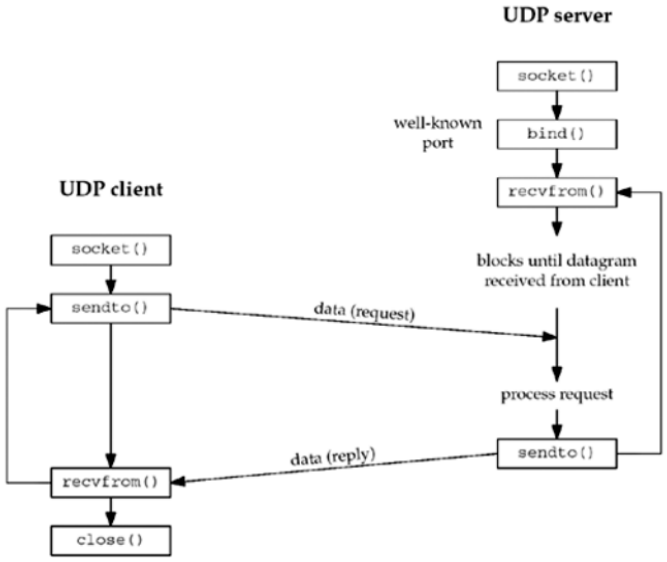
# UNIT – 3

**1. Illustrate the significance of socket functions for UDP TCP client/server with a neat block diagram.**

**Figure 8.1. Socket functions for UDP client/server.**




- The Figure shows the function calls for a typical UDP client/server.
- The client does not establish a connection with the server. Instead, the client just sends a datagram to the server using the **sendto** function (described in the next section), which requires the address of the destination (the server) as a parameter.
- Similarly, the server does not accept a connection from a client. Instead, the server just calls the **recvfrom** function, which waits until data arrives from some client.
- **recvfrom** returns the protocol address of the client, along with the datagram, so the server can send a response to the correct client.
- Figure shows a timeline of the typical scenario that takes place for a UDP client/server exchange.

**2.Explain the following functions of UDP socket:**

- **recvfrom**
- **sendto**

These two functions are similar to the standard `read` and `write` functions, but three additional arguments are required.

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

```
ssize_t recvfrom(int sockfd, void *buff, size_t nbytes, int flags, struct sockaddr *
from, socklen_t *addrlen);
```

```
ssize_t sendto(int sockfd, const void *buff, size_t nbytes, int flags, const struct
sockaddr *to, socklen_t addrlen);
```

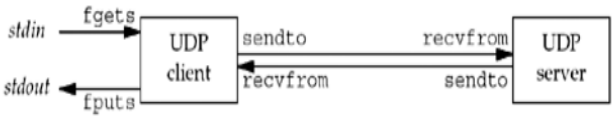
Both return: number of bytes read or written if OK, 1 on error

- The first three arguments, **sockfd**, **buff**, and **nbytes**, are identical to the first three arguments for **read** and **write**: descriptor, pointer to buffer to read into or write from, and number of bytes to read or write.
- Both functions return the length of the data that was read or written as the value of the function. In the typical use of **recvfrom**, with a datagram protocol, the return value is the amount of user data in the datagram received.

**3. List and explain with a neat block diagram the steps associated with simple UDP echo client and server.**

**4. Develop the 'C' program to demonstrate the UDP echo server: main function**

**Figure 8.2. Simple echo client/server using UDP.**




## Figure 8.3 UDP echo server.

*udpcliserv/udpserv01.c*

```
 1 #include     "unp.h"

 2 int
 3 main(int argc, char **argv)
 4 {
 5     int     sockfd;
 6     struct sockaddr_in servaddr, cliaddr;

 7     sockfd = Socket(AF_INET, SOCK_DGRAM, 0);

 8     bzero(&servaddr, sizeof(servaddr));
 9     servaddr.sin_family = AF_INET;
10     servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
11     servaddr.sin_port = htons(SERV_PORT);

12     Bind(sockfd, (SA *) &servaddr, sizeof(servaddr));

13     dg_echo(sockfd, (SA *) &cliaddr, sizeof(cliaddr));
14 }
```

### Create UDP socket, bind server's well-known port

*7 12* We create a UDP socket by specifying the second argument to `socket` as `SOCK_DGRAM` (a datagram socket in the IPv4 protocol). As with the TCP server example, the IPv4 address for the `bind` is specified as `INADDR_ANY` and the server's well-known port is the constant `SERV_PORT` from the `unp.h` header.

*13* The function `dg_echo` is called to perform server processing.

**5. Develop the 'C' program to demonstrate the UDP echo server: dg_echo function**

## Figure 8.4 dg_echo function: echo lines on a datagram socket.

*lib/dg_echo.c*

```
 1 #include     "unp.h"

 2 void
 3 dg_echo(int sockfd, SA *pcliaddr, socklen_t clilen)
 4 {
 5     int     n;
 6     socklen_t len;
 7     char    mesg[MAXLINE];

 8     for ( ; ; ) {
 9         len = clilen;
10         n = Recvfrom(sockfd, mesg, MAXLINE, 0, pcliaddr, &len);

11         Sendto(sockfd, mesg, n, 0, pcliaddr, len);
12     }
13 }
```

*8 12* This function is a simple loop that reads the next datagram arriving at the server's port using `recvfrom` and sends it back using `sendto`.

Next, this function provides an *iterative server*, not a concurrent server as we had with TCP. There is no call to `fork`, so a single server process handles any and all clients. In general, most TCP servers are concurrent and most UDP servers are iterative.

**6. Develop the 'C' program to demonstrate the UDP echo client: main function**

## Figure 8.7 UDP echo client.

*udpcliserv/udpcli01.c*

```
 1 #include     "unp.h"

 2 int
 3 main(int argc, char **argv)
 4 {
 5     int     sockfd;
 6     struct sockaddr_in servaddr;

 7     if(argc != 2)
 8        err_quit("usage: udpcli <IPaddress>");

 9     bzero(&servaddr, sizeof(servaddr));
10     servaddr.sin_family = AF_INET;
11     servaddr.sin_port = htons(SERV_PORT);
12     Inet_pton(AF_INET, argv[1], &servaddr.sin_addr);

13     sockfd = Socket(AF_INET, SOCK_DGRAM, 0);

14     dg_cli(stdin, sockfd, (SA *) &servaddr, sizeof(servaddr));

15     exit(0);
16 }
```

*9 12* An IPv4 socket address structure is filled in with the IP address and port number of the server. This structure will be passed to `dg_cli`, specifying where to send datagrams.

*13 14* A UDP socket is created and the function `dg_cli` is called.

**7. Develop the 'C' program to demonstrate the UDP echo client: dg_cli function**

**Figure 8.8 `dg_cli` function: client processing loop.**

*lib/dg_cli.c*

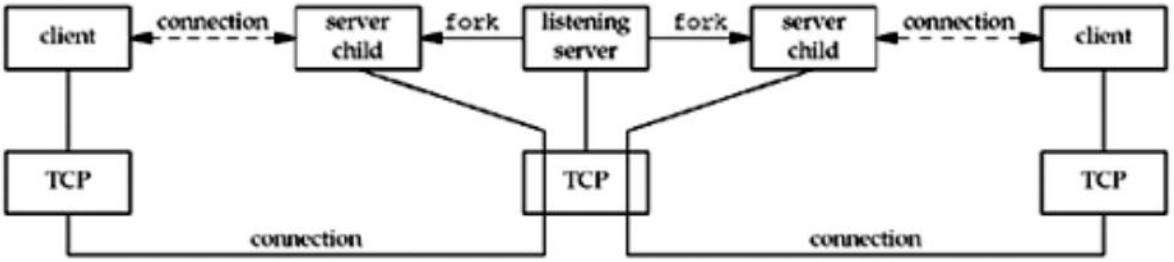
```
 1 #include     "unp.h"

 2 void
 3 dg_cli(FILE *fp, int sockfd, const SA *pservaddr, socklen_t servlen)
 4 {
 5     int     n;
 6     char    sendline[MAXLINE], recvline[MAXLINE + 1];

 7     while (Fgets(sendline, MAXLINE, fp) != NULL) {

 8         Sendto(sockfd, sendline, strlen(sendline), 0, pservaddr, servlen);

 9         n = Recvfrom(sockfd, recvline, MAXLINE, 0, NULL, NULL);

10         recvline[n] = 0;        /* null terminate */
11         Fputs(recvline, stdout);
12     }
13 }
```

*7 12* There are four steps in the client processing loop: read a line from standard input using `fgets`, send the line to the server using `sendto`, read back the server's echo using `recvfrom`, and print the echoed line to standard output using `fputs`.

Notice that the call to recvfrom specifies a null pointer as the fifth and sixth arguments. This tells the kernel that we are not interested in knowing who sent the reply.

**8. Outline the summary of TCP client/server with two clients.**

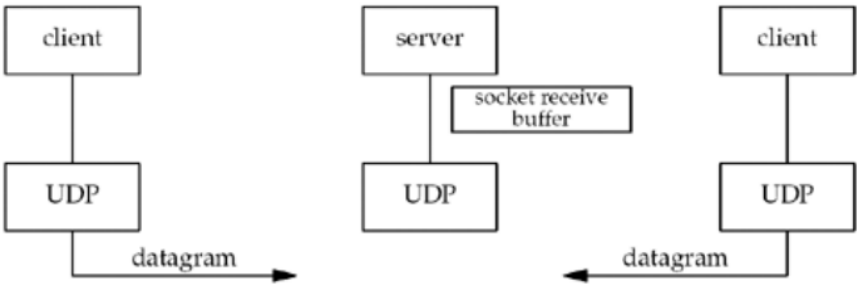
**Figure 8.5. Summary of TCP client/server with two clients.**




There are two connected sockets and each of the two connected sockets on the server host has its own socket receive buffer.

**9. Outline the summary of UDP client/server with two clients.**

## Figure 8.6. Summary of UDP client/server with two clients.




There is only one server process and it has a single socket on which it receives all arriving datagrams and sends all responses. That socket has a receive buffer into which all arriving datagrams are placed.

**10. Develop the 'C' program for dg_cli function that verifies returned socket address.**

## Figure 8.8 `dg_cli` function: client processing loop.

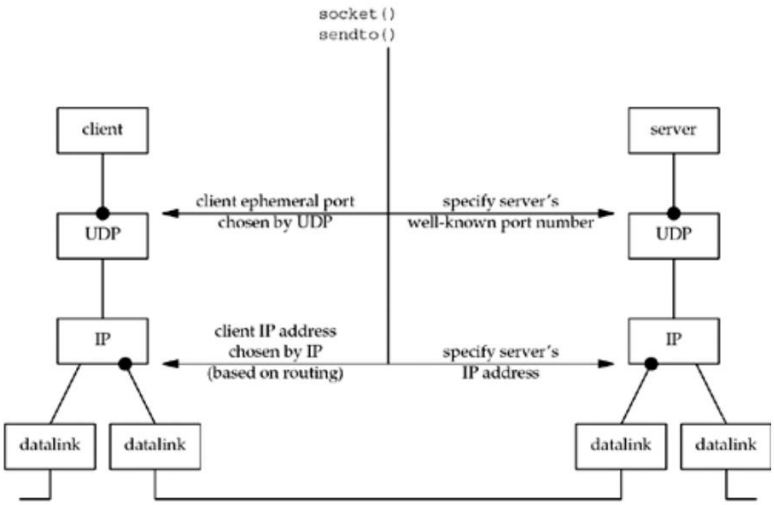
*lib/dg_cli.c*

```
 1 #include     "unp.h"

 2 void
 3 dg_cli(FILE *fp, int sockfd, const SA *pservaddr, socklen_t servlen)
 4 {
 5     int     n;
 6     char    sendline[MAXLINE], recvline[MAXLINE + 1];

 7     while (Fgets(sendline, MAXLINE, fp) != NULL) {

 8         Sendto(sockfd, sendline, strlen(sendline), 0, pservaddr, servlen);

 9         n = Recvfrom(sockfd, recvline, MAXLINE, 0, NULL, NULL);

10         recvline[n] = 0;        /* null terminate */
11         Fputs(recvline, stdout);
12     }
13 }
```

*7 12* There are four steps in the client processing loop: read a line from standard input using `fgets`, send the line to the server using `sendto`, read back the server's echo using `recvfrom`, and print the echoed line to standard output using `fputs`.

**11. Outline the summary of UDP client/server from client's perspective with a neat block diagram.**

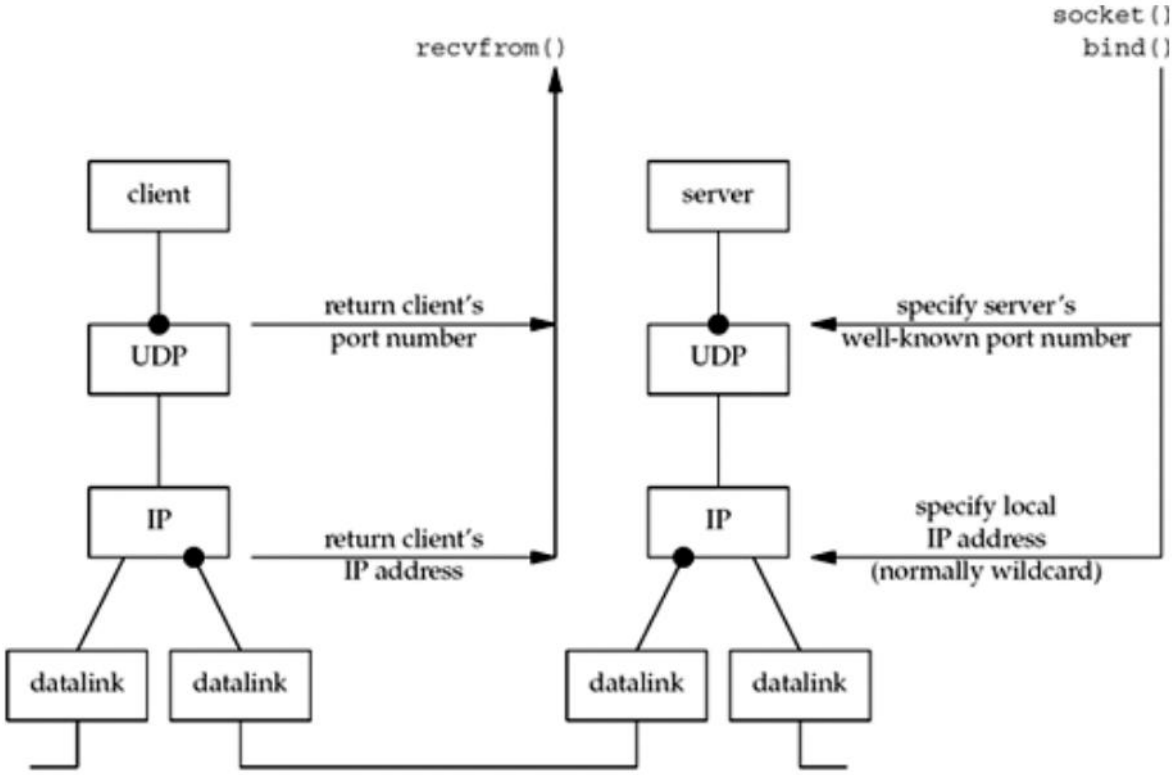
**Figure 8.11. Summary of UDP client/server from client's perspective.**




- The client must specify the server's IP address and port number for the call to **sendto**. Normally, the client's IP address and port are chosen automatically by the kernel, although we mentioned that the client can call **bind** if it so chooses.
- If these two values for the client are chosen by the kernel, we also mentioned that the client's ephemeral port is chosen once, on the first **sendto**, and then it never changes.
- The client's IP address, however, can change for every UDP datagram that the client sends, assuming the client does not bind a specific IP address to the socket.
- The reason is shown in Figure : If the client host is multihomed, the client could alternate between two destinations, one going out the datalink on the left, and the other going out the datalink on the right.
- What happens if the client **binds** an IP address to its socket, but the kernel decides that an outgoing datagram must be sent out some other datalink? In this case the IP datagram will contain a source IP address that is different from the IP address of the outgoing datalink

**12. Outline the summary of UDP client/server from server's perspective with a neat block diagram.**

**Figure 8.12. Summary of UDP client/server from server's perspective.**




There are at least four pieces of information that a server might want to know from an arriving IP datagram: **the source IP address**, **destination IP address**, **source port number**, and **destination port number.** Figure 8.13 shows the function calls that return this information for a TCP server and a UDP server.

**Figure 8.13. Information available to server from arriving IP datagram.**

| From client's IP datagram | TCP server | UDP server |
|---|---|---|
| Source IP address | `accept` | `recvfrom` |
| Source port number | `accept` | `recvfrom` |
| Destination IP address | `getsockname` | `recvmsg` |
| Destination port number | `getsockname` | `getsockname` |

**13. Develop the 'C' program to demonstrate the UDP dg_cli function that calls connect.**

## Figure 8.17 dg_cli function that calls connect.

*udpcliserv/dgcliconnect.c*

```
 1 #include     "unp.h"

 2 void
 3 dg_cli(FILE *fp, int sockfd, const SA *pservaddr, socklen_t servlen)
 4 {
 5     int     n;
 6     char    sendline[MAXLINE], recvline[MAXLINE + 1];

 7     Connect(sockfd, (SA *) pservaddr, servlen);

 8     while (Fgets(sendline, MAXLINE, fp) != NULL) {

 9         Write(sockfd, sendline, strlen(sendline));

10         n = Read(sockfd, recvline, MAXLINE);

11         recvline[n] = 0;        /* null terminate */
12         Fputs(recvline, stdout);
13     }
14 }
```

**14. Develop the 'C' program to demonstrate the UDP dg_cli function that writes a fixed number of datagrams to the server.**

## Figure 8.19 dg_cli function that writes a fixed number of datagrams to the server.

*udpcliserv/dgcliloop1.c*

```
 1 #include     "unp.h"

 2 #define NDG      2000          /*  datagrams to send */
 3 #define DGLEN    1400          /*  length of each datagram */

 4 void
 5 dg_cli(FILE *fp, int sockfd, const SA *pservaddr, socklen_t servlen)
 6 {
 7     int     i;
 8     char    sendline[DGLEN];

 9     for (i = 0; i < NDG; i++) {
10         Sendto(sockfd, sendline, DGLEN, 0, pservaddr, servlen);
11     }
12 }
```

**15. Develop the 'C' program to demonstrate the UDP dg_echo function that counts received datagrams.**

## Figure 8.20 dg_echo function that counts received datagrams.

*udpcliserv/dgecholoop1.c*

```
 1 #include     "unp.h"

 2 static void recvfrom_int(int);
 3 static int count;

 4 void
 5 dg_echo(int sockfd, SA *pcliaddr, socklen_t clilen)
 6 {
 7     socklen_t len;
 8     char    mesg[MAXLINE];

 9     Signal(SIGINT, recvfrom_int);

10     for ( ; ; ) {
11         len = clilen;
12         Recvfrom(sockfd, mesg, MAXLINE, 0, pcliaddr, &len);

13         count++;
14     }
15 }

16 static void
17 recvfrom_int(int signo)
18 {
19     printf("\nreceived %d datagrams\n", count);
20     exit(0);
21 }
```

**16. Develop the 'C' program to demonstrate the UDP dg_echo function that increases the size of the socket receive queue.**

## Figure 8.22 dg_echo function that increases the size of the socket receive queue.

*udpcliserv/dgecholoop2.c*

```
 1 #include    "unp.h"

 2 static void recvfrom_int(int);
 3 static int count;

 4 void
 5 dg_echo(int sockfd, SA *pcliaddr, socklen_t clilen)
 6 {
 7     int     n;
 8     socklen_t len;
 9     char    mesg[MAXLINE];

10     Signal(SIGINT, recvfrom_int);

11     n = 220 * 1024;
12     Setsockopt(sockfd, SOL_SOCKET, SO_RCVBUF, &n, sizeof(n));

13     for ( ; ; ) {
14         len = clilen;
15         Recvfrom(sockfd, mesg, MAXLINE, 0, pcliaddr, &len);

16         count++;
17     }
18 }

19 static void
20 recvfrom_int(int signo)
21 {
22     printf("\nreceived %d datagrams\n", count);


23     exit(0);
24 }
```

**17. Develop the 'C' program for UDP that uses connect to determine outgoing interface.**

**Figure 8.23 UDP program that uses `connect` to determine outgoing interface.**

*udpcliserv/udpcli09.c*

```
 1 #include     "unp.h"

 2 int
 3 main(int argc, char **argv)
 4 {
 5     int     sockfd;
 6     socklen_t len;
 7     struct sockaddr_in cliaddr, servaddr;

 8     if (argc != 2)
 9         err_quit("usage: udpcli <IPaddress>");

10     sockfd = Socket(AF_INET, SOCK_DGRAM, 0);

11     bzero(&servaddr, sizeof(servaddr));
12     servaddr.sin_family = AF_INET;
13     servaddr.sin_port = htons(SERV_PORT);
14     Inet_pton(AF_INET, argv[1], &servaddr.sin_addr);

15     Connect(sockfd, (SA *) &servaddr, sizeof(servaddr));

16     len = sizeof(cliaddr);
17     Getsockname(sockfd, (SA *) &cliaddr, &len);
18     printf("local address %s\n", Sock_ntop((SA *) &cliaddr, len));

19     exit(0);
20 }
```

**18. Make use of select function for TCP and UDP Echo server.**