3. Simple TCP/IP Client Server Communication

Server

```
import socket
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
print ("Socket created successfully!”)
port = 3431
s.bind(('', port))
print("Socket binded to %s " %(port)) s.listen(5)
print("Socket is listening")
while True:
c, addr = s.accept()
print("Got connection from ", addr)
c.send("Thank You for connecting :))”.encode())
c.close()
```

Client

```
import socket
s = socket.socket()
port = 3431
s.connect(('127.0.0.1', port))
print(s.recv(1024).decode())
s.close()
```

4. UDP ECHO CLIENT SERVER COMMUNICATION

Server

```
import socket
localIP     = "127.0.0.1"
localPort   = 30001
bufferSize  = 1024
msgFromServer       = "Hello UDP Client"
bytesToSend         = str.encode(msgFromServer)
UDPServerSocket = socket.socket(family=socket.AF_INET, type=socket.SOCK_DGRAM)
UDPServerSocket.bind((localIP, localPort))
print("UDP server up and listening")
while(True):
    bytesAddressPair = UDPServerSocket.recvfrom(bufferSize)
    message = bytesAddressPair[0]
    address = bytesAddressPair[1]
    clientMsg = "client says hi:{}".format(message)
    clientIP  = "Client IP Address:{}".format(address)
    print(clientMsg)
    print(clientIP)
    UDPServerSocket.sendto(bytesToSend, address)
```

Client

```
import socket
msgFromClient = "Hello UDP Server"
bytesToSend = str.encode(msgFromClient)
serverAddressPort = ("127.0.0.1", 30001)
bufferSize = 1024
UDPClientSocket = socket.socket(family=socket.AF_INET, type=socket.SOCK_DGRAM)
```

```
UDPClientSocket.sendto(bytesToSend, serverAddressPort)
msgFromServer = UDPClientSocket.recvfrom(bufferSize)
msg = "Message from Server {}".format(msgFromServer[0])
print(msg)
```

5. CONCURRENT TCP/IP DAY-TIME SERVER

Server

```
import socket
import time
serversocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
host = socket.gethostname()
port = 9996
serversocket.bind((host, port))
serversocket.listen(5)
while True:
    clientsocket,addr = serversocket.accept()
    print("Got a connection from %s" % str(addr))
    currentTime = time.ctime(time.time()) + "\r\n"
    clientsocket.send(currentTime.encode('ascii'))
    clientsocket.close()
```

Client

```
import socket
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
host = socket.gethostname()
port = 9996
s.connect((host, port))
tm = s.recv(1024)
s.close()
print("The time got from the server is %s" % tm.decode('ascii'))
```

6. HALF DUPLEX CHAT USING TCP/IP

Server

```
from socket import *
server_port = 5005
server_socket = socket(AF_INET,SOCK_STREAM)
server_socket.bind(('',server_port))
server_socket.listen(1)
print ("Welcome: The server is now ready to receive")
connection_socket, address = server_socket.accept()
while True:
  sentence = connection_socket.recv(2048).decode()
  print('>> ',sentence)
```

```
  message = input(">> ")
  connection_socket.send(message.encode())
  if(message == 'q'):
    connectionSocket.close()
```

Client

```
from socket import *
server_name = 'localhost'
server_port = 5005
client_socket = socket(AF_INET, SOCK_STREAM)
client_socket.connect((server_name,server_port))
while True:
  sentence = input(">> ")
  client_socket.send(sentence.encode())
  message = client_socket.recv(2048)
  print (">> ", message.decode())
  if(sentence == 'q'):
    client_socket.close()
```

7. FULL DUPLEX CHAT USING TCP/IP

Server

```
from socket import *
HOST = ''
PORT = 8007
BUFSIZ = 1024
ADDR = (HOST, PORT)
tcpSerSock = socket(AF_INET, SOCK_STREAM)
tcpSerSock.bind(ADDR)
tcpSerSock.listen(5)
# Just setting up the server upto now
print("Starting the server")
while True:
        tcpCliSock, addr = tcpSerSock.accept()      # passing on the client handler to a new socket
        print("Connected to ", addr)
        while True:
                data = tcpCliSock.recv(BUFSIZ)
                if not data:                    # Go to next client if this one send empty
                        break
                while data.decode().strip().lower() != 'continue':    # Receiving data loop
                        print(data.decode().strip())
                        data = tcpCliSock.recv(BUFSIZ)
        #client sends continue once its sending data loop is done
                message = input(">")
                while message.lower() != 'go' and message != '':       # Sending data loop
                        message += '\r\n'
                        tcpCliSock.send(message.encode())
                        message = input(">")
        # go is the word to stop the sending loop
                else:
                        if message != '':
                                tcpCliSock.send('continue\r\n'.encode())
                                break
                        else:
                                break
         tcpCliSock.close()
```

Client

```
from socket import *
HOST = 'localhost'              #Just for testing, can change it later
PORT = 8007
BUFSIZ = 1024
ADDR = (HOST, PORT)

tcpCliSock = socket(AF_INET, SOCK_STREAM)
tcpCliSock.connect(ADDR)
while True:
   message = input(">")           #The sending data loop for the client, 'go' stops the loop
   if message == '':
      break
   while message.lower() != 'go':
      message += '\r\n'
      tcpCliSock.send(message.encode())
      message = input(">")
      if message == '':
         break
      else:                        # Receiving data loop, stops when empty symbol received.
         message = 'continue\r\n'
         tcpCliSock.send(message.encode())
      data = tcpCliSock.recv(BUFSIZ)
      data = data.decode().strip()
      if not data:
         break
      while data.lower() != 'continue':       # sent by the server when it is done typing.
         print(data)
         data = tcpCliSock.recv(BUFSIZ)
         data = data.decode().strip()
tcpCliSock.close()
```

8. IMPLEMENTATION OF FILE TRANSFER PROTOCOL

Server

```
import socket port = 6000
s = socket.socket()
host = socket.gethostname()
s.bind((host, port)) s.listen(5)
print("Server listening...") while True:
conn, addr = s.accept()
print("Got connection from", addr) data = conn.recv(1024)
print("Server received", repr(data))
filename = "my_text.txt" f = open(filename, 'rb')
l = f.read(1024) while(l):
conn.send(l) print("Sent!", repr(l)) l = f.read(1024)
```

```
f.close() print("Successfully sent!”)
conn.send(b"Thank you for connecting :))") conn.close()
```

Client

```
import socket
s = socket.socket()
host = socket.gethostname() port = 6000
s.connect((host, port)) message = "Hello Server!" s.send(message.encode())
with open('received_file', 'wb') as f: print("File Opened")
while True:
print("Receiving data….")
data = s.recv(1024) print('data = %s', data) if not data:
break
f.write(data) f.close()
print("Successfully received the file”)
s.close()
```

9. REMOTE COMMAND EXECUTION USING UDP

Server

```
#include <sys/types.h>
#include <sys/socket.h>
#include <stdio.h>
#include <stdlib.h>
#include <netdb.h>
#include <netinet/in.h>
#include <string.h>
#include <sys/stat.h>
#include <arpa/inet.h>
#include <unistd.h>
#define MAX 1000
#include <strings.h>
void bzero(void *s, size_t n);
void explicit_bzero(void *s, size_t n);
int main() {
int serverDescriptor = socket(AF_INET, SOCK_DGRAM, 0);
int size;
char buffer[MAX], message[] = "Command Successfully executed !”;
struct sockaddr_in clientAddress, serverAddress; socklen_t clientLength = sizeof(clientAddress);
bzero(&serverAddress, sizeof(serverAddress)); serverAddress.sin_family = AF_INET;
serverAddress.sin_addr.s_addr = htonl(INADDR_ANY); serverAddress.sin_port = htons(9976);
bind(serverDescriptor, (struct sockaddr *)&serverAddress, sizeof(serverAddress));
while (1) {
        bzero(buffer, sizeof(buffer));
recvfrom(serverDescriptor, buffer, sizeof(buffer), 0, (struct sockaddr *)&clientAddress, &clientLength);
system(buffer);
printf("Command Executed ... %s ", buffer);
sendto(serverDescriptor, message, sizeof(message), 0, (struct sockaddr *)&clientAddress, clientLength);
}
    close(serverDescriptor);
    return 0;
}
```

Client

```
#include <sys/types.h>
#include <sys/socket.h>
#include <stdio.h>
#include <unistd.h>
#include <netdb.h>
#include <netinet/in.h>
#include <string.h>
#include <arpa/inet.h>
#define MAX 1000
#include <strings.h>
void bzero(void *s, size_t n);
void explicit_bzero(void *s, size_t n);
int main() {
int serverDescriptor = socket(AF_INET, SOCK_DGRAM, 0); char buffer[MAX], message[MAX];
struct sockaddr_in cliaddr, serverAddress; socklen_t serverLength = sizeof(serverAddress);
bzero(&serverAddress, sizeof(serverAddress)); serverAddress.sin_family = AF_INET;
serverAddress.sin_addr.s_addr = inet_addr("127.0.0.1"); serverAddress.sin_port = htons(9976);
bind(serverDescriptor, (struct sockaddr *)&serverAddress, sizeof(serverAddress));
while (1) {
printf("\nCOMMAND FOR EXECUTION ... ");
fgets(buffer, sizeof(buffer), stdin);
sendto(serverDescriptor, buffer, sizeof(buffer), 0, (struct sockaddr
*)&serverAddress, serverLength);
        printf("\nData Sent !”);
recvfrom(serverDescriptor, message, sizeof(message), 0, (struct sockaddr *)&serverAddress,
&serverLength);
printf("UDP SERVER : %s", message);
}
return 0;
}
```

10. ARP IMPLEMENTATION USING UDP

```
#include<sys/types.h>
#include<sys/socket.h>
#include<net/if_arp.h>
#include<sys/ioctl.h>
#include<stdio.h>
#include<string.h>
#include<unistd.h>
#include<math.h>
#include<complex.h>
#include<arpa/inet.h>
#include<netinet/in.h>
#include<netinet/if_ether.h>
#include<net/ethernet.h>
#include<stdlib.h>
int main()
{
struct sockaddr_in sin={0};
struct arpreq myarp={{0}};
unsigned char *ptr;
int sd;
sin.sin_family=AF_INET;
printf("Enter IP address: ");
char ip[20];
scanf("%s", ip);
if(inet_pton(AF_INET,ip,&sin.sin_addr)==0)
```

```
{
printf("IP address Entered '%s' is not valid \n",ip);
exit(0);
}
memcpy(&myarp.arp_pa,&sin,sizeof(myarp.arp_pa));
strcpy(myarp.arp_dev,”echo");
sd=socket(AF_INET,SOCK_DGRAM,0);
printf("\nSend ARP request\n”);
if(ioctl(sd,SIOCGARP,&myarp)==1) {
printf("No Entry in ARP cache for '%s'\n",ip);
exit(0);
}
ptr=&myarp.arp_pa.sa_data[0];
printf("Received ARP Reply\n”);
printf("\nMAC Address for '%s' : ",ip);
printf("%p:%p:%p:%p:%p:%p\n",ptr,(ptr+1),(ptr+2),(ptr+3),(ptr+4),(ptr+5));
return 0;
}
```