Q2] The main functionality of *getservbyname() is to get by port number by feeding the domain name as input.

Struct servent getservbyname(const char *servname, const char *protoname)

```
Struct servent {
    char * s_name;
    char * s_aliases;
    int  s_port;
    char * s_proto;
}
```

This is the general structure of getservbyname(). It contains the common aliases of the domain name that is fed and also the port number from where service has to be fetched by name.

Q3) Both getsockopt and setsockopt are used to manipulate the options which are present. Options can be of two types;-

Binary options can have a value of 0 or 1;

Along with getsockopt and setsockopt, bcmp and memset are also used commonly.

getsockopt(sockfd, level, *optname, *optval, sizeof(opt))

This is the structure of getsockopt, similarly it is used

to get the socket fd by passing in the option name.

Similarly setsockopt has the following structure:-

setsockopt(sockfd, level, optname, *optval, sizeof(opt));

this is used to set a socket file discription by the optionname fd.

both gethostbyname & gethostbyaddr support only IPV4.

getaddrinfo supports IPV6.

int addrinfo (const char * host name, const char * service, const struct addrinfo. *hints, struct addrinfo ** result)

to and addrinfo structure that the caller fills in with hints.

```
struct addrinfo {
    int   ai_flags;
    int   ai_family;
    int   ai_socktype;
    int   protocol;
    socklen_t ai_addrlen
    char * ai_canonname;
    struct sockaddr  * ai_addr;
    struct addrinfo * ai_next;
}
```

while the structure of gethostbyname is

struct hostent * gethostbyname (const char * hostname)

return non null if OK, NULL on error with h_errno set.

~~# include <stdio.h>~~

struct {
char *h_name;
char *h_aliases;
~~char~~ int h_addr type;
int h_length,
char R* h_addr_list;
Return IPV4 addresses
}

a)
i)
#include<stdio.h>
#include<string.h>
#include<sys/stat.h>
#include<netinet.h>
#include<arpa/inet.h>
#define PORT 8080
#define MAXLINE 1024
int main() {
int sockfd;
char buffer[MAXLINE];
struct sockaddr_in servaddr;
if (sockfd = socket(AF_INET, SOCK_DGRAM, 0))<0){
perror("socket creation failed");
exit();
}
memset(&servaddr, 0, sizeof(servaddr));
memset(&cliaddr, 0, sizeof(cliaddr));

```
servaddr.sin_family = AF_INET
servaddr.sin_addr = INADDR_ANY
servaddr.sin_port = htons(PORT);
if (bind(sockfd, const struct sockaddr *)){
        perror("bind failed")
        exit();
    }
int len, n;
len = sizeof(cliaddr);
while(1){                                   MSG_WAITALL
    n = recvfrom(sockfd(char), buffer, MAXLINE, MSG);

    buffer[n] = '\0'
    printf("Client : %s", buffer);
    printf("message sent\n");
        sendto(sockfd(const char *buffer, strlen(buffer), MSG_
                CONFIRMED, const struct
                sockaddr *)
        printf("message sent\n")
        close()
        return 0;
    }
```

ii)

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <netinet.h>
#include <arpa/inet.h>
#define PORT 8080
```

```
#define MAXLINE 1024

int main(){
    int sockfd;
    char buffer[MAXLINE];
    struct sockaddr_in servaddr;
    if((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0)
        perror("socket failed");
    }memset(&servaddr, 0, sizeof(servaddr));
    int n, len;
    while(1){
    printf("enter the message");
        fgets(buffer, MAXLINE, stdin);
        buffer[strcspn(buffer, "\n")] = '\0';
        sendto(sockfd(const char *buffer),
            strlen(buffer, MSG_CONFIRM, (const struct
            sockaddr *)
        servaddr, sizeof(servaddr);
        printf("message sent");
            n = recvfrom(sockfd, (char*)buffer, MAXLINE,
            MSG_WAIT, NULL);
                buffer[n] = '\0';
                printf("server: %s", buffer);
            }
            close()
        return 0;
    }
```

QUIZ

1. The DNS (Domain Name System) is used to map the host to IP address ~~of IP to host~~. It uses the function gethostbyaddr() to execute this. Real life example: In a reverse DNS lookup, gethostbyaddr() is called to map the IP address to the host
2. The server will expect an input from a client sending the IP-bind-host packet. After some time of not recieving the packet the process dies and the function is called off by ~~interup~~ interrupt.

3. UDP (User datagram Protocol) does not guarantee the delivery of packets; it is hence also described as ~~used~~ unreliable datagram protocol.

4. gai-strerror() is used to correct the error got from the getaddr() stream.

5. The recvfrom() function allows the caller to mention the sender's IP address to recieve the packets from that specific caller unlike the recv() call which lets any generic sender.

b) i) When the service request is sent from client to server, the request might get lost sometimes on the way. The UDP client will be trying to listen to incoming request using recvfrom, but it does not know the request is lost.

ii) In such a scenario, the UDP puts a timeout, the recvfrom function if it does not recieve a request within a speculated time, it will be notified that the packet has been lost. Then either a new request is sent by client or the prev request is abandoned.

c) i) recvfrom is used to recieve a request from the client or an acknowledgement from the server. the recvfrom specifies about who the sender is.

recvfrom(socket fd, Homeaddr, sockaddr, sizeof(sockaddr))

While in sendto, it is used to send the client an acknowledgement of the request recieved or it is used to send the server a service request. Send to takes in the sender's address as an argument and sends the request. The return type of both functions is the length of the request they make.

ii) Both functions are extensively used in UDP for connectionless communication.

Sendto(sockfd, senderaddr, sockaddr, sizeof(sockaddr));