Program:

Server Side

```
// server code for UDP socket programming

#include <arpa/inet.h>

#include <netinet/in.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/socket.h>

#include <sys/types.h>

#include <unistd.h>


#define IP_PROTOCOL 0

#define PORT_NO 15050

#define NET_BUF_SIZE 32

#define cipherKey 'S'

#define sendrecvflag 0

#define nofile "File Not Found!"


// function to clear buffer

void clearBuf(char* b)

{

        int i;

        for (i = 0; i < NET_BUF_SIZE; i++)

                b[i] = '\0';

}


// function to encrypt

char Cipher(char ch)

{

        return ch ^ cipherKey;
```

```
}

// function sending file
int sendFile(FILE* fp, char* buf, int s)
{
        int i, len;
        if (fp == NULL) {
                strcpy(buf, nofile);
                len = strlen(nofile);
                buf[len] = EOF;
                for (i = 0; i <= len; i++)
                        buf[i] = Cipher(buf[i]);
                return 1;
        }

        char ch, ch2;
        for (i = 0; i < s; i++) {
                ch = fgetc(fp);
                ch2 = Cipher(ch);
                buf[i] = ch2;
                if (ch == EOF)
                        return 1;
        }
        return 0;
}

// driver code
int main()
{
        int sockfd, nBytes;
        struct sockaddr_in addr_con;
```

```
int addrlen = sizeof(addr_con);
addr_con.sin_family = AF_INET;
addr_con.sin_port = htons(PORT_NO);
addr_con.sin_addr.s_addr = INADDR_ANY;
char net_buf[NET_BUF_SIZE];
FILE* fp;

// socket()
sockfd = socket(AF_INET, SOCK_DGRAM, IP_PROTOCOL);

if (sockfd < 0)
        printf("\nfile descriptor not received!!\n");
else
        printf("\nfile descriptor %d received\n", sockfd);

// bind()
if (bind(sockfd, (struct sockaddr*)&addr_con, sizeof(addr_con)) == 0)
        printf("\nSuccessfully binded!\n");
else
        printf("\nBinding Failed!\n");

while (1) {
        printf("\nWaiting for file name...\n");

        // receive file name
        clearBuf(net_buf);

        nBytes = recvfrom(sockfd, net_buf,
                                        NET_BUF_SIZE, sendrecvflag,
                                        (struct sockaddr*)&addr_con, &addrlen);
```

```
                fp = fopen(net_buf, "r");
                printf("\nFile Name Received: %s\n", net_buf);
                if (fp == NULL)
                        printf("\nFile open failed!\n");
                else
                        printf("\nFile Successfully opened!\n");

                while (1) {

                        // process
                        if (sendFile(fp, net_buf, NET_BUF_SIZE)) {
                                sendto(sockfd, net_buf, NET_BUF_SIZE,
                                        sendrecvflag,
                                        (struct sockaddr*)&addr_con, addrlen);
                                break;
                        }

                        // send
                        sendto(sockfd, net_buf, NET_BUF_SIZE,
                                sendrecvflag,
                                (struct sockaddr*)&addr_con, addrlen);
                        clearBuf(net_buf);
                }
                if (fp != NULL)
                        fclose(fp);
        }
        return 0;
}
```

Client Side

```
// client code for UDP socket programming
#include <arpa/inet.h>
```

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

#define IP_PROTOCOL 0
#define IP_ADDRESS "127.0.0.1" // localhost
#define PORT_NO 15050
#define NET_BUF_SIZE 32
#define cipherKey 'S'
#define sendrecvflag 0

// function to clear buffer
void clearBuf(char* b)
{
	int i;
	for (i = 0; i < NET_BUF_SIZE; i++)
		b[i] = '\0';
}

// function for decryption
char Cipher(char ch)
{
	return ch ^ cipherKey;
}

// function to receive file
int recvFile(char* buf, int s)
```

```
{
	int i;
	char ch;
	for (i = 0; i < s; i++) {
		ch = buf[i];
		ch = Cipher(ch);
		if (ch == EOF)
			return 1;
		else
			printf("%c", ch);
	}
	return 0;
}

// driver code
int main()
{
	int sockfd, nBytes;
	struct sockaddr_in addr_con;
	int addrlen = sizeof(addr_con);
	addr_con.sin_family = AF_INET;
	addr_con.sin_port = htons(PORT_NO);
	addr_con.sin_addr.s_addr = inet_addr(IP_ADDRESS);
	char net_buf[NET_BUF_SIZE];
	FILE* fp;

	// socket()
	sockfd = socket(AF_INET, SOCK_DGRAM,
					IP_PROTOCOL);

	if (sockfd < 0)
```

```
		printf("\nfile descriptor not received!!\n");
	else
		printf("\nfile descriptor %d received\n", sockfd);

	while (1) {
		printf("\nPlease enter file name to receive:\n");
		scanf("%s", net_buf);
		sendto(sockfd, net_buf, NET_BUF_SIZE,
			sendrecvflag, (struct sockaddr*)&addr_con,
			addrlen);

		printf("\n---------Data Received---------\n");

		while (1) {
			// receive
			clearBuf(net_buf);
			nBytes = recvfrom(sockfd, net_buf, NET_BUF_SIZE,
							sendrecvflag, (struct sockaddr*)&addr_con,
							&addrlen);

			// process
			if (recvFile(net_buf, NET_BUF_SIZE)) {
				break;
			}
		}
		printf("\n-------------------------------\n");
	}
	return 0;
}
```

****

Output:

Server Output

```
file descriptor 3 received


Successfully binded!


Waiting for file name...


File Name Received: dhruvil.txt


File Successfully opened!
```

Client Output

```
file descriptor 3 received


Please enter file name to receive:
dhruvil.txt


---------Data Received---------


------------------------------
```

****

Dhruvil Shah 047

F18111051 TE Comp 1

Assignment 6

Q1 What is socket ? Explain system calls related to UDP socket

Ans Socket is one end point of a two way communication link between two programs running on the network. The socket mechanism provides means of interprocess communication (IPC) established by named contact points between which the communication takes place.

System calls related to UDP socket:

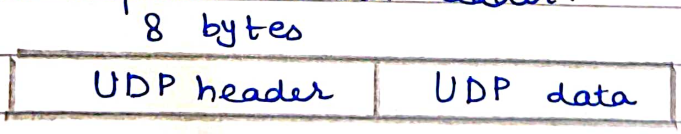
1] int socket (int domain, int type, int protocol);
Creates an unbound socket in the specified domain. Returns socket file descriptor.

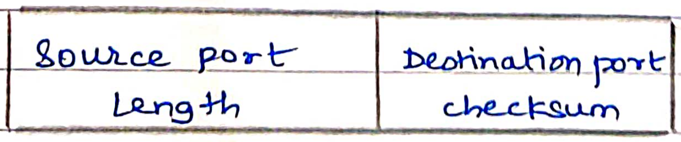
2] int bind (int socket, const struct address *addr, socketlen_t addrlen);
Assign address to the unbound socket.

3] ssize_t sendto (int sockfd, const void *buf, size const struct sockaddr *dest_addr);
Send a message on the socket.

4] ssize_t recvfrom (int sockfd, void *buf, size_t l struct sockaddr *src_addr, sock)
Receive message from the socket.

Q2 Draw and explain the UDP header.





→ all files are of 16 bits

1] Source Port : Source port is 2 bytes . Identifies source port number.

2] Destination Port : It's 2 bytes long and identified destination port number

3] Length : It's UDP length including header and data.

4] Checksum : It's a 2 byte field containing 16 bits 1's complement of the 1s complement checksum of UDP header : psuedo header of information from the IP header and the data, padded with 0 octates at the end if necessary to make a multiple of 2 octets. i.e.

Q3 Explain the FTP protocol.

Ans File transfer protocol is a standard internet protocol provided by TCP | IP . Mainly used for transmitting the web pages , files from their creator to the server and also used for downloading files to computer from server. It is used to encourage the use of remote computers. It transfers the data more reliably and efficiently.

Q4 Write down the Steps involved in establishing UDP socket on the client side and server side.

Ans **Server Side:**

1] Create UDP socket
2] Bind the socket to the server address
3] Wait until datagram packet arrive from client
4] Process the datagram packet and send a reply to client
5] Go Back to step 3 (waiting)

Client side:

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