SET-B
Part - A
(10 x 1 = 10 Marks)

Instructions: Answer all

Q. No	**Answer with choice variable**
1	A concurrent server can handle ___________clients and an iterative server can handle ______clients. a. **many, one** b. one, two c. one, many d. two, one
2	What is the purpose of the integer parameter in the shutdown() method? a. To specify the socket's priority level. b. **To indicate which buffers should be flushed before closing the socket.** c. To define the maximum number of concurrent connections. d. To set a timeout for the socket closure operation.
3	Beyond IP, UDP provides additional services such as _______ a. Routing and switching b. Reliable services c. Multiplexing and demultiplexing d. **Demultiplexing and error checking**
4	Port number used by Network Time Protocol (NTP) with UDP is ________ a. 161 b. **123** c. 162 d. 124
5	“Total length” field in UDP packet header is the length of _________ a. Only UDP header b. Only data c. Only checksum d. **UDP header plus data**
6	In telnet, the client echoes the character on the screen but does not send it until a whole line is completed in A. **default mode** B. character mode C. server mode D. Client mode
7	Which one of the following allows a user at one site to establish a connection to another site and then pass keystrokes from local host to remote host? a) http b) ftp c) **telnet** d) smtp
8	The Uniform Resource Locator (URL), is a standard for specifying any kind of information on the a. server-end

	b. client-end c. web page d. **internet**
9	In FTP, the port number ______ and ______is used for data and control connection. **a) 21, 20** b) 20,21 c) 20,12 d) 12,21
10	In the process of fetching a web page from a server , the HTTP request/response takes a) 2 RTT b) **1 RTT** c) 4 RTT d) 3 RTT

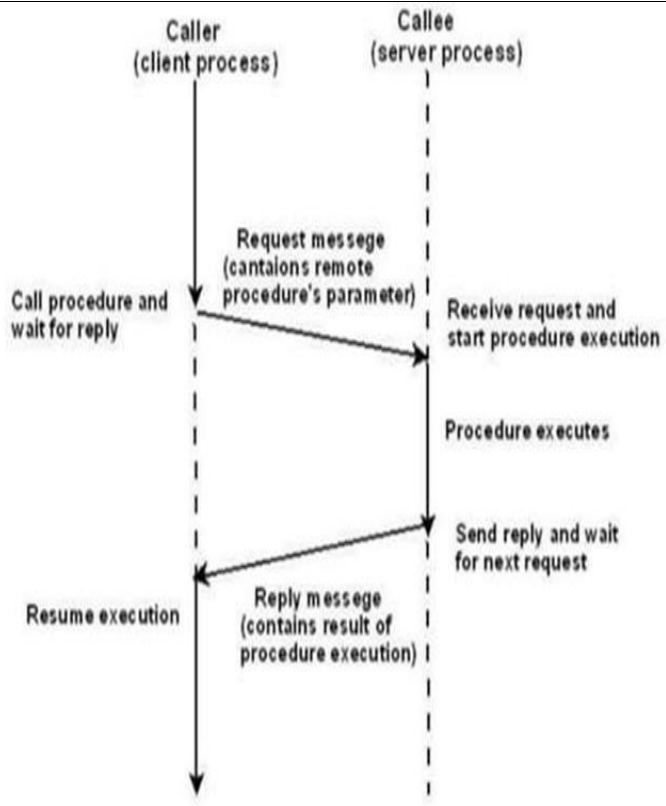
11	Write the Client, Server program for the following scenario. Rahul wants to open a text editor at his home from his office where the server system is placed. How will you help him to open the specific application without visiting his home?

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
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(8079);
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
    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(8079);
    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;
}
```

12	Explain the remote procedure call concepts with suitable diagram & sample snippets.



Remote procedure call model

●A remote procedure call is an interprocess communication technique that is used for client-server based applications. It is also known as a subroutine call or a function call.

●RPC allows programs to call the procedure which is located on the other machines.

●Message passing is not visible to the programmer , so it is called a Remote Procedure call (RPC).

●RPC enables a procedure call that does not reside in the address space of the calling process.

●In RPC, the caller and the callee has disjoint address space, hence there is no access to data and variables in the callers environment.

●RPC performs a message passing scheme for information exchange between the caller and the callee process.

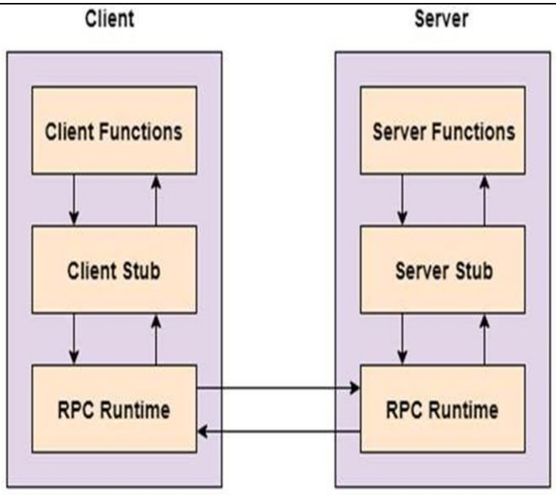
●A client has a request message that the RPC translates and sends to the server.

●This request may be a procedure or a function call to a remote server.

●When the server receives the request, it sends the required response back to the client.

●The client is blocked while the server is processing the call and only resumed execution after the server is finished.

Client Server RPC Model



Sequence of events in a RPC

●The client stub is called by the client.
●The client stub makes a system call to send the message to the server and puts the parameters in the message.

●The message is sent from the client to the server by the client's operating system.

●The message is passed to the server stub by the server operating system.

●The parameters are removed from the message by the server stub.

●Then, the server procedure is called by the server stub.

•RPC Features

●Remote procedure calls support process oriented and thread oriented models.

●The internal message passing mechanism of RPC is hidden from the user.

●The effort to re-write and re-develop the code is minimum in remote procedure calls.●

●Remote procedure calls can be used in distributed environment as well as the local environment.

●Many of the protocol layers are omitted by RPC to improve performance.

●Ease of use, efficiency.

13 A client does not want to establish a connection with the server and instantly sends a datagram. Analyse this scenario and write the client server program to perform the communication.

Server code:

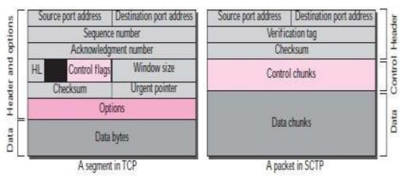
```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/socket.h>

#include <netinet/in.h>
```

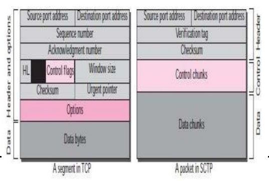
```
#define SERVER_PORT 12345

#define BUFFER_SIZE 1024


int main() {

    // Create UDP socket

    int sockfd = socket(AF_INET, SOCK_DGRAM, 0);

    if (sockfd < 0) {

        perror("socket creation failed");

        exit(EXIT_FAILURE);

    }


    // Define server address

    struct sockaddr_in server_addr;

    memset(&server_addr, 0, sizeof(server_addr));

    server_addr.sin_family = AF_INET;

    server_addr.sin_addr.s_addr = INADDR_ANY;

    server_addr.sin_port = htons(SERVER_PORT);


    // Bind socket to the server address

    if (bind(sockfd, (struct sockaddr*)&server_addr, sizeof(server_addr)) < 0) {

        perror("bind failed");

        exit(EXIT_FAILURE);

    }


    printf("Server listening on port %d\n", SERVER_PORT);


    // Receive data from the client

    while (1) {

        char buffer[BUFFER_SIZE];

        struct sockaddr_in client_addr;
        socklen_t client_addr_len = sizeof(client_addr);


        int bytes_received = recvfrom(sockfd, buffer, BUFFER_SIZE, 0, (struct
sockaddr*)&client_addr, &client_addr_len);
        if (bytes_received < 0) {
            perror("recvfrom failed");
            continue;
        }


        printf("Received data: %s\n", buffer);
    }


    close(sockfd);
    return 0;
}


client code:
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <netinet/in.h>


#define SERVER_PORT 12345
#define SERVER_ADDRESS "127.0.0.1"
#define BUFFER_SIZE 1024


int main() {
    // Create UDP socket
    int sockfd = socket(AF_INET, SOCK_DGRAM, 0);
    if (sockfd < 0) {
            perror("socket creation failed");

            exit(EXIT_FAILURE);

        }



        // Define server address

        struct sockaddr_in server_addr;

        memset(&server_addr, 0, sizeof(server_addr));

        server_addr.sin_family = AF_INET;

        server_addr.sin_port = htons(SERVER_PORT);

        inet_pton(AF_INET, SERVER_ADDRESS, &server_addr.sin_addr);



        // Send data to the server

        char message[] = "Hello, Server!";

        if (sendto(sockfd, message, strlen(message), 0, (struct sockaddr*)&server_addr,
    sizeof(server_addr)) < 0) {

            perror("sendto failed");

            exit(EXIT_FAILURE);

        }



        printf("Message sent to server: %s\n", message);



        close(sockfd);

        return 0;

    }
```

14 a. Draw and compare a TCP segment and an SCTP packet format.

b. Explain the association establishment and termination phases of SCTP with neat diagrams.

a.





TCP segment	SCTP packet
Control information is part of the header	Control information is included in the control chunks
Data is treated as one entity	Carry several data chunks, each can belong to a different stream
Options section exist separately	Options are handled by defining new chunk types
Mandatory part of header is 20 bytes	General header is only 12 bytes
Checksum is 16 bits	Checksum is 32 bits
Combination of	Verification tag is an association identifier
Includes one sequence number in the header	Includes several different data chunks
Some segments carry control information	Control chunks never use a TSN, IS, or SSN number, they are used for data chunks only

b.

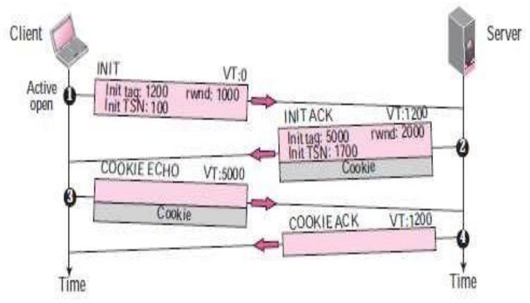
The association establishment of SCTP consists of the following

Four-way handshake

1. First packet has INIT chunk sent by client

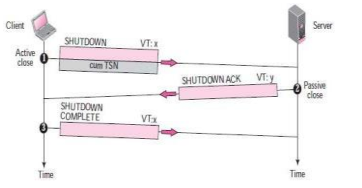
- Verification tag is 0
- Rwnd is advertised in a SACK chunk
- Inclusion of a DATA chunk in the third and fourth packets

2. Second packet has INIT ACK chunk sent by server
   - Verification tag is the initial tag field in the INIT chunk
   - Initiates the tag to be used in the other direction
   - Defines the initial TSN and sets the servers’ rwnd
3. Third packet has COOKIE ECHO chunk sent by client
   - Echoes the cookie sent by the server
   - Data chunks are included in this packet
4. Fourth packet has COOKIE ACK chunk sent by server
   - Acknowledges the receipt of the COOKIE ECHO chunk
   - Data chunks are included with this packet.



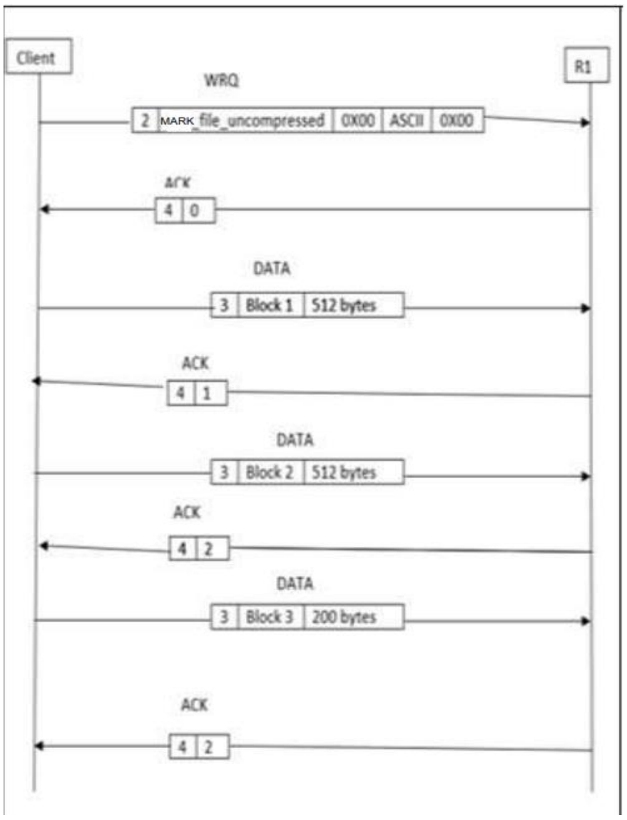
The Association termination in a SCTP is done by sending three packets between a client and a server

- SHUTDOWN
- SHUTDOWN ACK
- SHUTDOWN COMPLETE

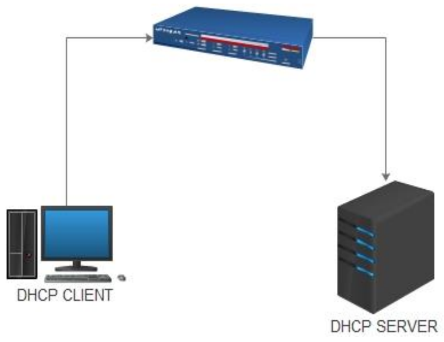


15 Mark uncompresses the file named “FILE_1” and needs to send it to server. “FILE_1” is an uncompressed file and it consumes 1224 bytes of data. Identify the suitable protocol and suggest Mark in framing message structure.

Trivial File Transfer Protocol



16 i) Can DHCP prevent unauthorized laptops from using a network that uses DHCP for dynamic addressing?
ii) Explain the communication flow between a DHCP client and server on a network with two DHCP Servers.
iii) Consider the below diagram, a DHCP client and server is connected to a switch. How does the DHCP process start?



Answer – i) No, DHCP is not capable of distinguishing between a permanent MAC address and the address by the user. So, it cannot stop unauthorized access to a network and cannot control the IP addresses used by users. (2)

ii) 4M

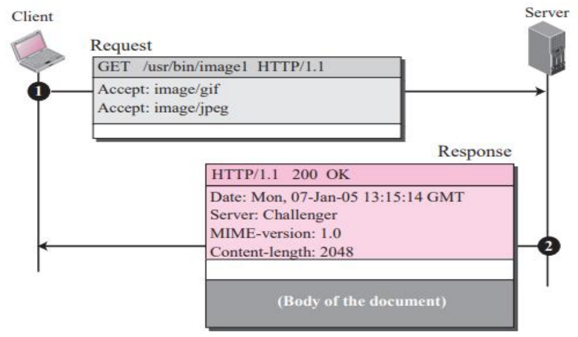
The first packet the DHCP Client initiates would be the DHCP Discover packet. The DHCP Discover packet is broadcast in nature and would be received by both the DHCP servers. The DHCP servers would respond with DHCP offer packet containing the IP addresses which they offer. Based on the first DHCP offer the client receives, the client would respond with DHCP request packet which contains the

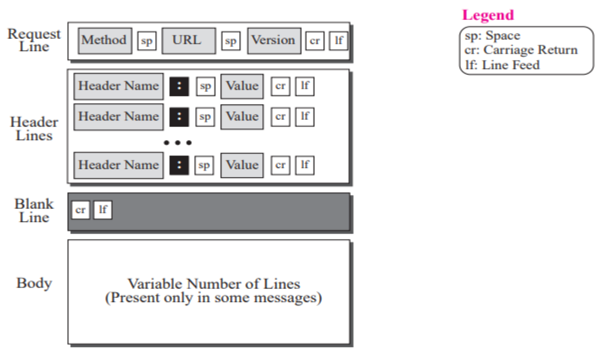
IP address which it would be using along with the DHCP servers IP address which had provide the respective. This packet is sent as broadcast. The packet, when received by the other DHCP server would understand that the IP address which it had leased to the client (In the DHCP offer packet) is not taken. So, the DHCP server would put the IP address back to its pool.

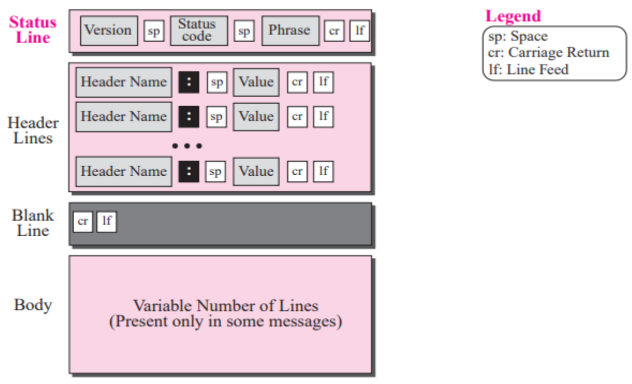
iii) 4M

The TCP/IP of the client would be configured with the option 'Obtain IP address automatically'. This is meant for DHCP clients. This configuration would automatically trigger a DHCP Discover packet from the PC. This packet would reach the DHCP server which would then respond with the DHCP offer packet.

17 Sketch the format of the HTTP request and response message. Illustrate the following scenario, assume in HTTP transactions for communication between client and server use the GET method to retrieve an image with the URL, path /usr/bin/image1. The client can accept images in GIF or JPEG format. The request does not have a body. The response message must contain the date, server, MIME version, and length of the document which is 2048 followed by a header, the body of the document can be blank.







18 Rahul sends mail to his parent. As Email has some limitations, supplementary protocols are used so that non-ASCII data can be sent through e-mail. Some specific header fields are added with respective to the conversion done in the message.

i. When the RFC subtype and Partial subtype will be used?

ii. In which type of the encoding scheme the non-ASCII character is represented as three characters.

iii. Explain how the following set of bits (Non-Ascii Data) can be encoded using Base 64.

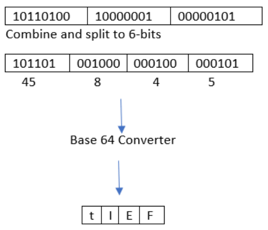
10110100	10000001	00000101

iv. Draw the structure of MIME Header for MIME version 1.1.

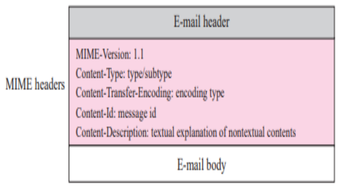
i) RFC822, partial, and external-body. The subtype RFC822 is used if the is encapsulating another message (including header and the body). The partial subtype is used if the original message has been fragmented into different mail messages and this mail message is one of the fragments. The fragments must be reassembled at the destination by MIME. Three parameters must be added: id, number, and the total. The id identifies the message and is present in all the fragments. The number defines the sequence order of the fragment. The total defines the number of fragments that comprise the original message.

ii) Quoted-printable

iii) Base64 transforms this type of data to printable characters, which can then be sent as ASCII characters or any type of character set supported by the underlying mail transfer mechanism. Base64 divides the binary data (made of streams of bits) into 24-bit blocks. Each block is then divided into four sections, each made of 6 bits. (3m)



iv)


E-mail header
MIME headers
MIME-Version: 1.1
Content-Type: type/subtype
Content-Transfer-Encoding: encoding type
Content-Id: message id
Content-Description: textual explanation of nontextual contents
E-mail body