UNP PROBABLE QUESTIONS-2022

UNIX Network Programming (CSE 4042)

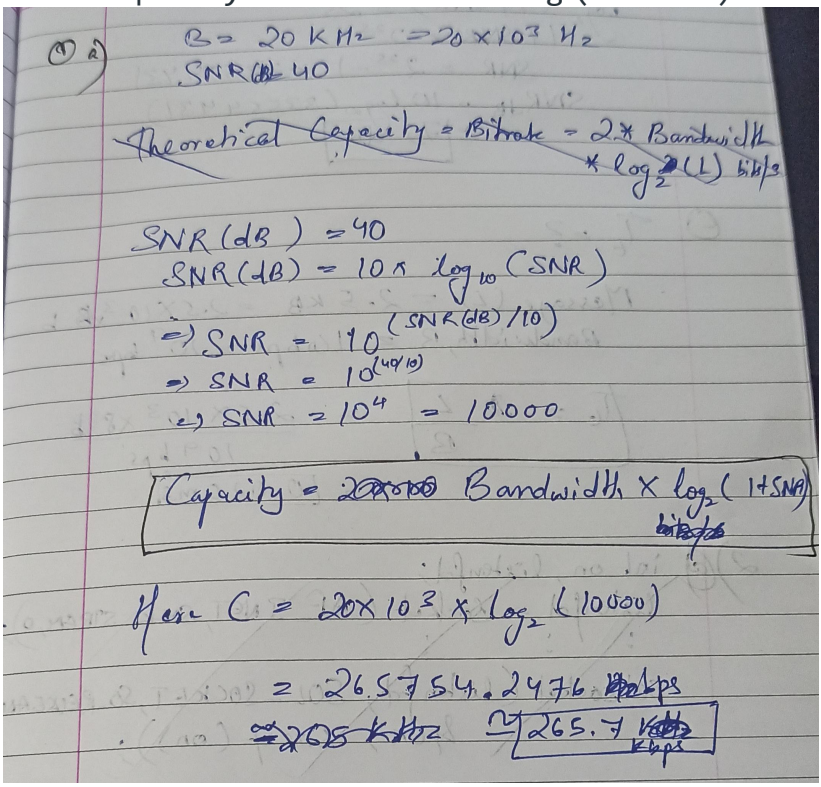
Programme: B.Tech. **Semester: 6th**
Instructor: Dibyasundar Das **Set : 1**

NB: Each bit carry 2 marks.

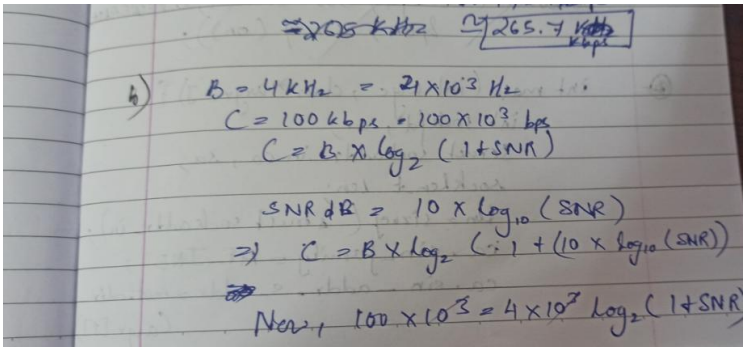
1. Answer each of the following in brief. [2 × 3]

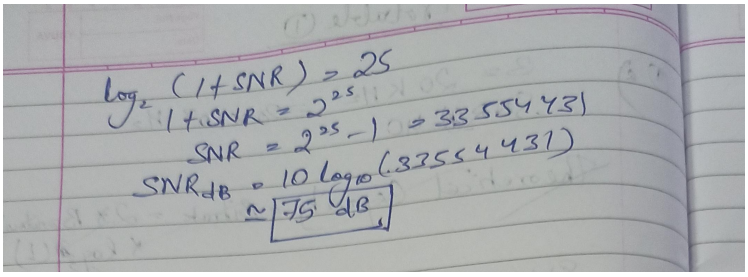
   (a) What is the theoretical capacity of a channel if the bandwidth is 20 KHz and SNR is 40dB?

Capacity = bandwidth * $\log_2$(1 + SNR) bits/sec

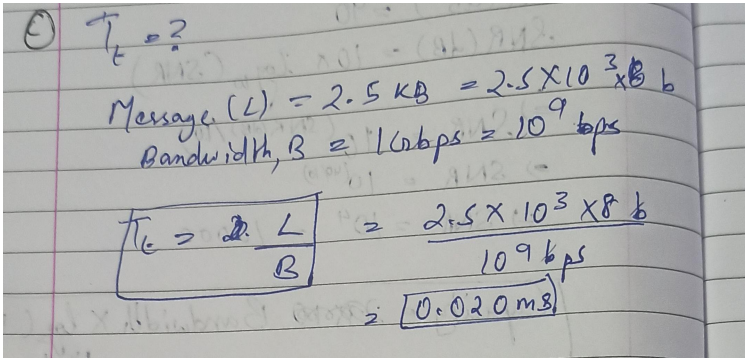


(b) We have a channel with 4 KHz bandwidth. If we want to send data at 100 Kbps,what is the minimum SNR in dB?





(c) What is the transmission time for a 2.5-kbyte message (an e-mail) if the bandwidth of the network is 1 Gbps?



2. Answer each of the following in brief. [2 × 3]

(a) Write a function to print ip and port details of local and remote end of a TCP socket.

```
int main(int argc, char *argv[]){
    int sockfd,cr;
    struct sockaddr_in ca,sa;
    socklen_t len;
    len=sizeof(struct sockaddr_in);
    ca.sin_family=AF_INET;
```

```
    ca.sin_addr.s_addr=inet_addr(argv[1]);
    ca.sin_port=htons(atoi(argv[2]));
    sockfd=socket(AF_INET,SOCK_STREAM,0);
    cr=connect(sockfd,(struct sockaddr *)&ca,sizeof(ca));
    if(cr==0){
        getpeername(sockfd,(struct sockaddr *)&sa,&len);
        printf("Sver port=%d\n",ntohs(sa.sin_port));
        printf("Server IP=%s\n",inet_ntoa(sa.sin_addr));
}}
```

(b) A TCP server with ip 200.130.40.1 is listening on port number 30. Write the code block for client to send and receive data from the server?

```
/*TCP client echo */
#include<stdio.h>
#include<unistd.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<arpa/inet.h>
#include<stdlib.h>
#include<string.h>
#include<signal.h>
#define MAXLINE 200
void str_cli(FILE *fp, int fd)
{
        int n;
        char buffer[MAXLINE];
        while(fgets(buffer, MAXLINE, fp) !=NULL)
        {
                write(fd, buffer, strlen(buffer));
                n = read(fd,  buffer, MAXLINE);
                if(n ==0)
                {
                    printf("Connection ended ...\n");
                    break;
                }
                if(n<0)
                {
                    printf("Read filed .... \n");
                }
                buffer[n]='\0';
                printf("data recieved from server : %s\n", buffer);
        }

}

int main(int argc, char *argv[]){
    int sockfd,cr;
    struct sockaddr_in ca,sa;
    socklen_t len;
    len=sizeof(struct sockaddr_in);
    if(argc!=3){
        fprintf(stderr,"Usage %s  <IP>  <PORT>\n",argv[0]);
        return 1;
    }
```

```
    ca.sin_family=AF_INET;
    ca.sin_addr.s_addr=inet_addr(argv[1]);
    ca.sin_port=htons(atoi(argv[2]));
    sockfd=socket(AF_INET,SOCK_STREAM,0);
    if(sockfd>0){
        fprintf(stderr,"Socket created success\n");
    }
    else{
        fprintf(stderr,"socket creat error\n");
    return 1;
    }
    cr=connect(sockfd,(struct sockaddr *)&ca,sizeof(ca));
    if(cr==0){
        fprintf(stderr,"Connect success return=%d\n",cr);
        printf("Connected server details\n");
        getpeername(sockfd,(struct sockaddr *)&sa,&len);
        printf("Sver port=%d\n",ntohs(sa.sin_port));
        printf("Server IP=%s\n",inet_ntoa(sa.sin_addr));
```

```
    }
    else{
    fprintf(stderr,"Connect Error=%d\n",cr);
    exit(1);
    }
    str_cli(stdin, sockfd);
    return 0;
}
```

(c) Write a code to design a concurrent echo-server that provides service on portnumber 33456, and can only provide service to local clients. Make sure that no zombie process should be created.

```
#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>
// PORT number
#define PORT 33456

int main(){
    // Server socket id
    int sockfd, ret;
```

```
// Server socket address structures
struct sockaddr_in serverAddr;
// Client socket id
int clientSocket;
// Client socket address structures
struct sockaddr_in cliAddr;
// Stores byte size of server socket address
socklen_t addr_size;
// Child process id
pid_t childpid;
// Creates a TCP socket id from IPV4 family
sockfd = socket(AF_INET, SOCK_STREAM, 0);
// Error handling if socket id is not valid
if (sockfd < 0) {
        printf("Error in connection.\n");
        exit(1);
}
// Initializing address structure with NULL
memset(&serverAddr, '\0',sizeof(serverAddr));
// Assign port number and IP address to the socket created
serverAddr.sin_family = AF_INET;
serverAddr.sin_port = htons(PORT);

// 127.0.0.1 is a loopback address
serverAddr.sin_addr.s_addr= inet_addr("127.0.0.1");
// Binding the socket id with
// the socket structure
ret = bind(sockfd,(struct sockaddr*)&serverAddr,sizeof(serverAddr));
// Error handling
if (ret < 0) {
        printf("Error in binding.\n");
        exit(1);
}
// Listening for connections (upto 10)
if (listen(sockfd, 10) == 0) {
```

```
        printf("Listening...\n\n");
    }
    int cnt = 0;
    while (1) {
        // Accept clients and store their information in cliAddr
        clientSocket = accept(sockfd, (struct sockaddr*)&cliAddr,&addr_size);
        // Error handling
        if (clientSocket < 0) {
            exit(1);
        }

        // Displaying information of connected client
        printf("Connection accepted
from %s:%d\n",inet_ntoa(cliAddr.sin_addr),ntohs(cliAddr.sin_port));
        // Print number of clients connected till now
        printf("Clients connected: %d\n\n",++cnt);
        // Creates a child process
        if ((childpid = fork()) == 0) {
            // Closing the server socket id
            close(sockfd);
            // Send a confirmation message
            // to the client
            send(clientSocket, "hi client",strlen("hi client"), 0);
        }
    }
    // Close the client socket id
    close(clientSocket);
    return 0;
}
```

3. Answer each of the following in brief. [2 × 3]

(a) A UDP client want to verify the IP and port number of the server. Write a code to keep receiving data until data is received from correct server.

```
#include <stdio.h>
```

```
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#define PORT     8080
int main() {
    int sockfd;
    char buffer[MAXLINE];
    char *hello = "Hello from client";
    struct sockaddr_in      servaddr;
    // Creating socket file descriptor
    if ( (sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0 ) {
        perror("socket creation failed");
        exit(EXIT_FAILURE);
    }
    memset(&servaddr, 0, sizeof(servaddr));
    // Filling server information
    servaddr.sin_family = AF_INET;
    servaddr.sin_port = htons(PORT);
    servaddr.sin_addr.s_addr = INADDR_ANY;
    int n, len;
    sendto(sockfd, (const char *)hello, strlen(hello),MSG_CONFIRM, (const struct
sockaddr *) &servaddr,sizeof(servaddr));
    printf("Hello message sent.\n");
    n= recvfrom(sockfd, (char *)buffer, MAXLINE,MSG_WAITALL, (struct
sockaddr *) &servaddr,&len);
```

```
        buffer[n] = '\0';

        printf("Server : %s\n", buffer);

        close(sockfd);

        return 0;

    }
```

(b) Write a code to create a UDP socket and set the send buffer size to twice of the current value.

```
/*Socket : UDP Server*/
int main(int argc, char **argv){

int sockfd,fd,len,p;
struct sockaddr_in servaddr,clientaddr;
char buff[1024];
len=sizeof(struct sockaddr_in);
sockfd=socket(AF_INET,SOCK_DGRAM,0);
servaddr.sin_family=AF_INET;
servaddr.sin_addr.s_addr=htonl(INADDR_ANY);
servaddr.sin_port=htons(0);
bind(sockfd,(struct sockaddr *)&servaddr, sizeof(servaddr));
getsockname(sockfd, (struct sockaddr *)&servaddr, &len);
printf("After bind ephemeral port=%d\n",ntohs(servaddr.sin_port));
recvfrom(sockfd,&p,4,0,(struct sockaddr *)&clientaddr,&len);
printf("\nClient send::%d\n",p);
printf("\nGive a string to send for client::");
scanf("%s",buff);
sendto(sockfd,buff,50,0,(struct sockaddr *)&clientaddr,len);
close(sockfd);
}
```

(c) Design a UDP echo-client that will resend the data if no response is received in 10 second.

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

```
#define MAXLINE 1024
int main(int argc,char* argv[])
{
int sockfd;
int n;
socklen_t len;
char sendline[1024],recvline[1024];
struct sockaddr_in servaddr;
strcpy(sendline,"");
printf("\n Enter the message : ");
scanf("%s",sendline);
sockfd=socket(AF_INET,SOCK_DGRAM,0);
bzero(&servaddr,sizeof(servaddr));
servaddr.sin_family=AF_INET;
servaddr.sin_addr.s_addr=inet_addr("127.0.0.1");
servaddr.sin_port=htons(5035);
connect(sockfd,(struct sockaddr*)&servaddr,sizeof(servaddr));
len=sizeof(servaddr);
sendto(sockfd,sendline,MAXLINE,0,(struct sockaddr*)&servaddr,len);
n=recvfrom(sockfd,recvline,MAXLINE,0,NULL,NULL);
recvline[n]=0;
printf("\n Server's Echo : %s\n\n",recvline);
return 0;
}
```

4. Answer each of the following in brief. [2 × 3]

   (a) Wite a code to use **`select`** function as as a replacement of sleep. Set the sleep time to 5 sec.

```
int main(void) {
fd_set rfds;
struct timeval tv;
int retval; /* Watch stdin (fd 0) to see when it has input. */
FD_ZERO(&rfds); FD_SET(0, &rfds); /* Wait up to five seconds. */
tv.tv_sec = 5;
 tv.tv_usec = 0;
retval = select(1, &rfds, NULL, NULL, &tv); /* Don't rely on the value of tv now! */
if (retval == -1)
        perror("select()");
else if (retval)
        printf("Data is available now.\n"); /* FD_ISSET(0, &rfds) will be true. */
else
        printf("No data within five seconds.\n");
return 0;
```

```
}
```

(b) Wite a code to use **poll** function as as a replacement of sleep. Set the sleep time to 5 sec.

```
int main() {
int timeout=5000,retpoll;
ssize_t byteread;
char buf[20];
struct pollfd fds[1];
fds[0].fd=0;
fds[0].events=POLLIN;
printf("Enter date within 5000 ms:\n");
retpoll=poll(fds,1,timeout);
if(retpoll==0){
        printf("Time out: No data entered\n");
}
if(fds[0].revents){
        printf("Data available:\n");
        byteread=read(0,buf,10);
        write(1,buf,byteread);
}
return 0;
}
```

(c) What is **pselect** ? Write it's prototype and describe how its better than **select**.

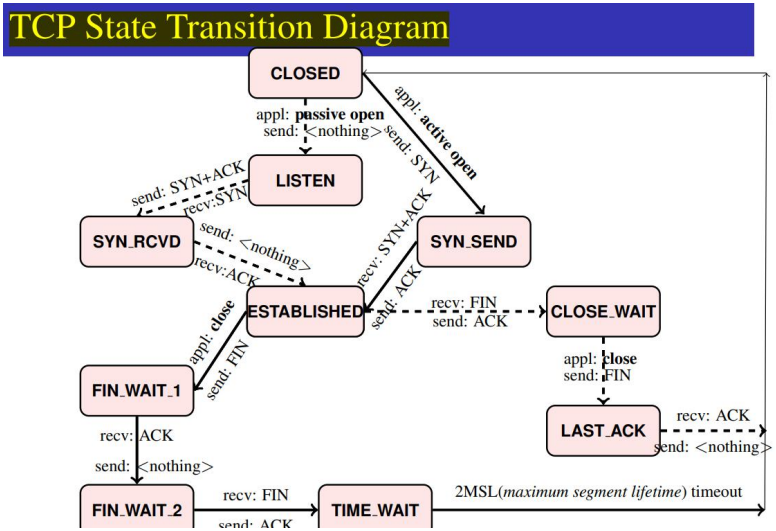
```
int pselect (int maxfdp1, fd_set * readset, fd_set * writeset, fd_set * exceptset,const struct
        timespec * timeout, const sigset_t * sigmask ) ;
```

Returns: count of ready descriptors, 0 on timeout, 1 on error

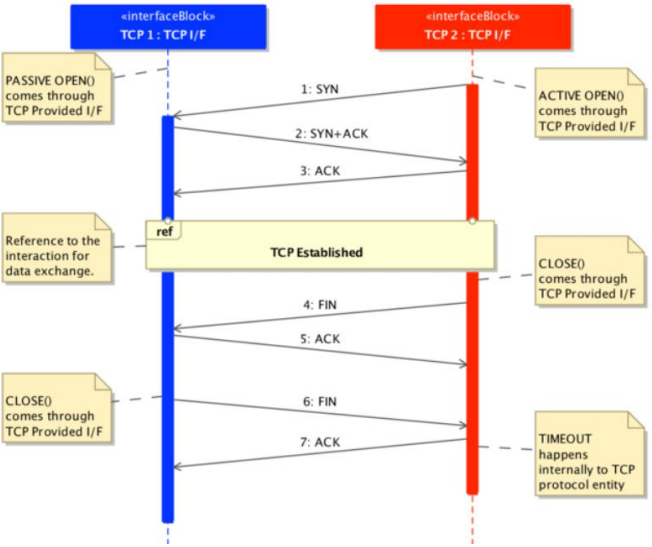
pselect contains two changes from the normal select function:
1. pselect uses the timespec structure, another POSIX invention, in stead of the timeval structure as in select.
2. pselect adds a sixth argument: a pointer to a signal mask.

5. Draw the following diagram. [2 × 4]

(a) Draw TCP state transition diagram for server.

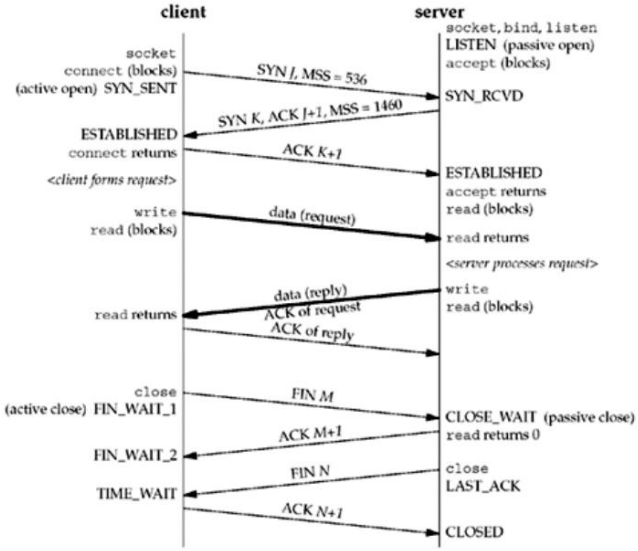


(b) Draw TCP function interaction diagram for client-server model.



(c) Draw Kernel interaction diagram for TCP server.

(d) Draw the Packet exchange diagram for TCP connection establishment and termination.



6. Answer each of the following in brief. [2 × 3]

(a) Compare between **sockaddr_in**, **sockaddr_in6** , and **sockaddr_un** structures.

For IPv4 it is *sockaddr_in*, and IPV6 *sockaddr_in6*, and *sockaddr_un* for *AF_UNIX* socket. *sockaddr* are used as the common data strut in the signature of APIs.

Both *sockaddr_in* and *sockaddr_in6* have *sa_family* .

sizeof(sockaddr_in6) > *sizeof(sockaddr)*, which cause allocate memory base on size of sockaddr is not enough for ipv6(that is error-prone).

(b) Show use of **inet_aton**, **inet_addr**, and **inet_ntoa** functions with example.

inet addr, and inet aton functions convert an IPv4 ad dresses from a dotted-decimal string (e.g. 206.234.56.78) to its 32-bit network byte ordered binary value. inet ntoa function does the reverse.

inet_aton: coverts C charcter string pointed to by **strptr** into its 32-bit binary betwork byte ordered value, which is strored through the pointer **addptr**. If successful, 1 is returned; otherwise, 0 is returned.

```
struct sockaddr_in servaddr;
inet_aton(argv[1], &servaddr.sin_addr);
```

inet_addr: same conversion as like **inet_aton**, returning the 32-bit binary network byte ordered value. The function returns the constant **INADDR_NONE** (typically 32 one-bits) on an error.

```
struct sockaddr_in servaddr;
servaddr.sin_addr.s_addr=inet_addr(argv[1]);
```

inet_aton: coverts C charcter string pointed to by **strptr** into its 32-bit binary betwork byte ordered value, which is strored through the pointer **addptr**. If successful, 1 is returned; otherwise, 0 is returned.

```
struct sockaddr_in servaddr;
inet_aton(argv[1], &servaddr.sin_addr);
```

inet_addr: same conversion as like **inet_aton**, returning the 32-bit binary network byte ordered value. The function returns the constant **INADDR_NONE** (typically 32 one-bits) on an error.

```
struct sockaddr_in servaddr;
servaddr.sin_addr.s_addr=inet_addr(argv[1]);
```

inet_ntoa: converts a 32-bit binary network byte ordered IPv4 address into its corresponding dotted-decimal string.

```
struct sockaddr_in servaddr;
printf("IP::%s\n", inet_ntoa(servaddr.sin_addr));
```

(c) Write short notes on **inet_pton** and **inet_ntop** functions.

inet_pton: convert the string pointed to by **strptr**, storing the binary result through the pointer **addrptr**. If successful, the return value is 1. If the input string is not a valid presentation format for the specified family, 0 is returned.

```
struct sockaddr_in servaddr;
inet_pton(AF_INET, argv[1], &servaddr.sin_addr);
```

inet_ntop: does the reverse conversion, from numeric (**addrptr**) to presentation (**strptr**).

```
#define INET_ADDRSTRLEN 16
struct sockaddr_in servaddr;
char strptr[INET_ADDRSTRLEN];
inet_ntop(AF_INET, &servaddr.sin_addr, strptr,
                                    INET_ADDRSTRLEN);
printf("IP::%s\n", strptr);
```

(d)

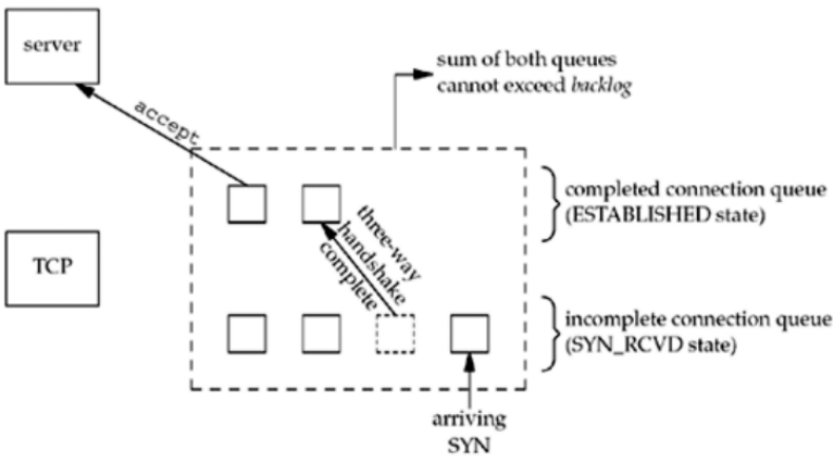
7. (a)

AF INET	-	IPv4 protocols
AF INET6	-	IPv6 protocols
SOCK STREAM	-	stream socket
IPPROTO TCP	-	TCP transport protocol
AF LOCAL	-	Unix domain protocols
SOCK DGRAM	-	datagram socket

(b)

```
/*Socket : Day Time Server*/
#include<stdio.h>
#include<sys/socket.h>
#include<sys/types.h>
#include<netinet/in.h>
#include<arpa/inet.h>
#include<stdlib.h>
#include<string.h>
#include<time.h>
int main(int argc, char **argv)
{
  int listenfd,connfd,len;
  struct sockaddr_in servaddr,clientaddr;
  char buff[1024];
  time_t ticks;
  len=sizeof(struct sockaddr_in);
  listenfd=socket(AF_INET,SOCK_STREAM,0);
  servaddr.sin_family=AF_INET;
  servaddr.sin_addr.s_addr=htonl(INADDR_ANY);
  servaddr.sin_port=htons(0);
  bind(listenfd,(struct sockaddr *)&servaddr, sizeof(servaddr));
  getsockname(listenfd, (struct sockaddr *)&servaddr, &len);
  printf("After bind ephemeral port=%d\n",(int)ntohs(servaddr.sin_port));
  listen(listenfd,5);
  connfd=accept(listenfd, (struct sockaddr *)&clientaddr,&len);
  ticks=time(NULL);
  snprintf(buff,sizeof(buff),"%s\r\n",ctime(&ticks));
  write(connfd,buff,strlen(buff));
  write(connfd,"ITER",4);
  close(connfd);
}
```

(c)



8. (a)

(b)

```
int whichisready(int fd1, int fd2) {
    int maxfd, nfds;
    fd_set readset;
    maxfd = (fd1 > fd2) ? fd1 : fd2;
    FD_ZERO(&readset);
    FD_SET(fd1, &readset);
    FD_SET(fd2, &readset);
    nfds = select(maxfd+1, &readset, NULL, NULL, NULL);
    if (nfds == -1)
        return -1;
    if (FD_ISSET(fd1, &readset))
        return fd1;
    if (FD_ISSET(fd2, &readset))
        return fd2;
    return -1;
}
```

(c)

9. (a)

```
int sockfd, len;
struct sockaddr_in serveraddr, clientaddr;
char buffer[100];

sockfd = socket(AF_INET, SOCK_STREAM, 0);

serveraddr.sin_family = AF_INET;
serveraddr.sin_port   = htons( 33456 );
serveraddr.sin_addr.s_addr = inet_addr(" 192.168.255.2      ");

len = sizeof(struct sockaddr_in);

connect(sockfd, (struct sockaddr *) &serveraddr, len);

read(fd, buffer, 99);

printf("Data recieved from sever : %s\n", buffer);

// Dispay own ip and port number
len = sizeof(struct sockaddr_in);
getsockname(sockfd, (struct sockaddr *) &clientaddr, &len);

printf("Client ip : %s\n", inet_ntoa(clientaddr.sin_addr));
printf("Client port : %d\n", ntohs(clientaddr.sin_port));
```

(b) **SO_LINGER**

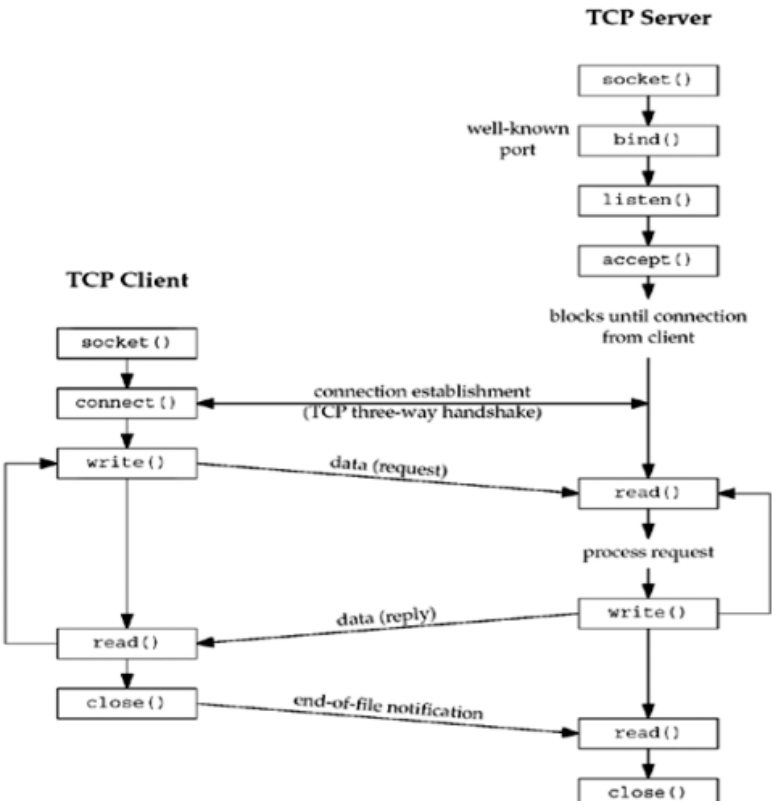
Lingers on close if data is present. If this option is enabled and there is unsent data present when close() is called, the calling application program is blocked during

the close() call, until the data is transmitted or the connection has timed out. If this option is disabled, the TCP/IP address space waits to try to send the data. Although the data transfer is usually successful, it cannot be guaranteed, because the TCP/IP address space waits only a finite amount of time trying to send the data.
The close() call returns without blocking the caller. This option has meaning only for stream sockets.

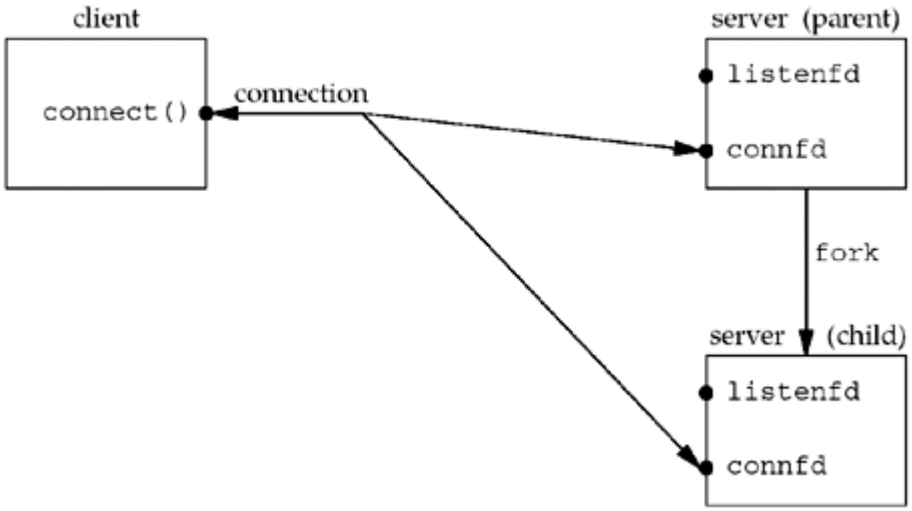
(c)

```c
#include<stdio.h>
#include<sys/socket.h>
#include <sys/un.h>
#include<string.h>
#include <unistd.h>
#include<stdlib.h>
int main(int argc, char *argv[])
{
        int sockfd;
        socklen_t len;
        struct sockaddr_un addr1, addr2;
        if (argc != 2)
        {
                printf("Enter␣path␣name");
                exit(0);
        }
        sockfd = socket(AF_LOCAL, SOCK_STREAM, 0);
        unlink(argv[1]);

        bzero(&addr1, sizeof(addr1));

        addr1.sun_family = AF_LOCAL;
        strcpy(addr1.sun_path, argv[1]);
        bind(sockfd, (struct sockaddr *) &addr1, strlen(addr1.sun_path)
                                +sizeof (addr1.sun_family));
        len = sizeof(addr2);
        getsockname(sockfd, (struct sockaddr *) &addr2, &len);
        printf("bound␣name␣=␣%s,␣returned␣len␣=␣%d\n", addr2.sun_path, len);
        return(0);
}
```

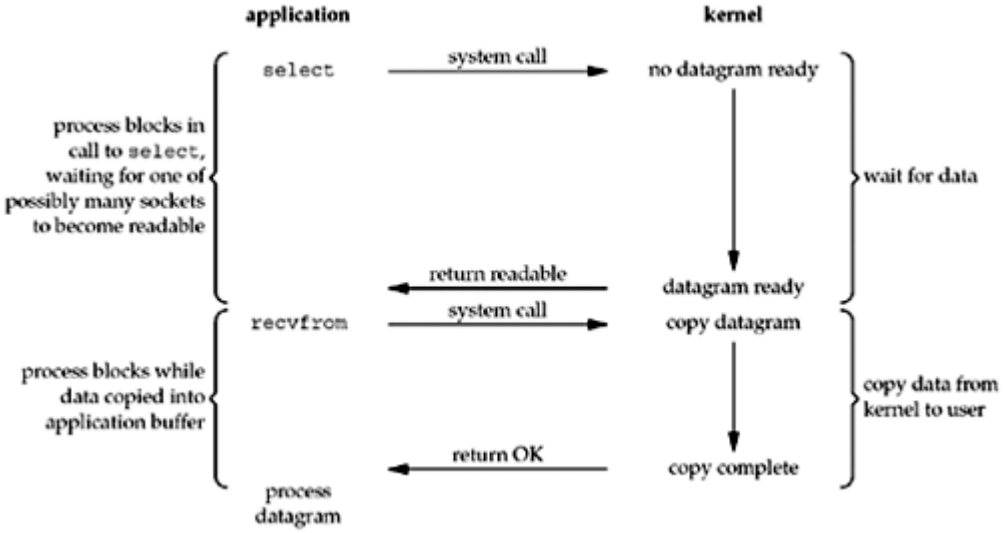
10. (a)



(b)



(c)



11. (a**) Class-full addressing :** In Classful addressing, the address space is divided into five classes: A, B, C, D, and E. Each of these classes has a valid range of IP addresses. Classes D and E are reserved for multicast and experimental purposes respectively. The order of bits in the first octet determine the classes of IP address.

    (b**) Private IP :** A private IP address is a range of non-internet facing IP addresses used in an internal network. Private IP addresses are provided by network devices, such as routers, using network address translation. Private IP addresses are commonly used for local area networks in residential, office and enterprise areas.

    (c) **Asynchronous I/O Model:** In computer science, asynchronous I/O (also non-sequential I/O) is a form of input/output processing that permits other processing to continue before the transmission has finished. A name used for asynchronous I/O in the Windows API is overlapped I/O.

12. (a) 116.29.118.26/19

= 0111 0100 0001 1101 0111 0110 0001 1010

Network Address = 0111 0100 0001 1101 0110 0000 0000 0000 (last 13 as 32-19=13)

= 116.29.96.0/19

Subnet mask = 255.225.224.0 (Class A)

Address = 116.0.0.0

(b) Class B, Subnet 172.16.13.0, Broadcast address 172.16.13.127

(c) 8 subnets, each with 8,190 hosts.

(13)
(a)
===

```
#include<stdio.h>     //printf
#include<string.h>    //strlen
#include<sys/socket.h> //socket
#include<arpa/inet.h>  //inet_addr
#include <fcntl.h>    //for open
#include <unistd.h>    //for close

#include <sys/types.h>  //sendto
#include <sys/socket.h> //sendto

#define DEFAULT_BUFLEN     512
#define DEFAULT_CLIENT_PORT     27016
#define DEFAULT_SERVER_PORT     27015
#define NUM_OF_PAR     7

int main(int argc, char *argv[])
{
   int sock;
   struct sockaddr_in server, client;
   char message[DEFAULT_BUFLEN];

      //Create socket
      sock = socket(AF_INET, SOCK_STREAM, 0);
      if (sock == -1)
      {
         printf("Could not create socket");
      }
      puts("Socket created");

      server.sin_addr.s_addr = inet_addr("ip_address");
      server.sin_family = AF_INET;
      server.sin_port = htons(DEFAULT_SERVER_PORT);
//Connect to remote server
      if(connect(sock, (struct sockaddr *)&server, sizeof(server)) < 0)
      {
         perror("connect failed. Error");
         return 1;
      }

      puts("Connected\n");

   int array[] = {1, 12, 3, 3, 300, 5, 3};

   printf("Sending array\n");
   for(int i = 0; i < NUM_OF_PAR; i++){
      printf("%d  ", array[i]);
 }

   if(send(sock, array, NUM_OF_PAR, 0) < 0)
   {
      printf("\nArray is not successfully sent\n");
   }
   else
   {
```

```
      printf("\nArray is sent\n");
   }

   printf("\n");

      close(sock);

      return 0;
}
```

(b)

```
===
#include<stdio.h>   //printf
#include<string.h>  //strlen
#include<sys/socket.h>  //socket
#include<arpa/inet.h>   //inet_addr
#include<unistd.h>     //write

#include<stdlib.h>

#include <sys/types.h>  //recvfrom
#include <sys/socket.h> //recvfrom

#define DEFAULT_BUFLEN     512
#define DEFAULT_CLIENT_PORT     27016
#define DEFAULT_SERVER_PORT     27015
#define NUM_OF_PAR     7

int main(int argc , char *argv[])
{
   int socket_desc, client_sock, c, read_size;
   struct sockaddr_in server, client;
   char client_message[DEFAULT_BUFLEN];
   int sock;

   //Create socket
   socket_desc = socket(AF_INET, SOCK_STREAM, 0);
   if (socket_desc == -1)
   {
         printf("Could not create socket");
      }
 puts("Socket created");

   //Prepare the sockaddr_in structure
   server.sin_family = AF_INET;
   server.sin_addr.s_addr = INADDR_ANY;
      server.sin_port = htons(DEFAULT_SERVER_PORT);

      //Bind
      if(bind(socket_desc, (struct sockaddr *)&server, sizeof(server)) < 0)
      {
         //print the error message
         perror("bind failed. Error");
         return 1;
      }
      puts("bind done");
```

```
    //Listen
    listen(socket_desc, 1);

    //Accept and incoming connection
    puts("Waiting for incoming connections...");
    c = sizeof(struct sockaddr_in);

    //accept connection from an incoming client
    client_sock = accept(socket_desc, (struct sockaddr *)&client, (socklen_t*)&c);
    if(client_sock < 0)
    {
       perror("accept failed");
       return 1;
 }
    puts("Connection accepted");


    //Receive a message from client
  int arrayReceived[NUM_OF_PAR];

  while((read_size = recv(client_sock, &arrayReceived, NUM_OF_PAR, 0)) > 0){
    printf("Received well\n");
  }

  if(read_size == 0)
    {
       puts("Client disconnected");
    }
    else if(read_size == -1)
    {
       perror("recv failed");
    }
 printf("Received parameters:    \n");
  for(int i = 0; i < NUM_OF_PAR; i++)
  {
    printf("%d  ", arrayReceived[i]);
  }

  printf("\n");

    close(sock);

    return 0;
}
(c)
===
void sig_chld(int signo)
{
  pid_t pid;
  int stat;

  // while ((pid = wait(&stat)) > 0)
  while ((pid = waitpid(-1, &stat, WNOHANG)) > 0)
  {
    printf("child %d terminated\n", pid);
  }
}
```

(2)
====
(a)
==
A synchronous I/O operation causes the requesting process to be blocked until that I/O operation completes.

An asynchronous I/O operation does not cause the requesting process to be blocked.

5 I/o model are
blocking I/O

nonblocking I/O

I/O multiplexing (select and poll)

signal driven I/O (SIGIO)

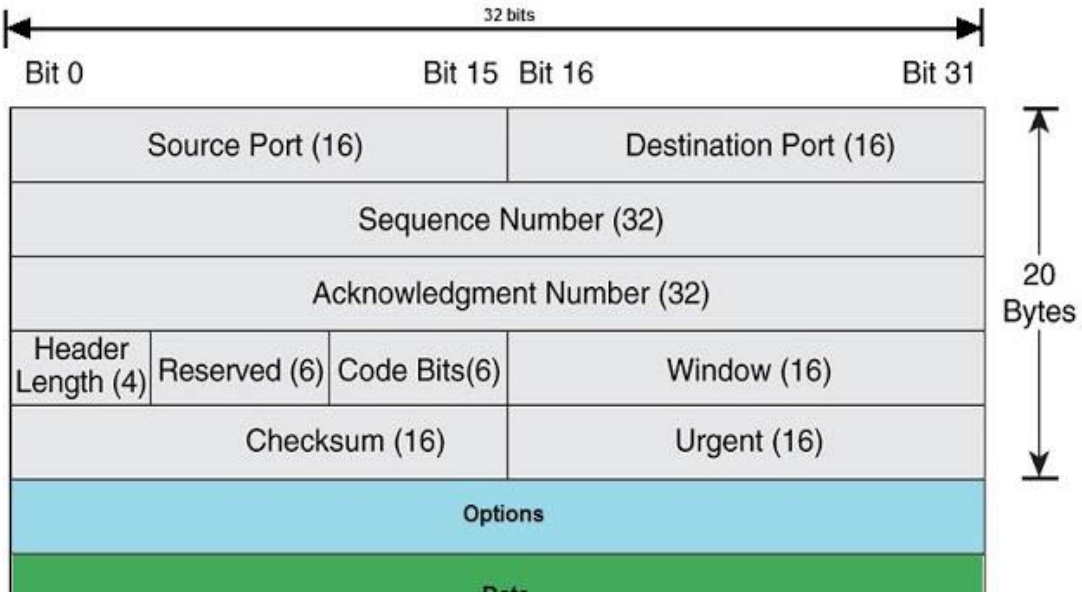
asynchronous I/O (the POSIX aio_functions)

(b)
===
The setsockopt function called with the SO_KEEPALIVE socket option allows an application to enable keep-alive packets for a socket connection.
The SO_KEEPALIVE option for a socket is disabled (set to FALSE) by default.

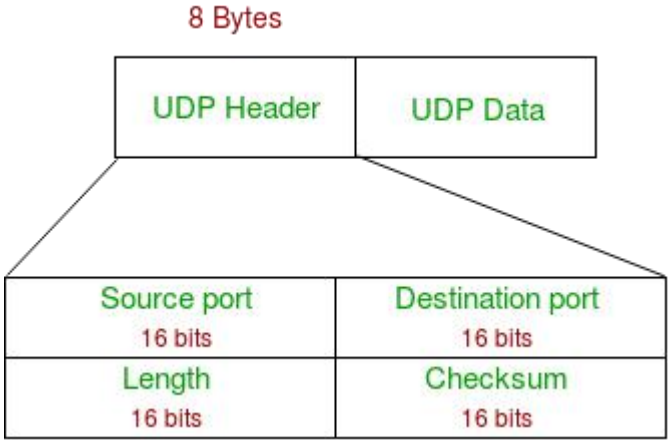
(c)
===
```
len = sizeof(rcvbuf);
getsockopt(sockfd, SOL_SOCKET, SO_RCVBUF, &rcvbuf, &len);
len = sizeof(mss);
getsockopt(sockfd, IPPROTO_TCP, TCP_MAXSEG, &mss, &len);
printf("defaults: SO_RCVBUF = %d, MSS = %d\n", rcvbuf, mss);

len2 = sizeof(sndbuf);
getsockopt(sockfd, SOL_SOCKET, SO_SNDBUF, &sndbuf, &len2);
//len2 = sizeof(mss2);
getsockopt(sockfd, IPPROTO_TCP, TCP_MAXSEG, &mss2, &len2);
printf("defaults: SO_SNDBUF = %d, MSS = %d\n", sndbuf, mss2);
```

(15)

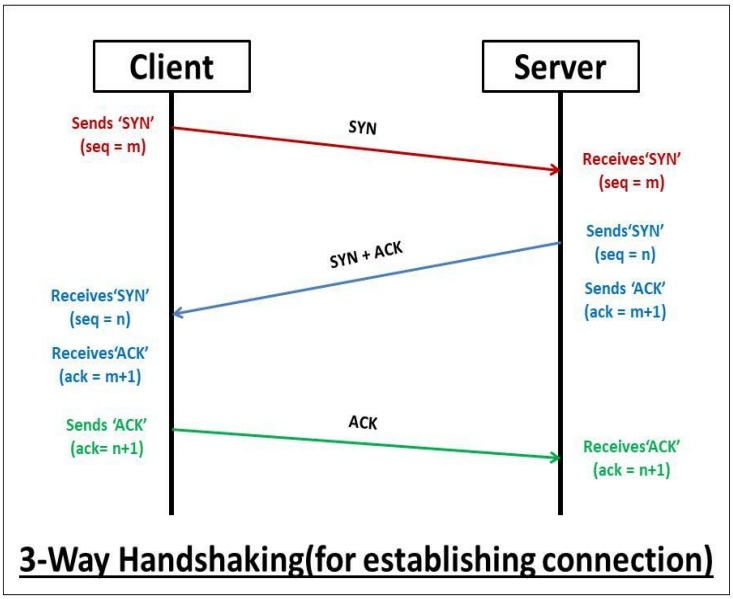
TCP header



UDp header:



3 way handshake diagram



(16)

(A)

TCP is reliable as it guarantees the delivery of data to the destination router.The delivery of data to the destination cannot be guaranteed in UDP.

(B)

If a connection request arrives before the server can process it, the request is queued until the server is ready. **When you call listen, you inform TCP/IP that you intend to be a server and accept incoming requests from the IP network**. By doing so, socket status is changed from active status to passive.

(c)

```
import socket

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

Here we made a socket instance and passed it two parameters. The first parameter is **AF_INET** and the second one is **SOCK_STREAM**. AF_INET refers to the address-family ipv4. The SOCK_STREAM means connection-oriented TCP protocol. Now we can connect to a server using this socket.3

(17)

(a)

fork() is how you create new processes in Unix. When you call *fork*, you're creating a copy of your own process that has its own address space. This allows multiple tasks

to run independently of one another as though they each had the full memory of the machine to themselves.

(b)
When a socket address structure is passed to any socket function, it is always passed by reference (a pointer to the structure is passed). The length of the structure is also passed as an argument.

©
The bind() function binds a unique local name to the socket with descriptor *socket*. After calling socket(), a descriptor does not have a name associated with it. However, it does belong to a particular address family as specified when socket() is called. The exact format of a name depends on the address family.
socket
The socket descriptor returned by a previous socket() call.
address
The pointer to a **sockaddr** structure containing the name that is to be bound to *socket*.
address_len
The size of *address* in bytes.

(18)
(b)para-1:&servaddr
Para-2: len

©
yes
yes
yes
yes
yes
yes
yes
yes
yes
yes
yes
yes
yes
yes
yes
yes
yes
yes