Network Application & Development

IT – 41

Assignment - II

: Khushal Kapoor

:1404

: MSc. Informatics

Q1. Write a server program using TCP protocol which returns Client IP address a Port number.

Syntax:

Server-Side

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<sys/un.h>
#include<string.h>
#include<netdb.h>
#include<netinet/in.h>
#include<arpa/inet.h>
#include<string.h>
int main()
{
// Two buffers for message communication
      char buffer1[256], buffer2[256];
      int server = socket(AF_INET, SOCK_STREAM, 0);
      if (server < 0)
            printf("Error in server creating\n");
      else
            printf("Server Created\n");
      struct sockaddr_in my_addr, peer_addr;
      my_addr.sin_family = AF_INET;
      my_addr.sin_addr.s_addr = INADDR_ANY;
// This ip address will change according to the machine
      my_addr.sin_addr.s_addr = inet_addr("127.0.0.1");
      my_addr.sin_port = htons(4500);
      if (bind(server, (struct sockaddr*) &my_addr, sizeof(my_addr))
== 0)
            printf("Binded Correctly\n");
      else
            printf("Unable to bind\n");
      if (listen(server, 3) == 0)
            printf("Listening ...\n");
      else
            printf("Unable to listen\n");socklen_t addr_size;
      addr_size = sizeof(struct sockaddr_in);
// Ip character array will store the ip address of client
      char *ip;
// while loop is iterated infinitely to
// accept infinite connection one by one
      while (1)
      {
            int acc = accept(server, (struct sockaddr*) &peer_addr,
&addr_size);
            printf("Connection Established\n");
            char ip[INET_ADDRSTRLEN];
            inet_ntop(AF_INET, &(peer_addr.sin_addr), ip,
INET_ADDRSTRLEN);
// "ntohs(peer_addr.sin_port)" function is
// for finding port number of client
```

```
            printf("connection established with IP : %s and PORT :
%d\n",
            ip, ntohs(peer_addr.sin_port));
            recv(acc, buffer2, 256, 0);
            printf("Client : %s\n", buffer2);
            strcpy(buffer1, "Hello");
            send(acc, buffer1, 256, 0);
      }
      return 0;
}
```

Client-Side

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<sys/un.h>
#include<string.h>
#include<netdb.h>
#include<netinet/in.h>
#include<arpa/inet.h>
#include<stdlib.h>
int main()
{// Two buffer are for message communication
      char buffer1[256], buffer2[256];
      struct sockaddr_in my_addr, my_addr1;
      int client = socket(AF_INET, SOCK_STREAM, 0);
      if (client < 0)
            printf("Error in client creating\n");
      else
      printf("Client Created\n");
      my_addr.sin_family = AF_INET;
      my_addr.sin_addr.s_addr = INADDR_ANY;
      my_addr.sin_port = htons(12000);
// This ip address will change according to the machine
      my_addr.sin_addr.s_addr = inet_addr("127.0.0.1");
// Explicitly assigning port number 12010 by
// binding client with that port
      my_addr1.sin_family = AF_INET;
      my_addr1.sin_addr.s_addr = INADDR_ANY;
      my_addr1.sin_port = htons(4500);
// This ip address will change according to the machine
      my_addr1.sin_addr.s_addr = inet_addr("10.32.40.213");
      if (bind(client, (struct sockaddr*) &my_addr1, sizeof(struct
sockaddr_in)) == 0)
            printf("Binded Correctly\n");
      else
            printf("Unable to bind\n");
      socklen_t addr_size = sizeof my_addr;
      int con = connect(client, (struct sockaddr*) &my_addr, sizeof
my_addr);
      if (con == 0)
            printf("Client Connected\n");
```

```
        else
                printf("Error in Connection\n");
        strcpy(buffer2, "Hello");
        send(client, buffer2, 256, 0);
        recv(client, buffer1, 256, 0);
        printf("Server : %s\n", buffer1);
        return 0;
}
```

Output :

Server-Side:

```
Server Created
Binded Correctly
Listening ...
```

Client-Side:

```
Client Created
Unable to bind
Error in Connection
```

Q2. Discuss the generic socket address structure, IPv4 socket Address structure, IPv6 socket address structure.

The name of socket address structures begin with sockaddr_ and end with a unique suffix for each protocol suite.

IPv4 Socket Address Structure

An IPv4 socket address structure, commonly called an "Internet socket address structure", is named sockaddr_in and is defined by including the <netinet/in.h> header.

struct in_addr {
in_addr_t s_addr; /* 32-bit IPv4 address */
/* network byte ordered */ }; struct sockaddr_in {

uint8_t sin_len; /* length of structure (16) */ sa_family_t sin_family; /* AF_INET */ In_port_t
sin_port; /* 16-bit TCP or UDP port number */
/* network byte ordered */ struct in_addr sin_addr; /* 32-bit IPv4 address */
/* network byte ordered */ char sin_zero[8]; /* unused */ };

●**sin_len:** the length field. We need never set it and need never examine it.
○ The four socket functions that pass a socket address structure from the process to the kernel, bind, connect, sendto, and sendmsg, all go through the sockargs function in a Berkeley-derived implementation. This function copies the socket address structure from the process and explicitly sets its sin_len member to the size of the structure that was passed as an argument to these four functions. The five socket functions that pass a socket address structure from the kernel to the process, accept, recvfrom, recvmsg, getpeername, and getsockname, all set the sin_len member before returning to the process.

●POSIX requires only three members in the structure: sin_family, sin_addr, and sin_port. Almost all implementations add the sin_zero member so that all socket address structures are at least 16 bytes in size.

●The in_addr_t datatype must be an unsigned integer type of at least 32 bits, in_port_t must be an unsigned integer type of at least 16 bits, and sa_family_t can be any unsigned integer type. The latter is normally an 8-bit unsigned integer if the implementation supports the length field, or an unsigned 16-bit integer if the length field is not supported.

●Both the IPv4 address and the TCP or UDP port number are always stored in the structure in network byte order.

●The sin_zero member is unused. By convention, we always set the entire structure to 0 before filling it in.

●Socket address structures are used only on a given host: The structure itself is not Communicated between different hosts.

IPv6 Socket Address Structure

The IPv6 socket address is defined by including the <netinet/in.h> header:

struct in6_addr {
uint8_t s6_addr[16]; /* 128-bit IPv6 address */
/* network byte ordered */ }; #define SIN6_LEN /* required for compile-time tests */
struct sockaddr_in6 {
uint8_t sin6_len; /* length of this struct (28) */ sa_family_t sin6_family; /* AF_INET6 */ in_port_t
sin6_port; /* transport layer port# */
/* network byte ordered */ uint32_t sin6_flowinfo; /*flow information, undefined */ struct in6_addr

sin6_addr; /* IPv6 address */
/*network byte ordered */ uint32_t sin6_scope_id; /*set of interfaces for a scope */ };

●The SIN6_LEN constant must be defined if the system supports the length member for socket address structures.

●The IPv6 family is AF_INET6, whereas the IPv4 family is AF_INET.

●The members in this structure are ordered so that if the sockaddr_in6 structure is 64-bit aligned, so is the 128-bit sin6_addr member.

●The sin6_flowinfo member is divided into two fields:
○The low-order 20 bits are the flow label
○The high-order 12 bits are reserved.

●The sin6_scope_id identifies the scope zone in which a scoped address is meaningful, most commonly an interface index for a link-local address.

Generic Socket Address Structure

A socket address structures is always passed by reference when passed as an argument to any socket functions. But any socket function that takes one of these pointers as an argument must deal with socket address structures from any of the supported protocol families.
A generic socket address structure in the <sys/socket.h> header:

struct sockaddr {
uint8_t sa_len; sa_family_t sa_family; /* address family: AF_xxx value */ char sa_data[14]; /
protocol-specific address */ };

The socket functions are then defined as taking a pointer to the generic socket address structure, as shown here in the ANSI C function prototype for the bind function:

int bind(int, struct sockaddr *, socklen_t);

This requires that any calls to these functions must cast the pointer to the protocol-specific socket address structure to be a pointer to a generic socket address structure. For example:

struct sockaddr_in serv; /* IPv4 socket address structure */
/* fill in serv{} */
bind(sockfd, (struct sockaddr *) &serv, sizeof(serv));

New Generic Socket Address Structure

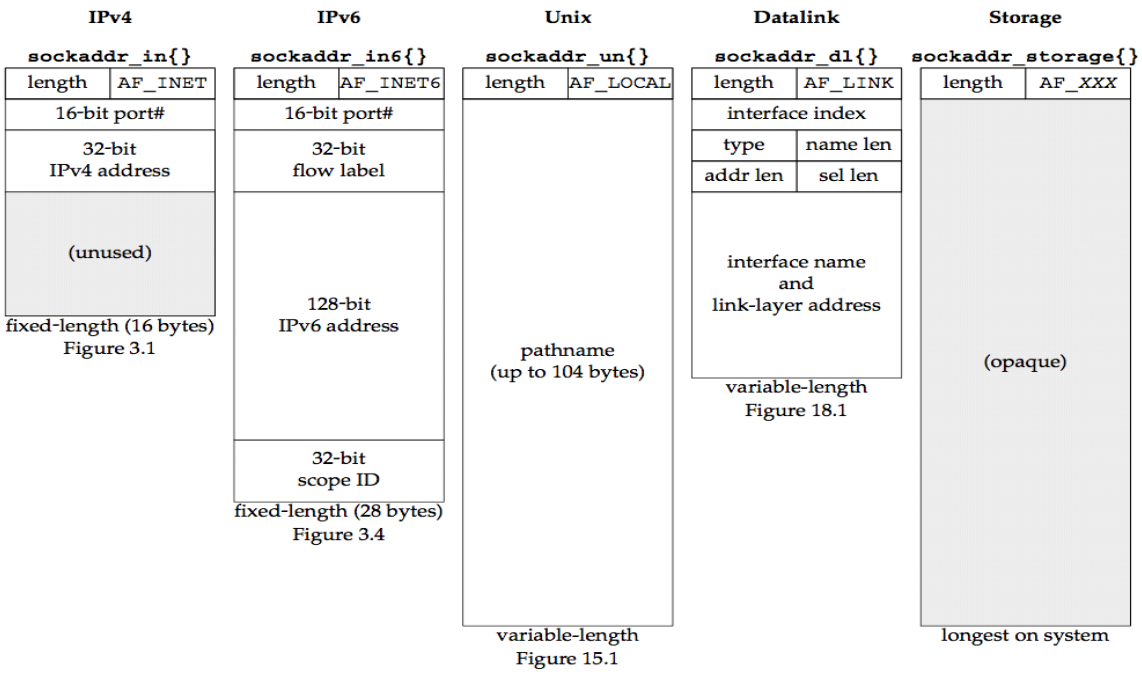
A new generic socket address structure was defined as part of the IPv6 sockets API, to overcome some of the shortcomings of the existing struct sockaddr. Unlike the struct sockaddr, the new struct sockaddr_storage is large enough to hold any socket address type supported by the system. The sockaddr_storage structure is defined by including the <netinet/in.h> header:

struct sockaddr_storage {
uint8_t ss_len; /*length of this struct(implementation dependent)*/ sa_family_t ss_family; /* address
family: AF_xxx value */
/* implementation-dependent elements to provide:
**** a) alignment sufficient to fulfill the alignment requirements of * all socket address types that the***
system supports. * b) enough storage to hold any type of socket address that the * system supports. */ };

The sockaddr_storage type provides a generic socket address structure that is different from struct sockaddr in two ways:

1. If any socket address structures that the system supports have alignment requirements, the sockaddr_storage provides the strictest alignment requirement.

2. The sockaddr_storage is large enough to contain any socket address structure that the system supports. The fields of the sockaddr_storage structure are opaque to the user, except for ss_family and ss_len (if present). The sockaddr_storage must be cast or copied to the appropriate socket address structure for the address given in ss_family to access any other fields.



Q3. Write a “C” program for TCP echo server.

Syntax:

Server-Side:

```
#include<stdlib.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<unistd.h>
#include<stdio.h>
#include<netinet/in.h>
#include<netdb.h>
#define SERV_TCP_PORT 5000
int main(int argc,char**argv)
{
      int sockfd,newsockfd,clength;
      struct sockaddr_in serv_addr,cli_addr;
      char buffer[4096];
      sockfd=socket(AF_INET,SOCK_STREAM,0);
      if(sockfd<0)
            { printf("socket creation failed\n"); }
      printf("Socket created
successfully\n");serv_addr.sin_family=AF_INET;
      serv_addr.sin_addr.s_addr=INADDR_ANY;
      serv_addr.sin_port=htons(SERV_TCP_PORT);
      bind(sockfd,(struct sockaddr*)&serv_addr,sizeof(serv_addr));
      printf("\nListening...");
      printf("\n");
      listen(sockfd,5);
      clength=sizeof(cli_addr);
      newsockfd=accept(sockfd,(struct sockaddr*)&cli_addr,&clength);
      printf("\nAccepted");
      printf("\n");
      while(1)
      {
            read(newsockfd,buffer,4096);
            printf("\nClient message:%s",buffer);
            write(newsockfd,buffer,4096);
            printf("\n");
      }
      close(sockfd);
      return 0;
}
```

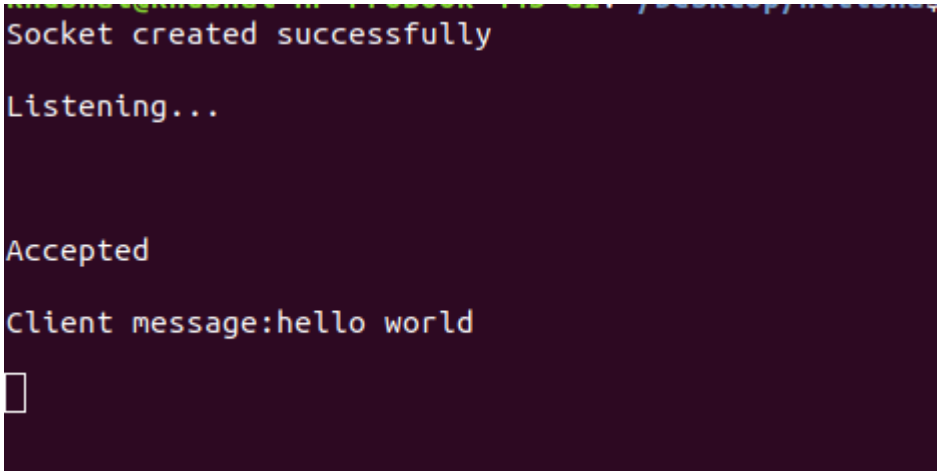
Client-Side:

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<arpa/inet.h>
```

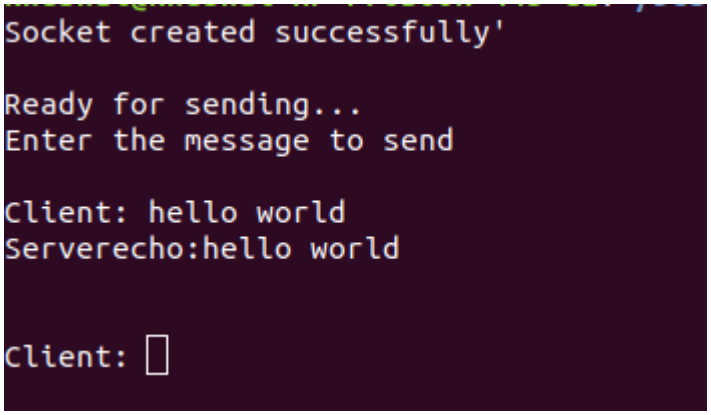
```
#include<netdb.h>
#define SERV_TCP_PORT 5000
int main(int argc,char*argv[])
{
      int sockfd;
      struct sockaddr_in serv_addr;
      struct hostent *server;
      char buffer[4096];
      sockfd=socket(AF_INET,SOCK_STREAM,0);
      if(sockfd<0)
            { printf("Socket creation failed\n"); }
      printf("Socket created successfully'\n");
      serv_addr.sin_family=AF_INET;
      serv_addr.sin_addr.s_addr = inet_addr("127.0.0.1");
      serv_addr.sin_port=htons(SERV_TCP_PORT);
      printf("\nReady for sending...");
      connect(sockfd,(struct
sockaddr*)&serv_addr,sizeof(serv_addr));
      printf("\nEnter the message to send\n");
      while(1)
      {
            printf("\nClient: ");
            fgets(buffer,4096,stdin);
            write(sockfd,buffer,4096);
            printf("Serverecho:%s",buffer);
            printf("\n");
      }
      close(sockfd);
      return 0;
}
```

Output:

Server-Side



Client-Side



Q4. Write a “C” program for TCP server to reverse string received from client.

Syntax:
Server-Side:

```
#include <sys/types.h>
#include <netinet/in.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <unistd.h>
#define PORT 22000
int main()
{
      int server_fd, new_socket, valread;
      struct sockaddr_in servaddr;
      char str[100];
      int addrlen = sizeof(servaddr);
      char buffer[1024] = { 0 };
      if ((server_fd = socket(AF_INET,SOCK_STREAM, 0)) == 0) {
            perror("socket failed");
            exit(EXIT_FAILURE);
      }
      printf("Socket created successfully.\n");
      servaddr.sin_family = AF_INET;
      servaddr.sin_addr.s_addr = INADDR_ANY;
      servaddr.sin_port = htons(PORT);
      if (bind(server_fd, (struct
sockaddr*)&servaddr,sizeof(servaddr)) < 0) {
            perror("bind failed");
```

```
            exit(EXIT_FAILURE);
      }
      if (listen(server_fd, 3) < 0) {
            perror("listen");
            exit(EXIT_FAILURE);
      }
      printf("Listening...\n");
      if ((new_socket = accept(server_fd,(struct
sockaddr*)&servaddr, (socklen_t*)&addrlen)) <0) {
            perror("accept");
            exit(EXIT_FAILURE);
      }
      valread = read(new_socket, str, sizeof(str));
      int i, j, temp;
      int l = strlen(str);
      printf("\nString sent by client:%s\n", str);
      // loop to reverse the string
      for (i = 0, j = l - 1; i < j; i++, j--) {
            temp = str[i];
            str[i] = str[j];
            str[j] = temp;
      }// send reversed string to client
      // by send system call
      send(new_socket, str, sizeof(str), 0);
      printf("\nModified string sent to client\n");
      return 0;
}
```

Client-Side:

```
#include <sys/types.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <unistd.h>
#define PORT 22000
int main()
{
      struct sockaddr_in address;
      int sock = 0, valread;
      struct sockaddr_in serv_addr;
      char str[100];
      printf("\nInput the string:");
      scanf("%[^\n]s", str);
      char buffer[1024] = { 0 };
      if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
            printf("\n Socket creation error \n");
            return -1;
      }
      memset(&serv_addr, '0', sizeof(serv_addr));
      serv_addr.sin_family = AF_INET;
      serv_addr.sin_port = htons(PORT);
```

```
	if (inet_pton(AF_INET, "127.0.0.1", &serv_addr.sin_addr) <= 0)
{
		printf("\nAddress not supported \n");
		return -1;
	}
	if (connect(sock, (struct sockaddr*)&serv_addr,
sizeof(serv_addr)) < 0) {
		printf("\nConnection Failed \n");
		return -1;
	}
	int l = strlen(str);
	// send string to server side
	send(sock, str, sizeof(str), 0);
	// read string sent by server
	valread = read(sock, str, l);
	printf("%s\n", str);
	return 0;
}
```

Output:

Server-Side

```
Socket created successfully.
Listening...

String sent by client:hello world

Modified string sent to client
```

Client-Side

```
Input the string:hello world
dlrow olleh
```

Q5. Write a "C" program to implement TCP echo server using select().

Syntax:
Server-Side:

```
#include <arpa/inet.h>
#include <errno.h>
#include <netinet/in.h>
#include <signal.h>
```

```
#include <stdio.h>
#include <stdlib.h>
#include <strings.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <unistd.h>
#define PORT 5000
#define MAXLINE 1024
int max(int x, int y)
{
      if (x > y)
            return x;
      else
      return y;
}

int main()
{
      int listenfd, connfd, udpfd, nready, maxfdp1;
      char buffer[MAXLINE];
      pid_t childpid;fd_set rset;
      ssize_t n;
      socklen_t len;
      const int on = 1;
      struct sockaddr_in cliaddr, servaddr;
      char *message = "Hello Client.";
      void sig_chld(int);
      listenfd = socket(AF_INET, SOCK_STREAM, 0);
      if(listenfd<0)
            { printf("error in socket creation.\n"); }
      printf("TCP Socket created successfully.\n");
      bzero(&servaddr, sizeof(servaddr));
      servaddr.sin_family = AF_INET;
      servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
      servaddr.sin_port = htons(PORT);
      // binding server
      bind(listenfd, (struct sockaddr*)&servaddr, sizeof(servaddr));
      listen(listenfd, 5);
      printf("Listening...\n");
      udpfd = socket(AF_INET, SOCK_DGRAM, 0);
      if(udpfd<0)
            { printf("error in socket creation.\n"); }
      printf("UDP Socket created successfully.\n");
      // binding server
      bind(udpfd, (struct sockaddr*)&servaddr, sizeof(servaddr));
      // clear the descriptor set
      FD_ZERO(&rset);
      // get maxfd
      maxfdp1 = max(listenfd, udpfd) + 1;
      for (;;) {
      // set listenfd and udpfd in readset
            FD_SET(listenfd, &rset);
            FD_SET(udpfd, &rset);
      // select the ready descriptor
            nready = select(maxfdp1, &rset, NULL, NULL, NULL);// if
tcp socket is readable then handle
```

```
	// it by accepting the connection
		if (FD_ISSET(listenfd, &rset)) {
			len = sizeof(cliaddr);
			connfd = accept(listenfd, (struct
sockaddr*)&cliaddr, &len);
			if ((childpid = fork()) == 0) {
				close(listenfd);
				read(connfd,buffer,1024);
				printf("\nClient message:%s",buffer);
				write(connfd,buffer,1024);
			}
			close(connfd);
		}
	// if udp socket is readable receive the message.
		if (FD_ISSET(udpfd, &rset)) {
			len = sizeof(cliaddr);
			bzero(buffer, sizeof(buffer));
			printf("\nMessage from UDP client: ");
			n = recvfrom(udpfd, buffer, sizeof(buffer), 0,
(struct sockaddr*)&cliaddr, &len);
			puts(buffer);
			sendto(udpfd, (const char*)message, sizeof(buffer),
0, (struct sockaddr*)&cliaddr,
			sizeof(cliaddr));
		}
	}
}
```

Client-Side:

```
#include <netinet/in.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include<unistd.h>
#include<arpa/inet.h>
#define PORT 5000
#define MAXLINE 1024
int main()
{
	int sockfd;
	char buffer[MAXLINE];struct sockaddr_in servaddr;
	int n, len;
	if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
		printf("socket creation failed");
		exit(0);
	}
	printf("Socket creation successful\n");
	memset(&servaddr, 0, sizeof(servaddr));
	servaddr.sin_family = AF_INET;
	servaddr.sin_port = htons(PORT);
	servaddr.sin_addr.s_addr = inet_addr("127.0.0.1");
```

```
	if (connect(sockfd, (struct
sockaddr*)&servaddr,sizeof(servaddr)) < 0) {
		printf("\n Error : Connect Failed \n");
	}
	printf("\nWrite here: ");
	fgets(buffer,4096,stdin);
	write(sockfd,buffer,4096);
	printf("\nServerecho:%s",buffer);
	printf("\n");
	close(sockfd);
}
```

UDP-Client:

```
#include <arpa/inet.h>
#include <netinet/in.h>
#include <stdio.h>
#include <stdlib.h>
#include<unistd.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#define PORT 5000
#define MAXLINE 1024
int main()
{
	int sockfd;char buffer[MAXLINE];
	char* message = "Hello Server";
	struct sockaddr_in servaddr;
	int n, len;
	if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0) {
		printf("socket creation failed");
		exit(0);
	}
	printf("Socket creation successful.\n");
	memset(&servaddr, 0, sizeof(servaddr));
	servaddr.sin_family = AF_INET;
	servaddr.sin_port = htons(PORT);
	servaddr.sin_addr.s_addr = inet_addr("127.0.0.1");
	// send hello message to server
	sendto(sockfd, (const char*)message, strlen(message),0, (const
struct sockaddr*)&servaddr,
	sizeof(servaddr));
	// receive server's response
	printf("Message from server: ");
	n = recvfrom(sockfd, (char*)buffer, MAXLINE,0, (struct
sockaddr*)&servaddr, &len);
	puts(buffer);
	close(sockfd);
	return 0;
}
```

Output:

Server-Side

```
TCP Socket created successfully.
Listening...
UDP Socket created successfully.

Client message:hello world


Client message:hello world


Message from UDP client: Hello Server
```

Client-Side

```
Socket creation successful

Write here: hello world

Serverecho:hello world
```

UDP-Client

```
Socket creation successful.
Message from server: Hello Client.
```