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));
```

```
if( n < 0)
{
perror("error in matrix 1 sending");
exit(1);
}
// SENDING MATRIX 1
n = sendto(sock_fd, matrix_1, sizeof(matrix_1),0, (struct sockaddr*)&servaddr, sizeof(servaddr));
if( n < 0)
{
perror("error in matrix 1 sending");
exit(1);
}
// SENDING MATRIX 2
n = sendto(sock_fd, matrix_2, sizeof(matrix_2),0, (struct sockaddr*)&servaddr, sizeof(servaddr));
if( n < 0)
{
perror("error in matrix 2 sending");
exit(1);
}
if((n=recvfrom(sock_fd, matrix_product, sizeof(matrix_product),0, NULL, NULL)) == -1)
{
perror("read error from server:");
exit(1);
}
printf("\n\nTHE PRODUCT OF MATRICES IS \n\n\n");
for( i=0; i < num_rows_1; i++)
{
for( j=0; j<num_cols_2; j++)
{
printf("%d ",matrix_product[i][j]);
}
printf("\n");
}
close(sock_fd);
}
```

Server 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 n;
int sock_fd;
int i,j,k;
int row_1, row_2, col_1, col_2;
struct sockaddr_in servaddr, cliaddr;
int len = sizeof(cliaddr);
int matrix_1[10][10], matrix_2[10][10], matrix_product[10][10];
int size[2][2];
if(argc != 2)
{
fprintf(stderr, "Usage: ./server port\n");
exit(1);
}

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[1]));
servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
if(bind(sock_fd, (struct sockaddr*)&servaddr, sizeof(servaddr)) < 0)
{
perror("bind failed:");
exit(1);
}
// MATRICES RECEIVE
```