Assignment 6-2. Implement a DNS server using UDP socket.

What is UDP Protocol?

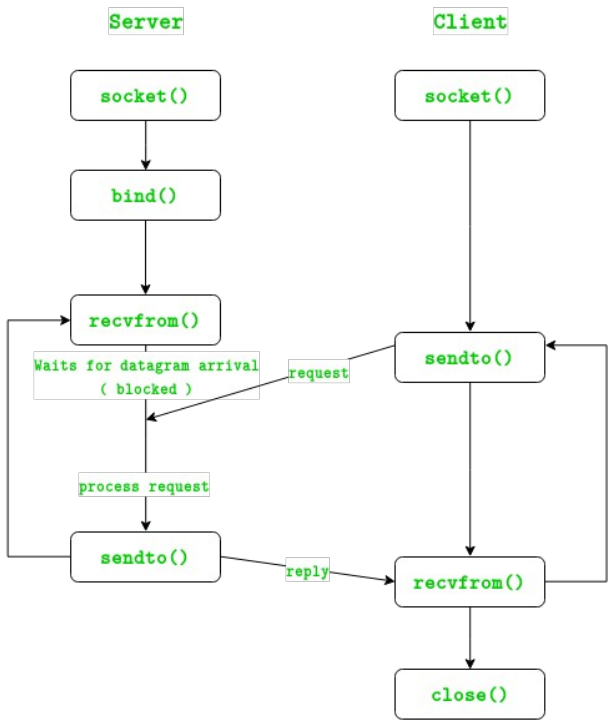
Udp is a conectionless data sending protocol. In UDP, the client does not form a connection with the server like in TCP and instead just sends a datagram. Similarly, the server need not accept a connection and just waits for datagrams to arrive. Datagrams upon arrival contain the address of sender which the server uses to send data to the correct client.
UDP uses a simple connectionless communication model with a minimum of protocol mechanism. UDP provides checksums for data integrity, and port numbers for addressing different functions at the source and destination of the datagram. It has no handshaking dialogues, and thus exposes the user's program to any unreliability of the underlying network; There is no guarantee of delivery, ordering, or duplicate protection.

Difference between TCP and UCP:

TCP stands for Transmission Control Protocol. It is the most commonly used protocol on the Internet. TCP guarantees the recipient will receive the packets in order by numbering them. The recipient sends messages back to the sender saying it received the messages. If the sender does not get a correct response, it will resend the packets to ensure the recipient received them. Packets are also checked for errors. TCP is all about this reliability — packets sent with TCP are tracked so no data is lost or corrupted in transit. This is why file downloads do not become corrupted even if there are network hiccups. Of course, if the recipient is completely offline, your computer will give up and you will see an error message saying it can not communicate with the remote host.

UDP stands for User Datagram Protocol — a datagram is the same thing as a packet of information. When using UDP, packets are just sent to the recipient. The sender will not wait to make sure the recipient received the packet — it will just continue sending the next packets. If you are the recipient and you miss some UDP packets, too bad — you can not ask for those packets again. There is no guarantee you are getting all the packets and there is no way to ask for a packet again if you miss it, but losing all this overhead means the computers can communicate more quickly.

UDP is used when speed is desirable and error correction is not necessary. For example, UDP is frequently used for live broadcasts and online games.



UDP Server :

1. Create UDP socket.
2. Bind the socket to server address.
3. Wait until datagram packet arrives from client.
4. Process the datagram packet and send a reply to client.
5. Go back to Step 3.

UDP Client :

1. Create UDP socket.
2. Send message to server.
3. Wait until response from server is recieved.
4. Process reply and go back to step 2, if necessary.
5. Close socket descriptor and exit.

This is the UDPServer in c program.

```
14 int main() {
15     int sockfd;
16     char buffer[MAXLINE];
17     char *hello = "Hello from server";
18     struct sockaddr_in servaddr, cliaddr;
19
20     // Creating socket file descriptor
21     if ( (sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0 ) {
22         perror("socket creation failed");
23         exit(EXIT_FAILURE);
24     }
25
26     memset(&servaddr, 0, sizeof(servaddr));
27     memset(&cliaddr, 0, sizeof(cliaddr));
28
29     // Filling server information
30     servaddr.sin_family    = AF_INET; // IPv4
31     servaddr.sin_addr.s_addr = INADDR_ANY;
32     servaddr.sin_port = htons(PORT);
33
34     // Bind the socket with the server address
35     if ( bind(sockfd, (const struct sockaddr *)&servaddr,
36             sizeof(servaddr)) < 0 )
37     {
38         perror("bind failed");
39         exit(EXIT_FAILURE);
40     }
```

The message is Hello from Server.

If sockfd <0 socket creation failed.

Servaddr and cliaddr is initialised to 0.

The binding of the servaddr and sockaddr is done.

```
42      int len, n;
43      n = recvfrom(sockfd, (char *)buffer, MAXLINE,
44                  MSG_WAITALL, ( struct sockaddr *) &cliaddr,
45                  &len);
46      buffer[n] = '\0';
47      printf("Client : %s\n", buffer);
48      sendto(sockfd, (const char *)hello, strlen(hello),
49          MSG_CONFIRM, (const struct sockaddr *) &cliaddr,
50              len);
51      printf("Hello message sent.\n");
52
53      return 0;
54  }
```

This is received from the client.

This is required to send the message to the client.

```
14  int main() {
15      int sockfd;
16      char buffer[MAXLINE];
17      char *hello = "Hello from client";
18      struct sockaddr_in     servaddr;
19
20      // Creating socket file descriptor
21      if ( (sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0 ) {
22          perror("socket creation failed");
23          exit(EXIT_FAILURE);
24      }
25
26      memset(&servaddr, 0, sizeof(servaddr));
27
28      // Filling server information
29      servaddr.sin_family = AF_INET;
30      servaddr.sin_port = htons(PORT);
31      servaddr.sin_addr.s_addr = INADDR_ANY;
32
33      int n, len;
34
35      sendto(sockfd, (const char *)hello, strlen(hello),
36          MSG_CONFIRM, (const struct sockaddr *) &servaddr,
37              sizeof(servaddr));
38      printf("Hello message sent.\n");
39
40      n = recvfrom(sockfd, (char *)buffer, MAXLINE,
41                  MSG_WAITALL, (struct sockaddr *) &servaddr,
42                  &len);
43      buffer[n] = '\0';
44      printf("Server : %s\n", buffer);
45
46      close(sockfd);
47      return 0;
48  }
```

This is the client side program. UDPClient.c

Socket is created here.

Servaddr is initialised to 0.

Server information needed to implement sockets.

This is required to send the hello string to the server.

This part receives the string from the Server and stores in a buffer.

Buffer is printed.

Sample Output is given:

```
Client : Hello from client
Hello message sent.
```

This is the output of the UDPServer.c
Says it received a message from the client and a message is sent.

```
Hello message sent.
Server : Hello from server
```

This is the output of the UDPClient.c
Says it sent a message and received a message from the server.