CSCI 330
The UNIX System

User Datagram Protocol

Unit Overview

- Transport layer
- User datagram protocol
- UDP programming

Network Layer

- also called: Internet Protocol Layer
  - provides host to host transmission service,

    where hosts are not necessarily adjacent
- layer provides services:
  - addressing
  - hosts have global addresses: IPv4, IPv6
  - routing and forwarding
  - find path from host to host

Transport Layer

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

- 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

UDP

- simple message-based connectionless protocol
  - transmits information in one direction from source to destination without verifying the readiness or state of the receiver
- uses datagram as message
- stateless and fast

UDP packet format

bits	0 – 7	8 – 15	16 – 23	24 – 31
0	Source IP address			
32	Destination IP address			
64	Zeros	Protocol	UDP length	
96	Source Port		Destination Port	
128	Length		Checksum	
160+	Data			

UDP programming

- common abstraction: socket
- first introduced in BSD Unix in 1981

- socket is end-point of communication link
  - identified as IP address + port number
- operates as client and server

Socket system calls

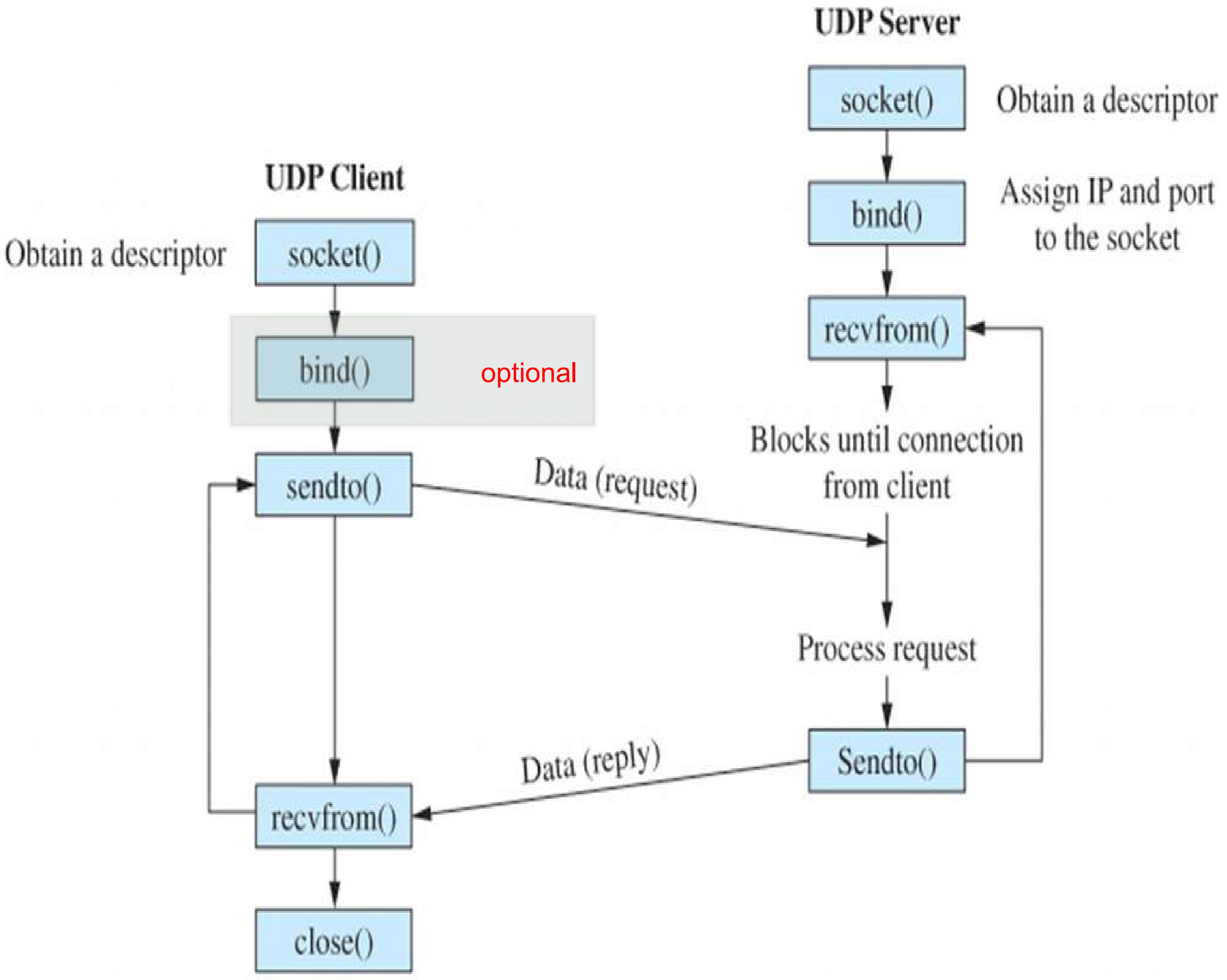
server

client

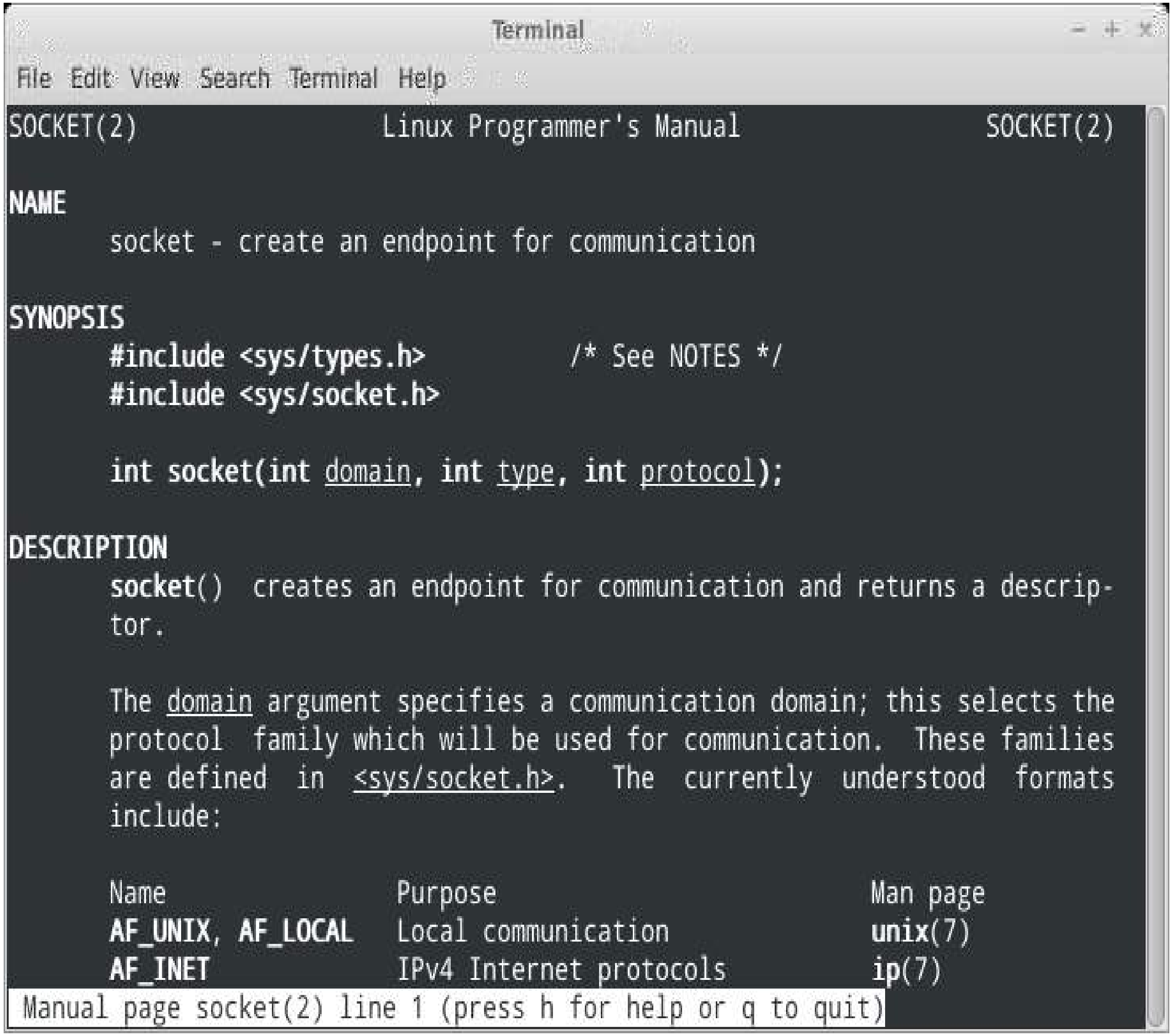
System call	Meaning
socket	Create a new communication endpoint
bind	Attach a local address to a socket
sendto	Send(write) some data over the connection
recvfrom	Receive(read) some data over the connection
close	Release the connection

optional

UDP communications pattern



System call: socket



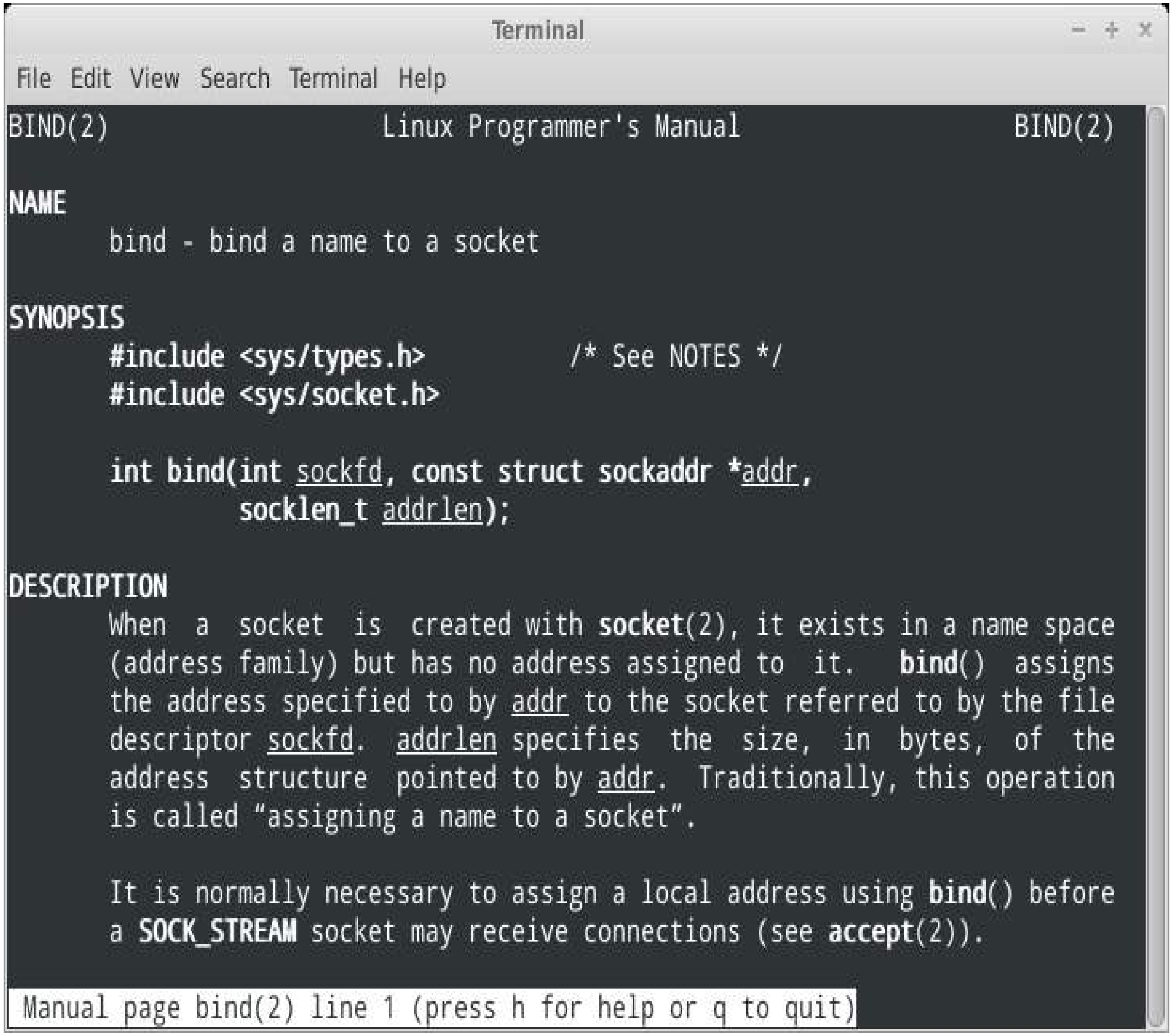
```
SOCKET(2)                  Linux Programmer's Manual                 SOCKET(2)

NAME
       socket - create an endpoint for communication

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

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

DESCRIPTION
       socket()  creates an endpoint for communication and returns a descrip-
       tor.

       The domain argument specifies a communication domain; this selects the
       protocol  family which will be used for communication.  These families
       are defined  in  <sys/socket.h>.   The  currently  understood  formats
       include:

       Name                Purpose                          Man page
       AF_UNIX, AF_LOCAL   Local communication              unix(7)
       AF_INET             IPv4 Internet protocols          ip(7)
 Manual page socket(2) line 1 (press h for help or q to quit)
```

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_DGRAM`** for datagrams
- **`protocol`** is set to 0, i.e. default UDP
- returns socket descriptor:
    - used in bind, sendto, recvfrom, close

System call: bind



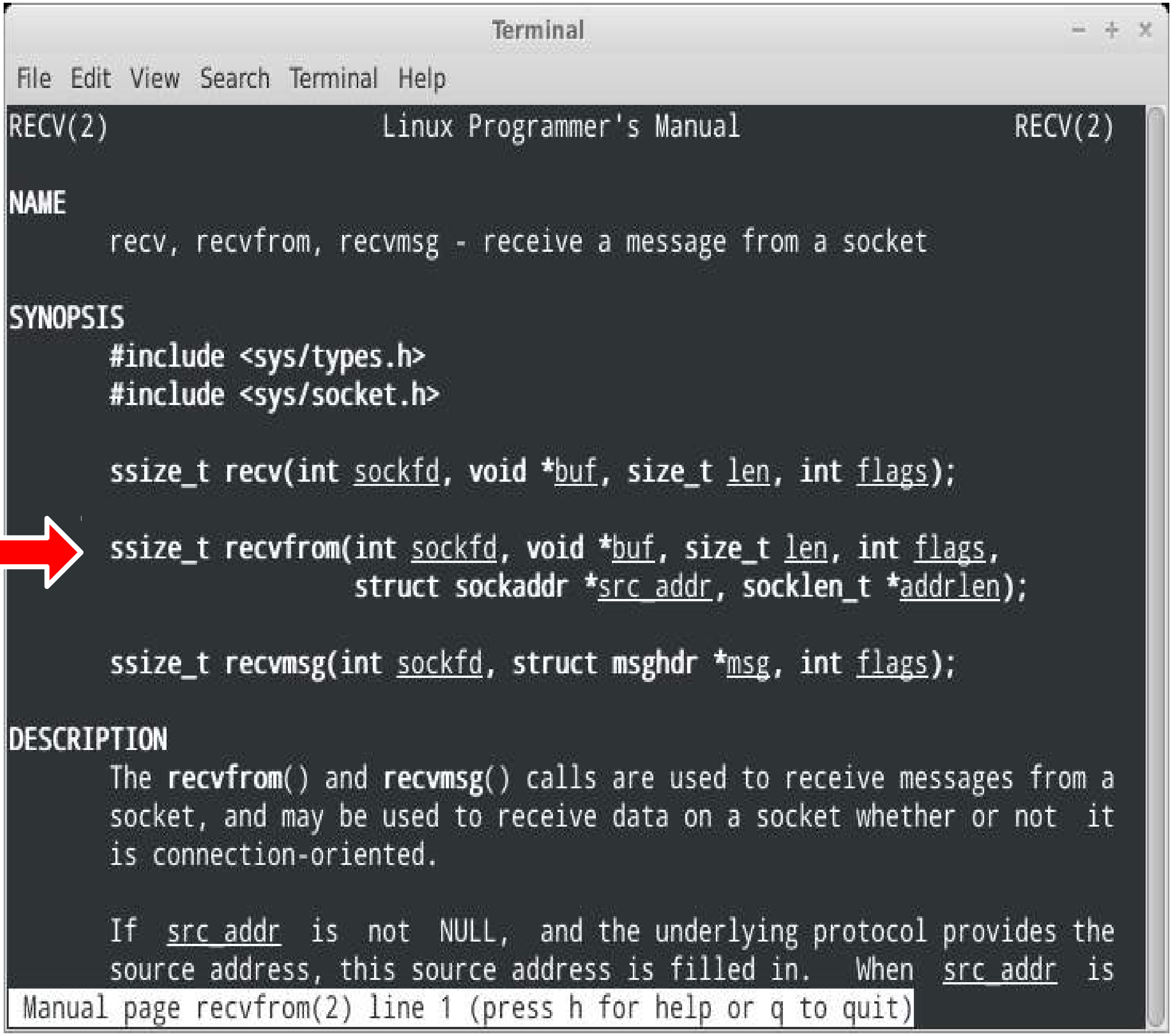
```
BIND(2)                    Linux Programmer's Manual                   BIND(2)

NAME
       bind - bind a name to a socket

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

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

DESCRIPTION
       When  a  socket  is  created with socket(2), it exists in a name space
       (address family) but has no address assigned to  it.   bind()  assigns
       the address specified to by addr to the socket referred to by the file
       descriptor sockfd.  addrlen specifies  the  size,  in  bytes,  of  the
       address  structure  pointed to by addr.  Traditionally, this operation
       is called “assigning a name to a socket”.

       It is normally necessary to assign a local address using bind() before
       a SOCK_STREAM socket may receive connections (see accept(2)).

 Manual page bind(2) line 1 (press h for help or q to quit)
```

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`
- `addrlen` specifies length of `addr` structure
- returns 0 on success, -1 otherwise

System call: recvfrom



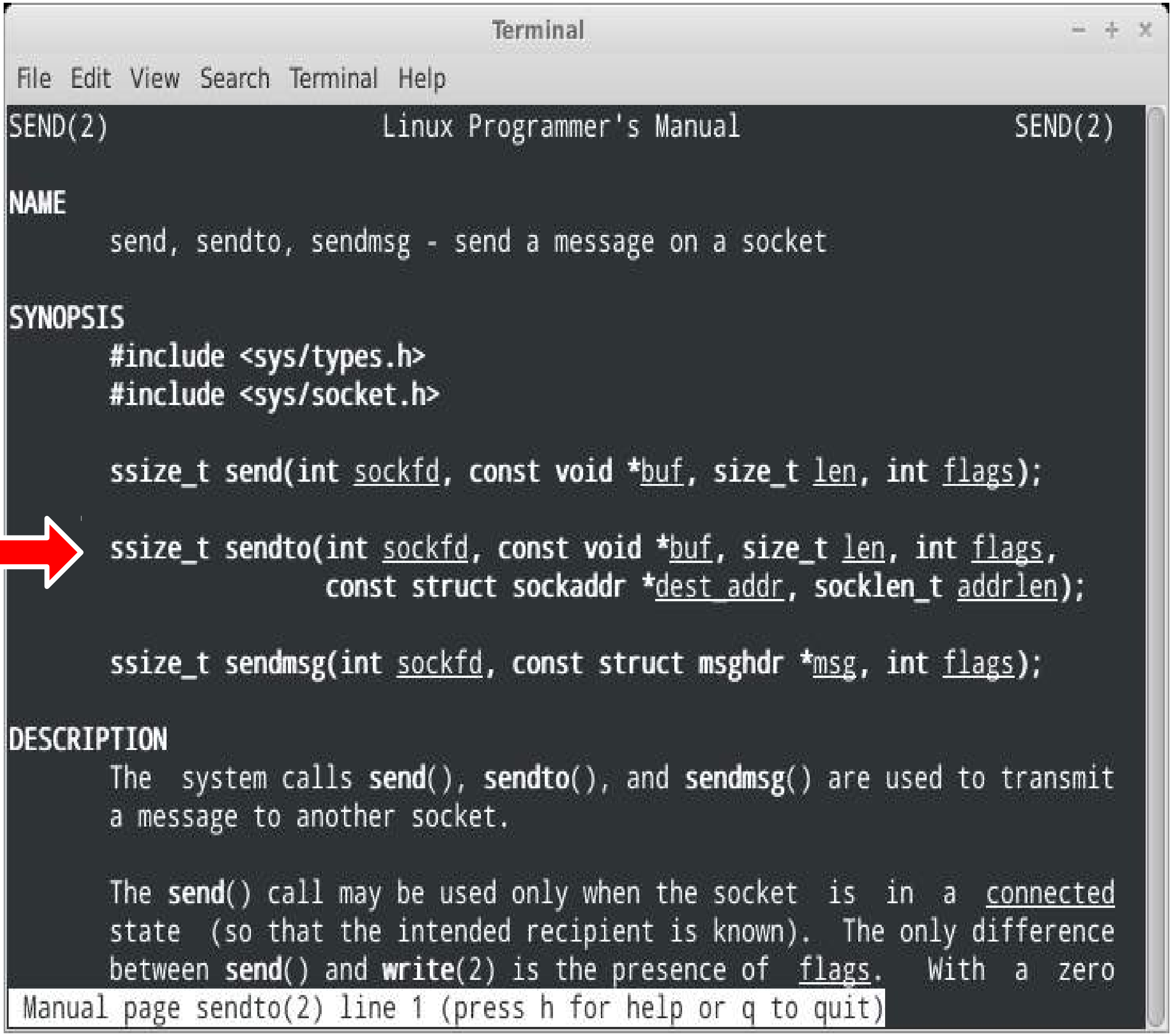
```
RECV(2)                    Linux Programmer's Manual                   RECV(2)

NAME
       recv, recvfrom, recvmsg - receive a message from a socket

SYNOPSIS
       #include <sys/types.h>
       #include <sys/socket.h>

       ssize_t recv(int sockfd, void *buf, size_t len, int flags);

       ssize_t recvfrom(int sockfd, void *buf, size_t len, int flags,
                        struct sockaddr *src_addr, socklen_t *addrlen);

       ssize_t recvmsg(int sockfd, struct msghdr *msg, int flags);

DESCRIPTION
       The recvfrom() and recvmsg() calls are used to receive messages from a
       socket, and may be used to receive data on a socket whether or not  it
       is connection-oriented.

       If  src_addr  is  not  NULL,  and the underlying protocol provides the
       source address, this source address is filled in.   When  src_addr  is
 Manual page recvfrom(2) line 1 (press h for help or q to quit)
```

System call: recvfrom

```
ssize_t recvfrom(int sockfd, void *buf, size_t len,
    int flags, struct sockaddr *src_addr,
                socklen_t *addrlen)
```

- receives a datagram **`buf`** of size **`len`** from socket
  - will wait until a datagram is available
  - flags specifies wait behavior, e.g.: 0 for default
- **`src_addr`** will hold address information of sender
  - **`struct sockaddr`** holds address information
  - **`addrlen`** specifies length of **`src_addr`** structure
- returns the number of bytes received, i.e. size of datagram

System call: sendto



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

NAME
       send, sendto, sendmsg - send a message on a socket

SYNOPSIS
       #include <sys/types.h>
       #include <sys/socket.h>

       ssize_t send(int sockfd, const void *buf, size_t len, int flags);

       ssize_t sendto(int sockfd, const void *buf, size_t len, int flags,
                      const struct sockaddr *dest_addr, socklen_t addrlen);

       ssize_t sendmsg(int sockfd, const struct msghdr *msg, int flags);

DESCRIPTION
       The  system calls send(), sendto(), and sendmsg() are used to transmit
       a message to another socket.

       The send() call may be used only when the socket  is  in  a  connected
       state  (so that the intended recipient is known).  The only difference
       between send() and write(2) is the presence of  flags.   With  a  zero
 Manual page sendto(2) line 1 (press h for help or q to quit)
```

System call: sendto

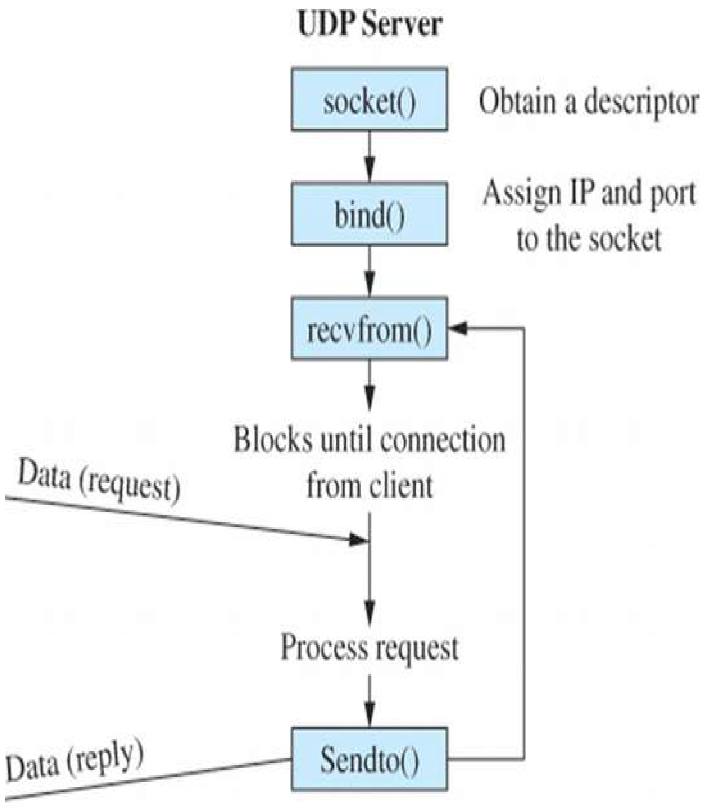
```
ssize_t sendto(int sockfd,
  const void *buf, size_t len, int flags,
  const struct sockaddr *dest_addr, socklen_t
addrlen)
```

- sends datagram **`buf`** of size **`len`** to socket
  - will wait if there is no ready receiver
  - flags specifies wait behavior, e.g.: 0 for default
- **`dest_addr`** holds address information of receiver
  - **`struct sockaddr`** holds address information
  - **`addrlen`** specifies length of **`dest_addr`** structure

UDP Programming

- simple server: echo
  - sends all received datagrams back to sender
- simple client
  - send datagram to server

Illustration: echoServer.cc



UDP Server

socket()

Obtain a descriptor

bind()

Assign IP and port
to the socket

recvfrom()

Blocks until connection
from client

Data (request)

Process request

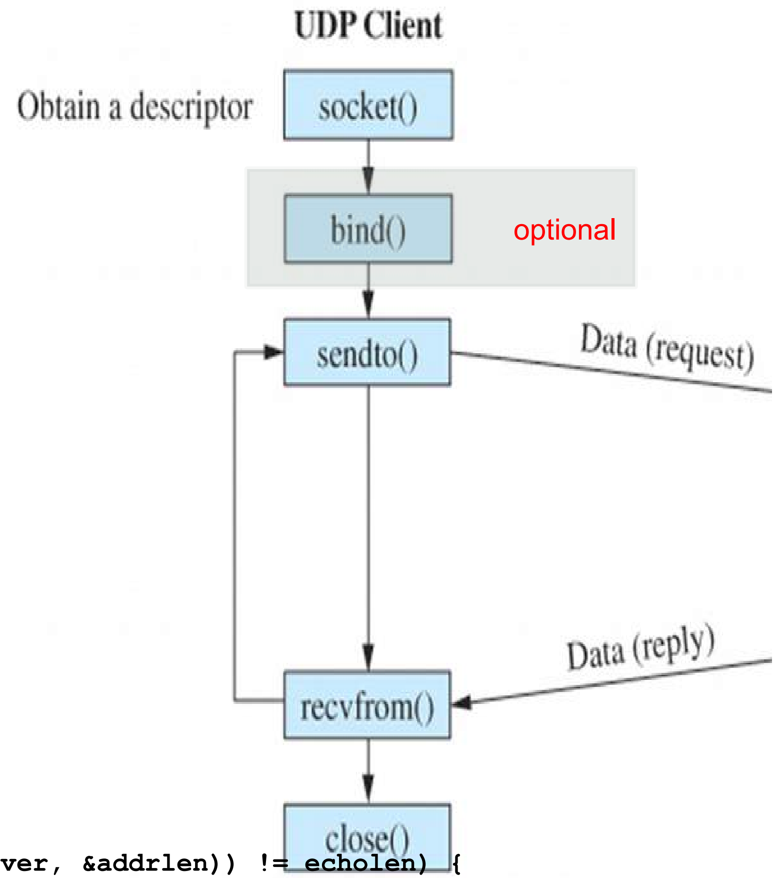
Data (reply)

Sendto()

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

// Create the UDP socket
if ((sock = socket(AF_INET, SOCK_DGRAM, 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("Faled to bind server socket"); exit(EXIT_FAILURE); }

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

close(sock);
```

Illustration: echoClient.cc

```
int main(int argc, char * argv[])
{
  if(argc != 4) {
    cerr << "Usage: echoClient server_ip port message\n";
    exit(EXIT_FAILURE);
  }
  char buffer[256]; int echolen, received = 0;  unsigned int addrlen;
  int sock;
  struct sockaddr_in echoserver; // structure for server address

  // Create the UDP socket
  if ((sock = socket(AF_INET, SOCK_DGRAM, 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

  // Send the message to the server
  echolen = strlen(argv[3]);
  if (sendto(sock, argv[3], strlen(argv[3]), 0,
      (struct sockaddr *) &echoserver, sizeof(echoserver)) != echolen) {
    perror("Mismatch in number of sent bytes"); exit(EXIT_FAILURE);}

  // Receive the message back from the server
  addrlen = sizeof(echoserver);
  if ((received = recvfrom(sock, buffer, 256, 0, (struct sockaddr *) &echoserver, &addrlen)) != echolen) {
    perror("Mismatch in number of received bytes"); exit(EXIT_FAILURE); }

  buffer[received] = '\0';
  cout << "Message received: " << buffer << endl;
}
```



Detail: create UDP socket

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

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 */
```

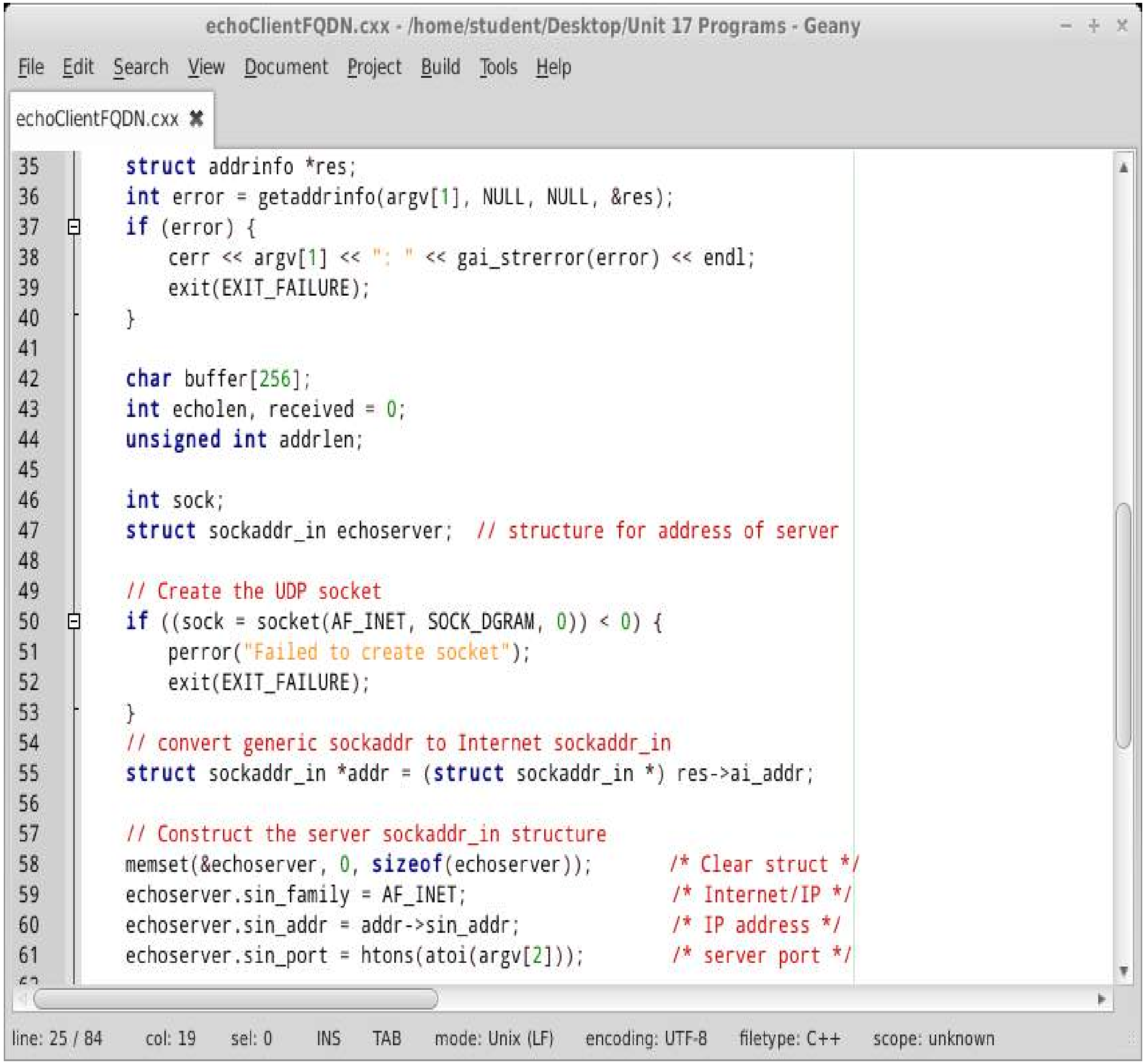
Detail: receive from socket

```
addrlen = sizeof(echoserver);
received = recvfrom(sock, buffer, 256, 0,
              (struct sockaddr *) &echoserver, &addrlen);


cout << "Received: << received bytes\n";


buffer[received] = '\0';   /* Assure null-terminated string */


cout << "Server ("
     << inet_ntoa(echoserver.sin_addr)
     << ") echoed: " << buffer << endl;
```

Detail: send to 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 */

// Send the message to the server
echolen = strlen(argv[3]);
if (sendto(sock, argv[3], strlen(argv[3]), 0,
     (struct sockaddr *) &echoserver, sizeof(echoserver))
                                   != echolen) {
     perror("Mismatch in number of sent bytes");
     exit(EXIT_FAILURE);
}
```

Illustration: echoClientFQDN.cxx


echoClientFQDN.cxx - /home/student/Desktop/Unit 17 Programs - Geany
File Edit Search View Document Project Build Tools Help
echoClientFQDN.cxx

```
    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;
    unsigned int addrlen;

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

    // Create the UDP socket
    if ((sock = socket(AF_INET, SOCK_DGRAM, 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 */
```

line: 25 / 84   col: 19   sel: 0   INS   TAB   mode: Unix (LF)   encoding: UTF-8   filetype: C++   scope: unknown

Summary

- Transport layer
- User datagram protocol
- UDP programming