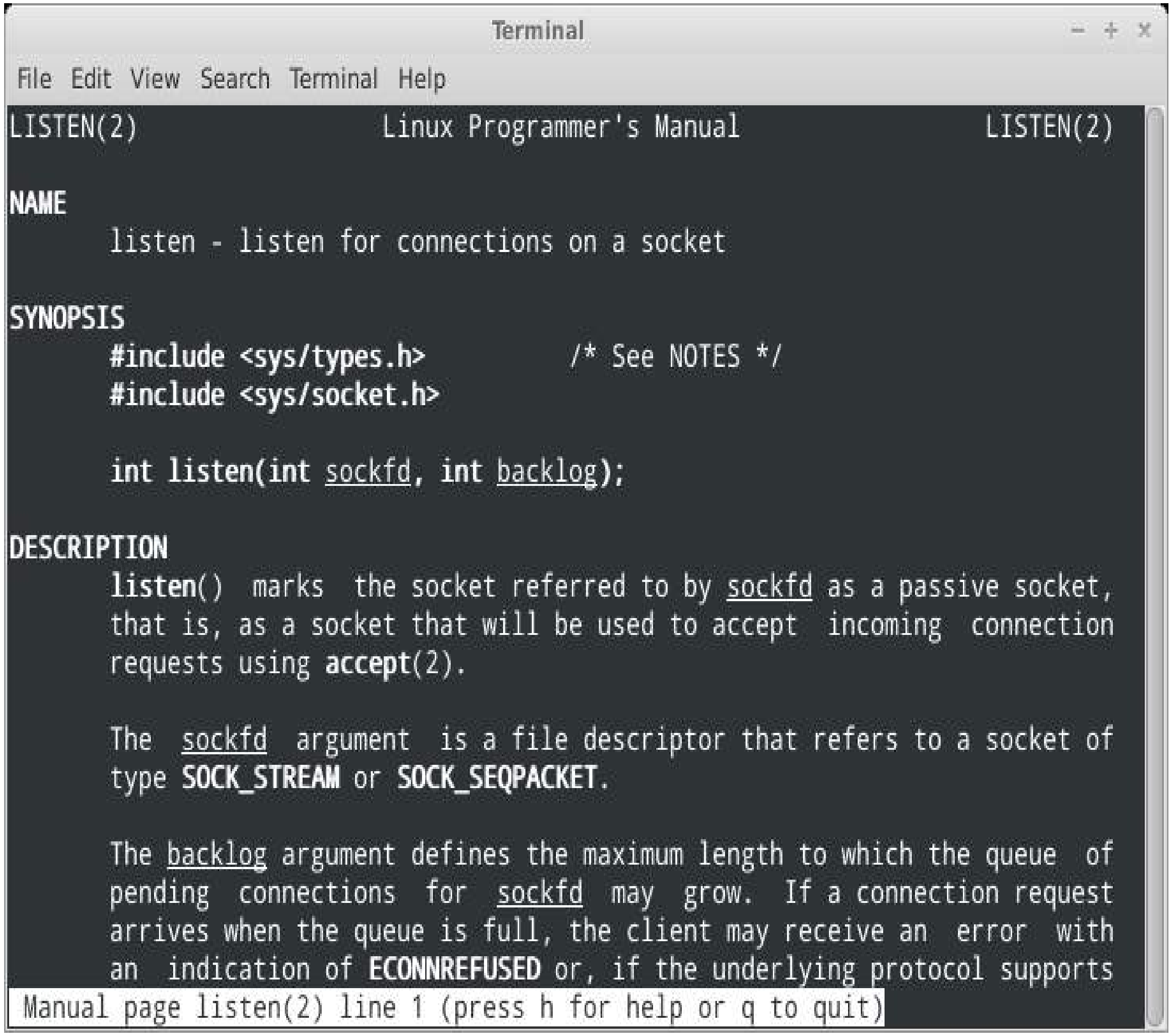
```
// Receive the message back from the server
if ((received = read(sock, buffer, 256)) != echolen) {
   perror("Mismatch in number of received bytes");
   exit(EXIT_FAILURE);
}
/* Assure null-terminated string */
buffer[received] = '\0';
cout << "Server ("
     << inet_ntoa(echoserver.sin_addr)
     << ") echoed: " << buffer << endl;
```

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

Server system call: listen



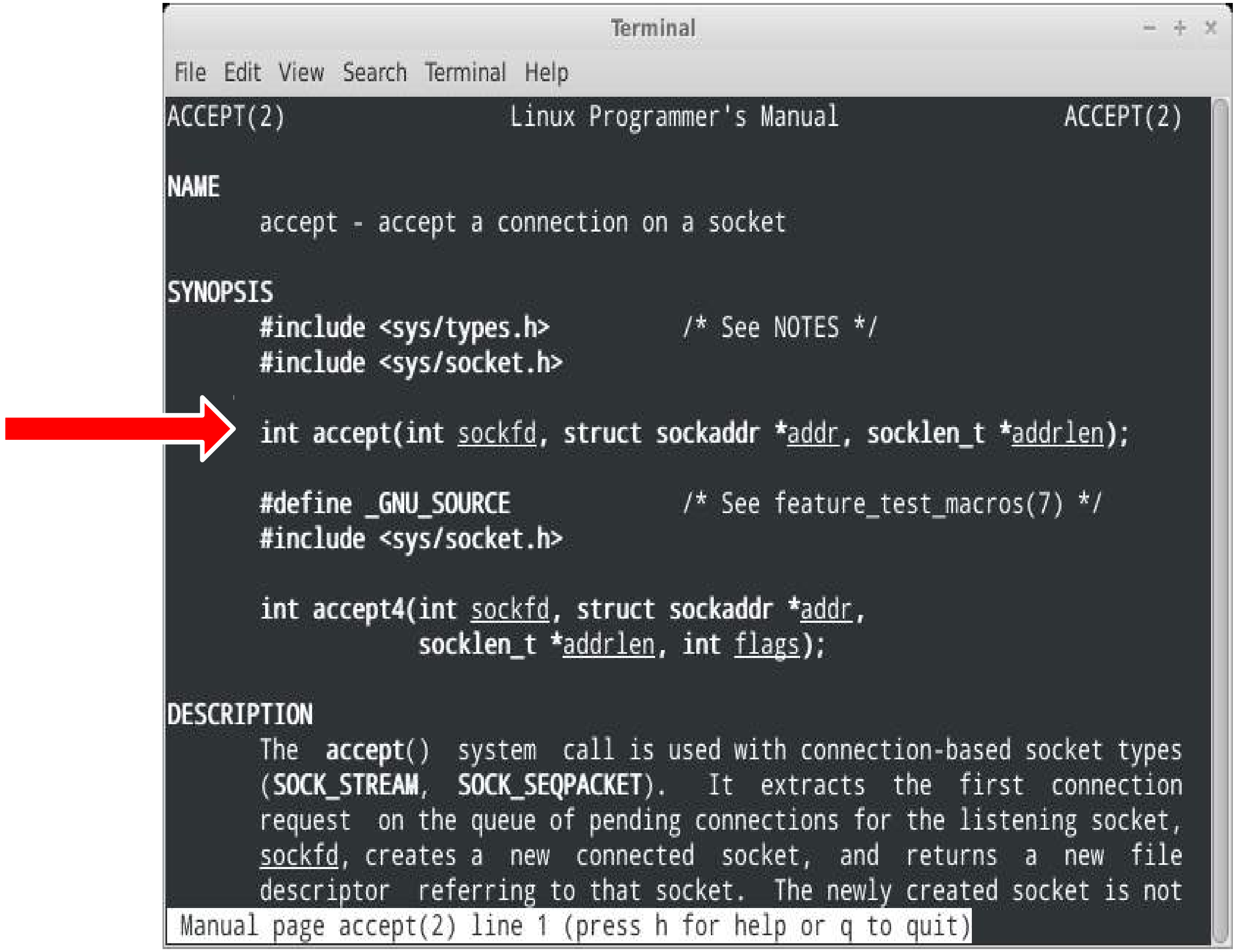
```
LISTEN(2)                  Linux Programmer's Manual                 LISTEN(2)

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)
```

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
- `backlog` specifies length of incoming connection queue
- returns 0 on success, -1 otherwise

Server system call: accept



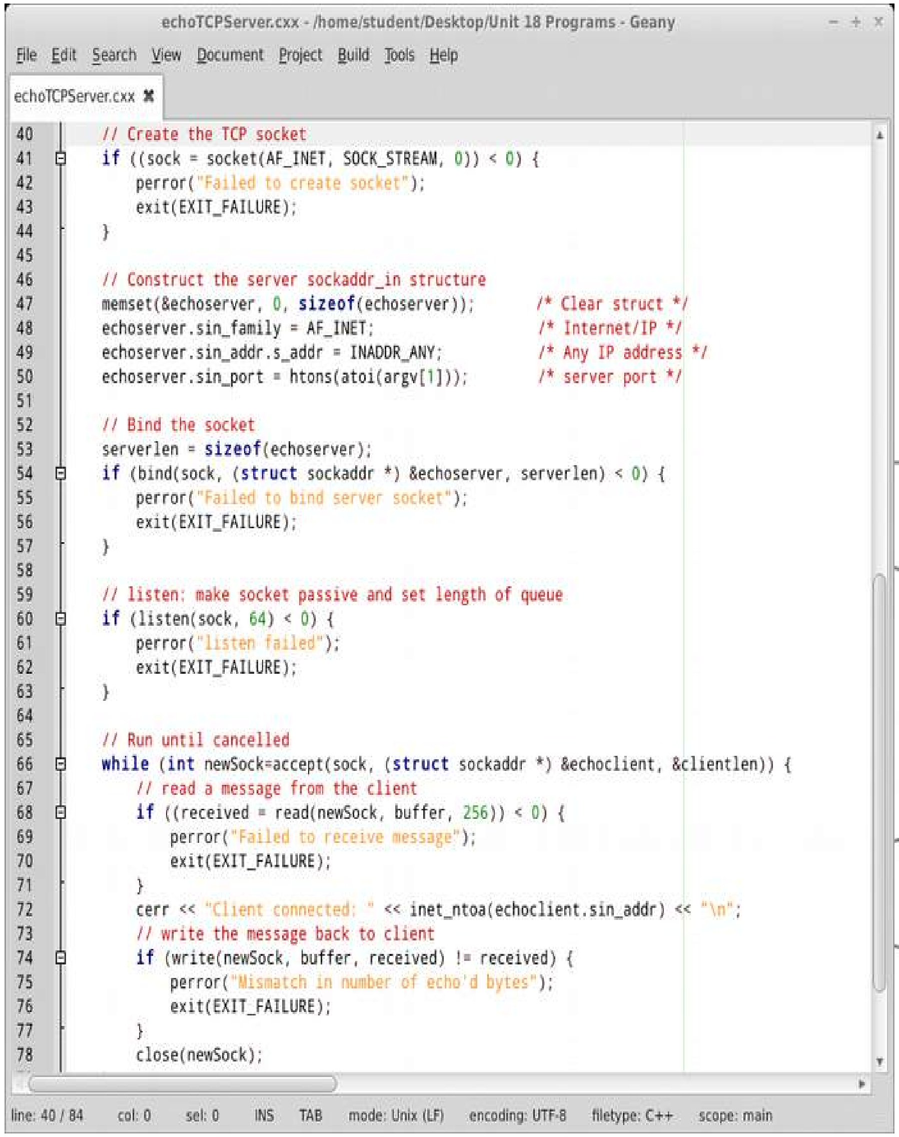
```
ACCEPT(2)                  Linux Programmer's Manual                 ACCEPT(2)

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)
```

Server system call: accept

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

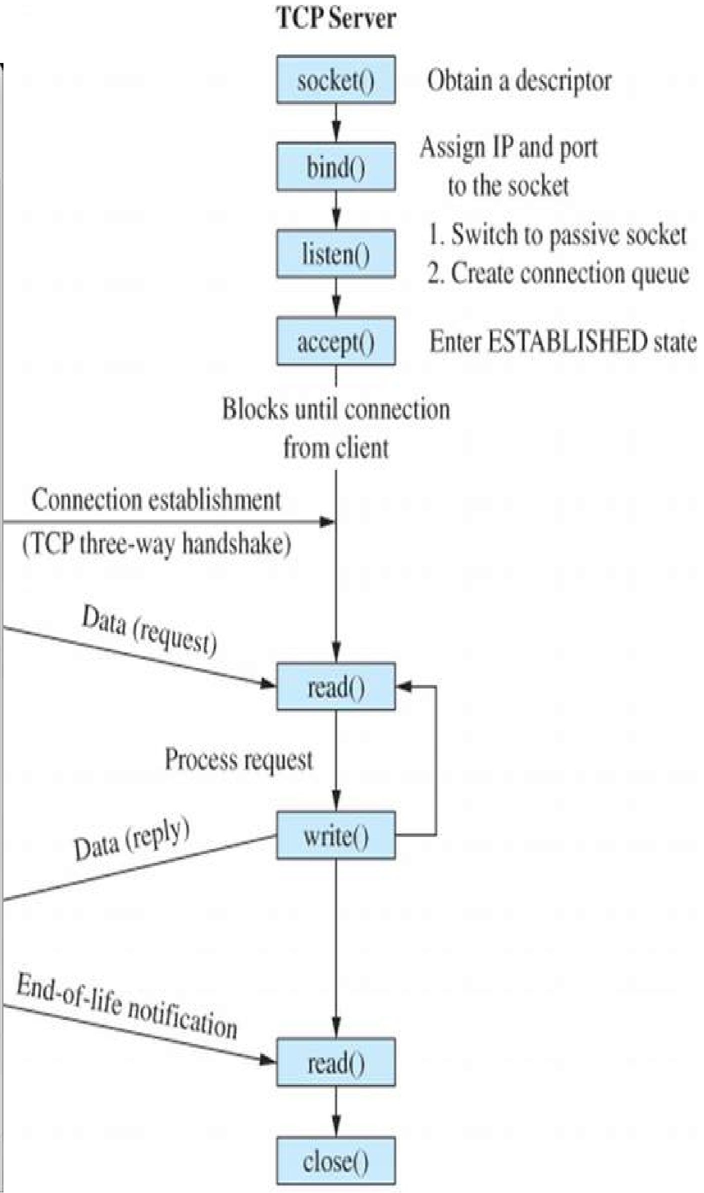
- extracts connection request from incoming queue
- creates a new connected socket
  - returns a new file descriptor for that socket, returns -1 on failure
- **`struct sockaddr`** holds address information
  - will accept **`struct sockaddr_in`** pointer
- **`addrlen`** specifies length of **`addr`**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

TCP server illustration


echoTCPServer.cxx - /home/student/Desktop/Unit 18 Programs - Geany
File Edit Search View Document Project Build Tools Help
echoTCPServer.cxx

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

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

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

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

// Run until cancelled
while (int newSock=accept(sock, (struct sockaddr *) &echoclient, &clientlen)) {
    // read a message from the client
    if ((received = read(newSock, buffer, 256)) < 0) {
        perror("Failed to receive message");
        exit(EXIT_FAILURE);
    }
    cerr << "Client connected: " << inet_ntoa(echoclient.sin_addr) << "\n";
    // write the message back to client
    if (write(newSock, buffer, received) != received) {
        perror("Mismatch in number of echo'd bytes");
        exit(EXIT_FAILURE);
    }
    close(newSock);
```

line: 40 / 84 col: 0 sel: 0 INS TAB mode: Unix (LF) encoding: UTF-8 filetype: C++ scope: main



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);
}
```

Server detail: bind the socket

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

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

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

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);
}
```

Server detail: accept new socket

```
// Run until cancelled
while (int newSock=accept(sock,
                 (struct sockaddr *) &echoclient,
                      &clientlen)) {

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

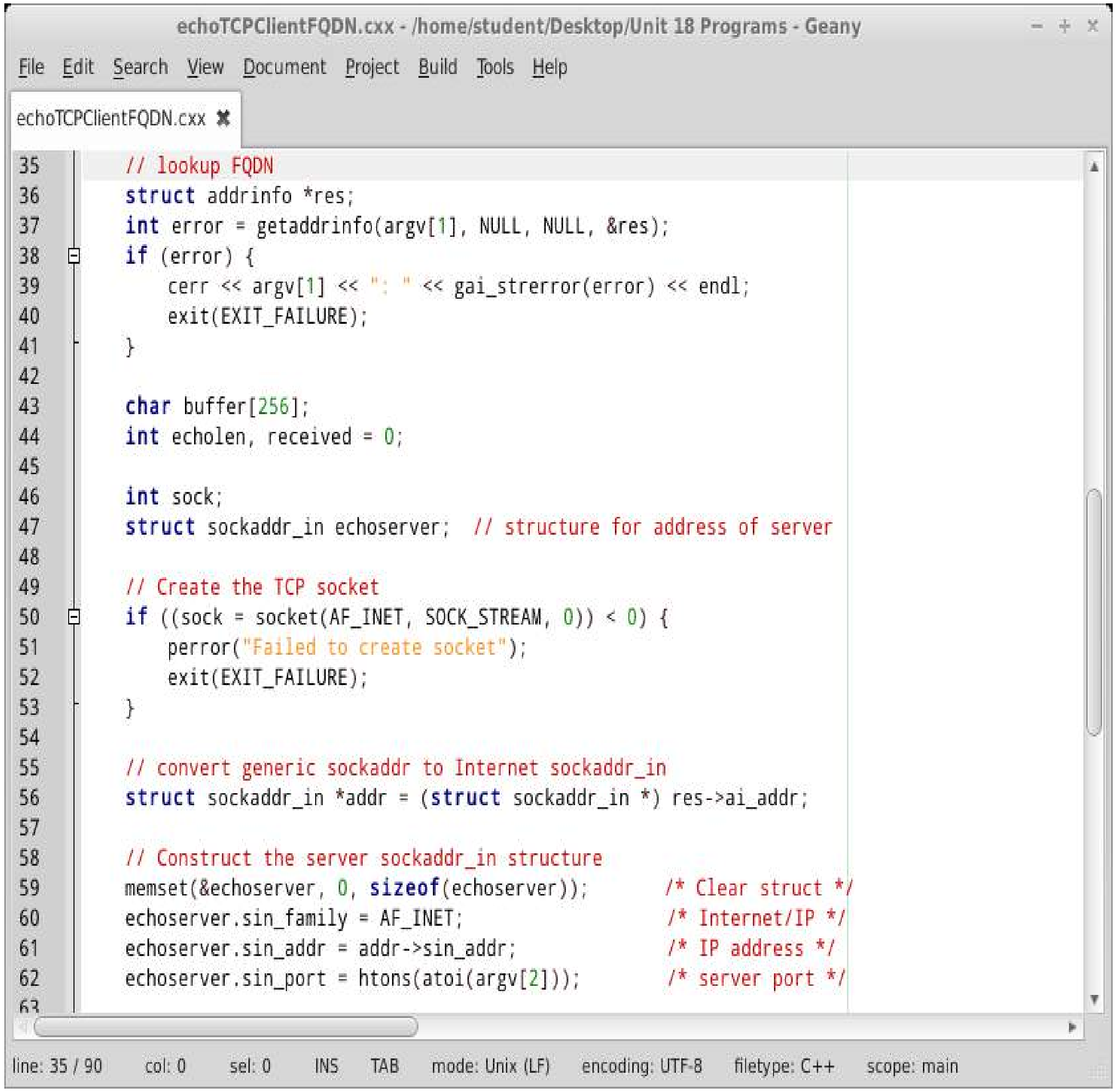
Server detail: read from socket

```
// read a message from the client
if ((received = read(newSock, buffer, 256)) < 0) {
   perror("Failed to receive message");
   exit(EXIT_FAILURE);
}
cerr << "Client connected: "
      << inet_ntoa(echoclient.sin_addr) << "\n";
```

Server detail: write to socket

```
// write the message back to client
if (write(newSock, buffer, received)
                   != received) {
    perror("Mismatch in number of bytes");
    exit(EXIT_FAILURE);
}
```

Illustration: echoTCPClientFQDN.cxx


echoTCPClientFQDN.cxx - /home/student/Desktop/Unit 18 Programs - Geany
File Edit Search View Document Project Build Tools Help
echoTCPClientFQDN.cxx
line: 35 / 90 col: 0 sel: 0 INS TAB mode: Unix (LF) encoding: UTF-8 filetype: C++ scope: main

```cpp
    // lookup FQDN
    struct addrinfo *res;
    int error = getaddrinfo(argv[1], NULL, NULL, &res);
    if (error) {
        cerr << argv[1] << ": " << gai_strerror(error) << endl;
        exit(EXIT_FAILURE);
    }

    char buffer[256];
    int echolen, received = 0;

    int sock;
    struct sockaddr_in echoserver;  // structure for address of server

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

    // convert generic sockaddr to Internet sockaddr_in
    struct sockaddr_in *addr = (struct sockaddr_in *) res->ai_addr;

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

Summary

- Transport layer
- Transmission control protocol
- TCP programming