the domain **AF_INET** is used. The next field type has the value **SOCK_DGRAM**. It supports datagrams (connectionless, unreliable messages of a fixed maximum length). The protocol field specifies the protocol used. We always use 0. If the socket function call is successful, a socket descriptor is returned. Otherwise -1 is returned. The header files necessary for this function call are sys/types.h and s**ys/socket.h.**

2. Filling the fields of the server address structure.

The socket address structure is of type struct sockaddr_in.

```
struct sockaddr_in {

u_short sin_family;
u_short sin_port;
struct in_addr sin_addr;
char sin_zero[8]; /*unused, always zero*/
};
struct in_addr {

u_long s_addr;

};
```

The fields of the socket address structure are

sin_family which in our case is **AF_INET**
sin_port which is the port number where socket binds
sin_addr is used to store the IP address of the server machine and is of type struct **in_addr**

The header file that is to be used is **netinet/in.h**

The value for servaddr.sin_addr is assigned using the following function

inet_pton(AF_INET, "IP_Address", &servaddr.sin_addr);

The binary value of the dotted decimal IP address is stored in the field when the function returns.

3. Binding of a port to the socket in the case of server

This call is used to specify for a socket the protocol port number where it will wait for messages. A call to bind is optional in the case of client and compulsory on the server side.

int bind(int sd, struct sockaddr* addr, int addrlen);

The first field is the socket descriptor. The second is a pointer to the address structure of this socket. The third field is the length in bytes of the size of the structure referenced by **addr**. The header files are **sys/types.h** and s**ys/socket.h.** This function call returns an integer, which is 0 for success and -1 for failure.

4. Receiving data

ssize_t recvfrom(int s, void * buf, size_t len, int flags, struct sockaddr * from, socklen_t * fromlen);

The **recvfrom** calls are used to receive messages from a socket, and may be used to receive data on a socket whether or not it is connection oriented. The first parameter s is the socket descriptor to read from. The second parameter buf is the buffer to read information into. The third parameter len is the maximum length of the buffer. The fourth parameter is flag. It is set to zero. The fifth parameter from is a pointer to **struct sockaddr** variable that will be filled with the IP address and port of the originating machine. The sixth parameter fromlen is a pointer to a **local int** variable that should be initialized to **sizeof(struct sockaddr).** When the function returns, the integer variable that fromlen points to will contain the actual number of bytes that is contained in the socket address structure. The header files required are **sys/types.h** and **sys/socket.h.** When the function returns, the number of bytes received is returned or -1 if there is an error.

5. Sending data

sendto- sends a message from a socket

ssize_t sendto(int s, const void * buf, size_t len, int flags, const struct sockaddr * to, socklen_t tolen);

The first parameter s is the socket descriptor of the sending socket. The second parameter buf is the array which stores data that is to be sent. The third parameter len is the length of that data in bytes. The fourth parameter is the flag parameter. It is set to zero. The fifth parameter to points to a variable that contains the destination IP address and port. The sixth parameter tolen is set to **sizeof(struct sockaddr**). This function returns the number of bytes actually sent or -1 on error. The header files used are **sys/types.h** and **sys/socket.h.**

Algorithm

Client

1. Create socket
2. Read the matrices from the standard input and send it to server using socket
3. Read product matrix from the socket and display it on the standard output
4. Close the socket

Server

1. Create socket
2. bind IP address and port number to the socket
3. Read the matrices socket from the client using socket
4. Find product of matrices
5. Send the product matrix to the client using socket
6. close the socket

Client program

```
#include<stdio.h>
#include<string.h>
#include<sys/socket.h>
#include<sys/types.h>
#include<netinet/in.h>
#include<arpa/inet.h>
#include<fcntl.h>
#include<stdlib.h>
main(int argc, char * argv[])
{
int i,j,n;
int sock_fd;
struct sockaddr_in servaddr;
int matrix_1[10][10], matrix_2[10][10], matrix_product[10][10];
int size[2][2];
int num_rows_1, num_cols_1, num_rows_2, num_cols_2;
if(argc != 3)
{
fprintf(stderr, "Usage: ./client IPaddress_of_server port\n");
exit(1);
}
printf("Enter the number of rows of first matrix\n");
```

```
scanf("%d", &num_rows_1);
printf("Enter the number of columns of first matrix\n");
scanf("%d", &num_cols_1);
printf("Enter the values row by row one on each line\n" );
for ( i = 0; i < num_rows_1; i++)
for( j=0; j<num_cols_1; j++)
{
scanf("%d", &matrix_1[i][j]);
}
size[0][0] = num_rows_1;
size[0][1] = num_cols_1;
printf("Enter the number of rows of second matrix\n");
scanf("%d", &num_rows_2);
printf("Enter the number of columns of second matrix\n");
scanf("%d", &num_cols_2);
if( num_cols_1 != num_rows_2)
{
printf("MATRICES CANNOT BE MULTIPLIED\n");
exit(1);
}
printf("Enter the values row by row one on each line\n");
for (i = 0; i < num_rows_2; i++)
for(j=0; j<num_cols_2; j++)
{
scanf("%d", &matrix_2[i][j]);
}
size[1][0] = num_rows_2;
size[1][1] = num_cols_2;
if((sock_fd = socket(AF_INET, SOCK_DGRAM, 0)) < 0)
{
printf("Cannot create socket\n");
exit(1);
}
bzero((char*)&servaddr, sizeof(servaddr));
servaddr.sin_family = AF_INET;
servaddr.sin_port = htons(atoi(argv[2]));
inet_pton(AF_INET, argv[1], &servaddr.sin_addr);

// SENDING MATRIX WITH SIZES OF MATRICES 1 AND 2
n = sendto(sock_fd, size, sizeof(size),0, (struct sockaddr*)&servaddr, sizeof(servaddr));
```