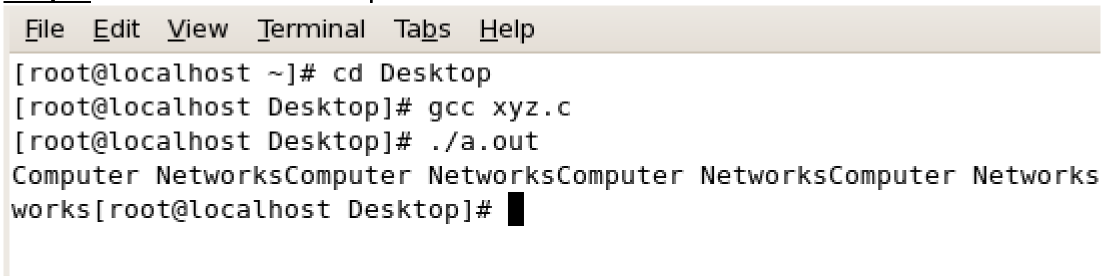
B.Sc.Computer Science VI Sem.

Computer Networks Lab

Write a program to display hello world using signals.

```
#include<stdio.h>
#include<stdlib.h>
#include<signal.h>
voidsign_handler(int);
int main()
{
        signal(SIGINT,sign_handler);
        while(1)
        {
                printf("Hello World");
                sleep(1);
        }
        return(0);
}

voidsign_handler(intnum)
{
        exit(1);
}
```

Output: Use Ctrl+C for Interrupt



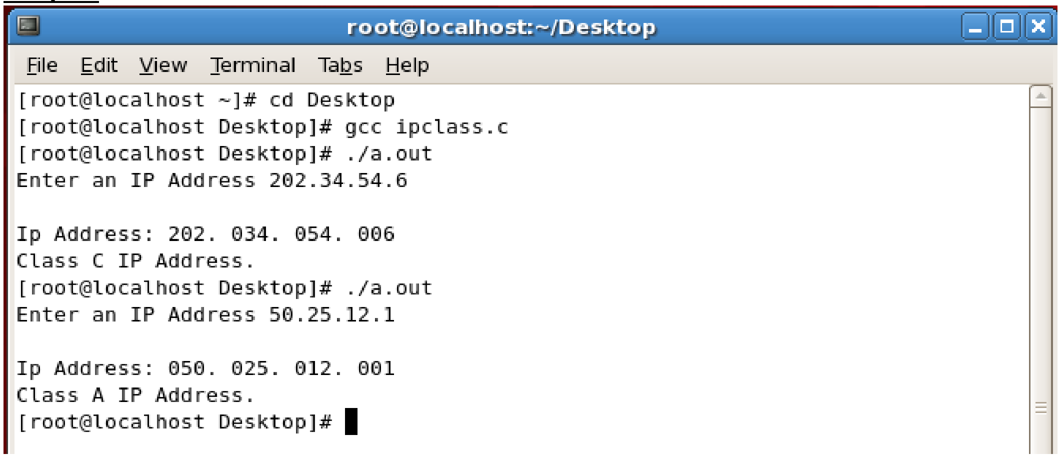
```
[root@localhost ~]# cd Desktop
[root@localhost Desktop]# gcc xyz.c
[root@localhost Desktop]# ./a.out
Computer NetworksComputer NetworksComputer NetworksComputer Networks
works[root@localhost Desktop]#
```

Note:
The **signal.h** different signals reported during a program's execution.
SIGINT -Interrupt signal such as ctrl-C.

Write a program to identify the category of IP address for a given IP address.

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

voidextractIpAddress(unsigned char *sourceString,short *ipAddress)
{
unsigned short len=0;
   unsigned char oct[4]={0},cnt=0,cnt1=0,i,buf[5];

len=strlen(sourceString);
   for(i=0;i<len;i++)
   {
      if(sourceString[i]!='.'){
buf[cnt++] =sourceString[i];
      }
      if(sourceString[i]=='.' || i==len-1){
buf[cnt]='\0';
cnt=0;
oct[cnt1++]=atoi(buf);
      }
   }
ipAddress[0]=oct[0];
ipAddress[1]=oct[1];
ipAddress[2]=oct[2];
ipAddress[3]=oct[3];
}

int main()
{
   unsigned char ip[20]={0};
   short ipAddress[4];

printf("Enter an IP Address ");
scanf("%s",ip);

extractIpAddress(ip,&ipAddress[0]);

printf("\nIp Address: %d. %d. %d.%d\n",ipAddress[0],ipAddress[1],ipAddress[2],ipAddress[3]);

   if(ipAddress[0]>=0 &&ipAddress[0]<=127)
printf("Class A IP Address.\n");
   if(ipAddress[0]>127 &&ipAddress[0]<191)
printf("Class B IP Address.\n");
   if(ipAddress[0]>191 &&ipAddress[0]<224)
printf("Class C IP Address.\n");
   if(ipAddress[0]>224 &&ipAddress[0]<=239)
printf("Class D IP Address.\n");
   if(ipAddress[0]>239)
printf("Class E IP Address.\n");

   return 0;
}
```

Output:



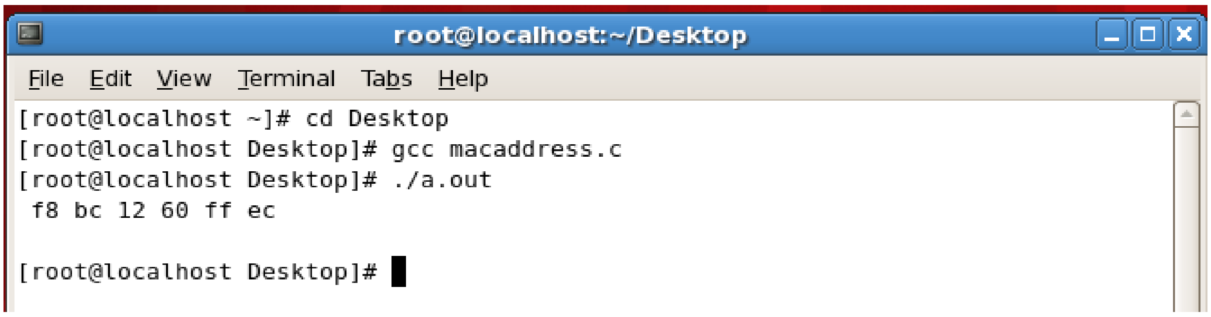
```
[root@localhost ~]# cd Desktop
[root@localhost Desktop]# gcc ipclass.c
[root@localhost Desktop]# ./a.out
Enter an IP Address 202.34.54.6

Ip Address: 202. 034. 054. 006
Class C IP Address.
[root@localhost Desktop]# ./a.out
Enter an IP Address 50.25.12.1

Ip Address: 050. 025. 012. 001
Class A IP Address.
[root@localhost Desktop]#
```

Note:
1) sourceString - String pointer that contains ip address
2) ipAddress - Target variable short type array pointer that will store ip address octets

Write a program to get MAC address.

```
#include<sys/socket.h>
#include <sys/ioctl.h>
#include <linux/if.h>
#include <netdb.h>
#include <stdio.h>
#include <string.h>

int main()
{
structifreq s;
intfd = socket(PF_INET, SOCK_DGRAM, IPPROTO_IP);

strcpy(s.ifr_name, "eth0");
   if (0 == ioctl(fd, SIOCGIFHWADDR, &s)) {
int i;
      for (i = 0; i < 6; ++i)
printf(" %02x", (unsigned char) s.ifr_addr.sa_data[i]);
      puts("\n");
      return 0;
   }

   return 1;
}
```

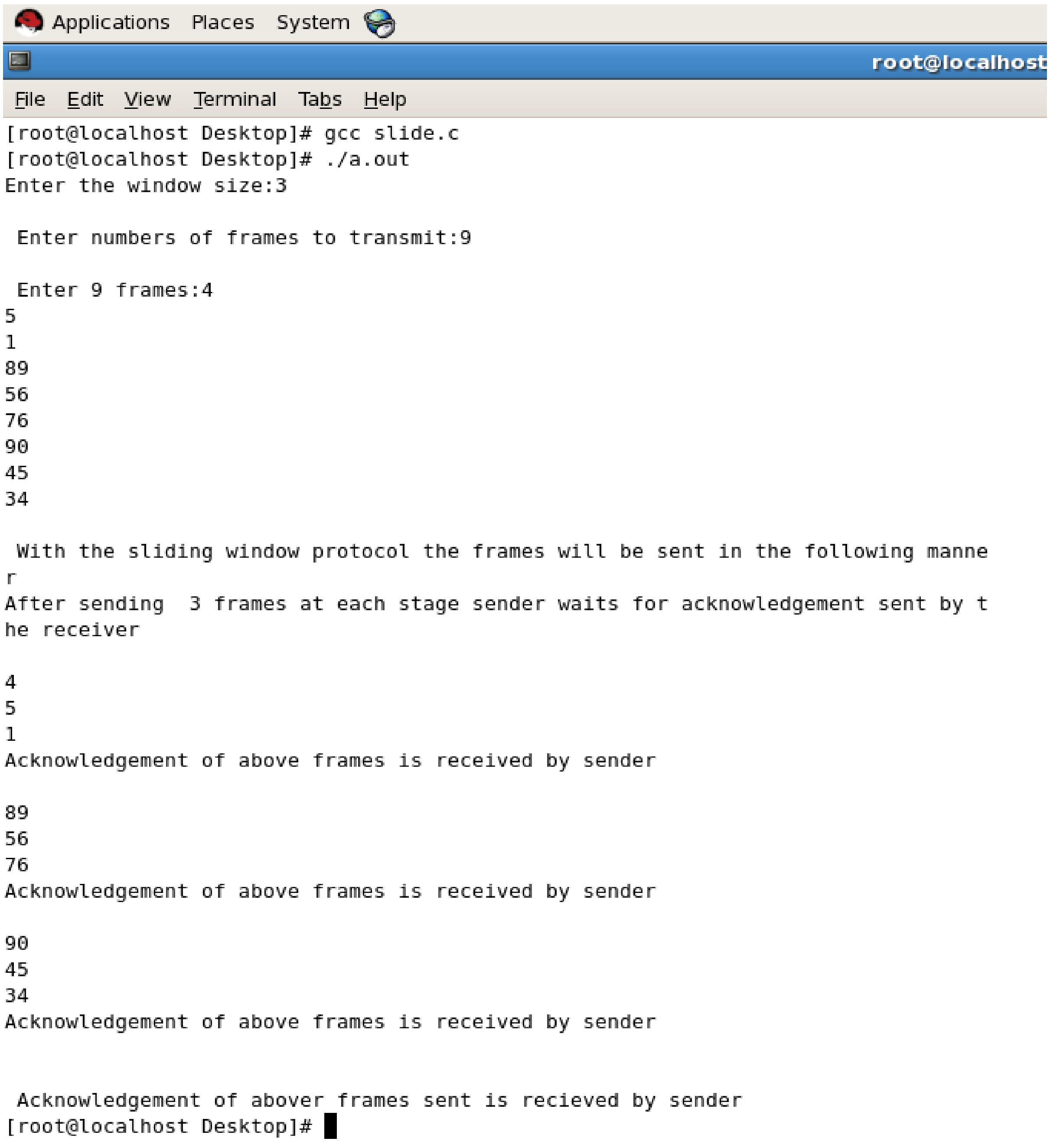
Output:



```
[root@localhost ~]# cd Desktop
[root@localhost Desktop]# gcc macaddress.c
[root@localhost Desktop]# ./a.out
 f8 bc 12 60 ff ec

[root@localhost Desktop]#
```

Write a program to implement the sliding window protocol.

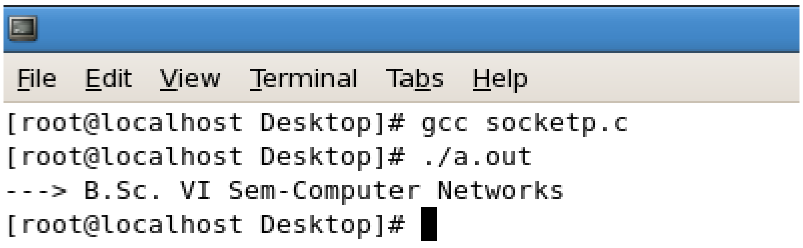
```
#include<stdio.h>

int main()
{
intw,f,i,frames[50];

printf("Enter window size: ");
scanf("%d",&w);

printf("\nEnter number of frames to transmit: ");
scanf("%d",&f);

printf("\nEnter %d frames: ",f);

   for(i=1;i<=f;i++)
scanf("%d",&frames[i]);

printf("\nWith sliding window protocol the frames will be sent in the following manner \n");
printf("After sending %d frames at each stage sender waits for acknowledgement sent by the
receiver\n\n",w);

   for(i=1;i<=f;i++)
   {
      if(i%w==0)
      {
printf("%d\n",frames[i]);
printf("Acknowledgement of above frames sent is received by sender\n\n");
      }
      else
printf("%d ",frames[i]);
   }

   if(f%w!=0)
printf("\nAcknowledgement of above frames sent is received by sender\n");

   return 0;
}
```



```
[root@localhost Desktop]# gcc slide.c
[root@localhost Desktop]# ./a.out
Enter the window size:3

 Enter numbers of frames to transmit:9

 Enter 9 frames:4
5
1
89
56
76
90
45
34

 With the sliding window protocol the frames will be sent in the following manne
r
After sending  3 frames at each stage sender waits for acknowledgement sent by t
he receiver

4
5
1
Acknowledgement of above frames is received by sender

89
56
76
Acknowledgement of above frames is received by sender

90
45
34
Acknowledgement of above frames is received by sender


 Acknowledgement of abover frames sent is recieved by sender
[root@localhost Desktop]# 
```

Write a program for socket pair system call using IPC.

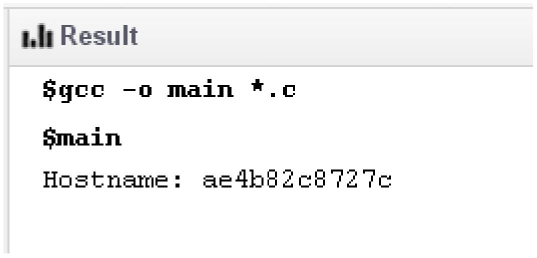
```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#define DATA "B.Sc. VI Sem-Computer Networks"

main()
{
int sockets[2], child;

   /* Create a pipe */
if (pipe(sockets) < 0) {
perror("opening stream socket pair");
exit(10);
   }

if ((child = fork()) == -1)
perror("fork");
else if (child) {
charbuf[1024];

      /*  This is still the parent.
          It reads the child's message. */
close(sockets[1]);
if (read(sockets[0], buf, sizeof(buf)) < 0)
perror("reading message");
printf("-->%s\n", buf);
close(sockets[0]);
   } else {
      /* This is the child.
         It writes a message to its parent. */
close(sockets[0]);
if (write(sockets[1], DATA, sizeof(DATA)) < 0)
perror("writing message");
close(sockets[1]);
   }
}
```



```
[root@localhost Desktop]# gcc socketp.c
[root@localhost Desktop]# ./a.out
---> B.Sc. VI Sem-Computer Networks
[root@localhost Desktop]# 
```

Write a program to print details of DNS host.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <netdb.h>

// Returns DNS hostname for the local computer
void checkHostName(int hostname)
{
    if (hostname == -1)
    {
        perror("gethostname");
        exit(1);
    }
}

// Returns DNS host information corresponding to host name
void checkHostEntry(struct hostent * hostentry)
{
    if (hostentry == NULL)
    {
        perror("gethostbyname");
        exit(1);
    }
}

// Driver code
int main()
{
   char hostbuffer[256];
   struct hostent *host_entry;
    int hostname;

    // To retrieve hostname
    hostname = gethostname(hostbuffer, sizeof(hostbuffer));
    checkHostName(hostname);

    // To retrieve host information
    host_entry = gethostbyname(hostbuffer);
    checkHostEntry(host_entry);
      printf("Hostname: %s\n", hostbuffer);
       return 0;
}
```



```
$gcc -o main *.c

$main
Hostname: ae4b82c8727c
```

Write a program to implement TCP echo using client–server program.

TCP-Server

```
#include <netdb.h>
#include <netinet/in.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#define MAX 80
#define PORT 8080
#define SA struct sockaddr

// Function designed for chat between client and server.
void func(int sockfd)
{
	char buff[MAX];
	int n;
	// infinite loop for chat
	for (;;) {
		bzero(buff, MAX);

		// read the message from client and copy it in buffer
		read(sockfd, buff, sizeof(buff));
		// print buffer which contains the client contents
		printf("From client: %s\t To client : ", buff);
		bzero(buff, MAX);
		n = 0;
		// copy server message in the buffer
		while ((buff[n++] = getchar()) != '\n')
			;

		// and send that buffer to client
		write(sockfd, buff, sizeof(buff));

		// if msg contains "Exit" then server exit and chat ended.
		if (strncmp("exit", buff, 4) == 0) {
			printf("Server Exit...\n");
			break;
		}
	}
}

// Driver function
int main()
{
	int sockfd, connfd, len;
	struct sockaddr_in servaddr, cli;

	// socket create and verification
	sockfd = socket(AF_INET, SOCK_STREAM, 0);
	if (sockfd == -1) {
		printf("socket creation failed...\n");
		exit(0);
	}
	else
		printf("Socket successfully created..\n");
	bzero(&servaddr, sizeof(servaddr));

	// assign IP, PORT
	servaddr.sin_family = AF_INET;
	servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
	servaddr.sin_port = htons(PORT);

	// Binding newly created socket to given IP and verification
	if ((bind(sockfd, (SA*)&servaddr, sizeof(servaddr))) != 0) {
		printf("socket bind failed...\n");
		exit(0);
	}
	else
		printf("Socket successfully binded..\n");

	// Now server is ready to listen and verification
	if ((listen(sockfd, 5)) != 0) {
		printf("Listen failed...\n");
		exit(0);
	}
	else
		printf("Server listening..\n");
	len = sizeof(cli);

	// Accept the data packet from client and verification
	connfd = accept(sockfd, (SA*)&cli, &len);
	if (connfd < 0) {
		printf("server acccept failed...\n");
		exit(0);
	}
	else
		printf("server acccept the client...\n");

	// Function for chatting between client and server
	func(connfd);

	// After chatting close the socket
	close(sockfd);
}
```

TCP-Client

```
#include <netdb.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#define MAX 80
#define PORT 8080
#define SA struct sockaddr
void func(int sockfd)
```

```
{
        char buff[MAX];
        int n;
        for (;;) {
                bzero(buff, sizeof(buff));
                printf("Enter the string : ");
                n = 0;
                while ((buff[n++] = getchar()) != '\n')
                        ;
                write(sockfd, buff, sizeof(buff));
                bzero(buff, sizeof(buff));
                read(sockfd, buff, sizeof(buff));
                printf("From Server : %s", buff);
                if ((strncmp(buff, "exit", 4)) == 0) {
                        printf("Client Exit...\n");
                        break;
                }
        }
}

int main()
{
        int sockfd, connfd;
        struct sockaddr_in servaddr, cli;

        // socket create and varification
        sockfd = socket(AF_INET, SOCK_STREAM, 0);
        if (sockfd == -1) {
                printf("socket creation failed...\n");
                exit(0);
        }
        else
                printf("Socket successfully created..\n");
        bzero(&servaddr, sizeof(servaddr));

        // assign IP, PORT
        servaddr.sin_family = AF_INET;
        servaddr.sin_addr.s_addr = inet_addr("127.0.0.1");
        servaddr.sin_port = htons(PORT);

        // connect the client socket to server socket
        if (connect(sockfd, (SA*)&servaddr, sizeof(servaddr)) != 0) {
                printf("connection with the server failed...\n");
                exit(0);
        }
        else
                printf("connected to the server..\n");

        // function for chat
        func(sockfd);

        // close the socket
        close(sockfd);
}
```

Write a program to implement UDP echo using client–server program.

UDP-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
#define MAXLINE 1024

// Driver code
int main() {
        int sockfd;
        char buffer[MAXLINE];
        char *hello = "Hello from server";
        struct sockaddr_in servaddr, cliaddr;

        // 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));
        memset(&cliaddr, 0, sizeof(cliaddr));

        // Filling server information
        servaddr.sin_family = AF_INET; // IPv4
        servaddr.sin_addr.s_addr = INADDR_ANY;
        servaddr.sin_port = htons(PORT);

        // Bind the socket with the server address
        if ( bind(sockfd, (const struct sockaddr *)&servaddr,
                        sizeof(servaddr)) < 0 )
        {
                perror("bind failed");
                exit(EXIT_FAILURE);
        }

        int len, n;
        n = recvfrom(sockfd, (char *)buffer, MAXLINE,
                                MSG_WAITALL, ( struct sockaddr *) &cliaddr,
                                &len);
        buffer[n] = '\0';
        printf("Client : %s\n", buffer);
        sendto(sockfd, (const char *)hello, strlen(hello),
                MSG_CONFIRM, (const struct sockaddr *) &cliaddr,
                    len);
        printf("Hello message sent.\n");

        return 0;
}
```

UDP-Client:

```
#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
#define MAXLINE 1024

// Driver code
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;
}
```

Write a UDP client–server program to convert lowercase letters to uppercase letters.

UDP –Server

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

int main(){
  int welcomeSocket, newSocket, portNum, clientLen, nBytes;
  char buffer[1024];
  struct sockaddr_in serverAddr;
  struct sockaddr_storage serverStorage;
  socklen_t addr_size;
  int i;

  welcomeSocket = socket(PF_INET, SOCK_STREAM, 0);

  portNum = 7891;

  serverAddr.sin_family = AF_INET;
  serverAddr.sin_port = htons(portNum);
  serverAddr.sin_addr.s_addr = inet_addr("127.0.0.1");
  memset(serverAddr.sin_zero, '\0', sizeof serverAddr.sin_zero);

  bind(welcomeSocket, (struct sockaddr *) &serverAddr, sizeof(serverAddr));

  if(listen(welcomeSocket,5)==0)
    printf("Listening\n");
  else
    printf("Error\n");

  addr_size = sizeof serverStorage;

  /*loop to keep accepting new connections*/
  while(1){
    newSocket = accept(welcomeSocket, (struct sockaddr *) &serverStorage, &addr_size);
    /*fork a child process to handle the new connection*/
    if(!fork()){
      nBytes = 1;
      /*loop while connection is live*/
      while(nBytes!=0){
        nBytes = recv(newSocket,buffer,1024,0);

        for (i=0;i<nBytes-1;i++){
          buffer[i] = toupper(buffer[i]);
        }

        send(newSocket,buffer,nBytes,0);
      }
      close(newSocket);
    exit(0);
   }
   /*if parent, close the socket and go back to listening new requests*/
   else{
    close(newSocket);
   }
  }

  return 0;
}
```

UDP –Client

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

int main(){
  int clientSocket, portNum, nBytes;
  char buffer[1024];
  struct sockaddr_in serverAddr;
  socklen_t addr_size;

  clientSocket = socket(PF_INET, SOCK_STREAM, 0);

  portNum = 7891;

  serverAddr.sin_family = AF_INET;
  serverAddr.sin_port = htons(portNum);
  serverAddr.sin_addr.s_addr = inet_addr("127.0.0.1");
  memset(serverAddr.sin_zero, '\0', sizeof serverAddr.sin_zero);

  addr_size = sizeof serverAddr;
  connect(clientSocket, (struct sockaddr *) &serverAddr, addr_size);

  while(1){
    printf("Type a sentence to send to server:\n");
    fgets(buffer,1024,stdin);
    printf("You typed: %s",buffer);

    nBytes = strlen(buffer) + 1;

    send(clientSocket,buffer,nBytes,0);

    recv(clientSocket, buffer, 1024, 0);

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

  return 0;
}
```