

# **EXPERIMENT NO.-1**

## **Objective:**

Take a decimal number from user. Convert it to different bases (e.g.: 2,8,16 etc.) and send those values to message queue. Write three separate programs to read and display the binary, octal and hex value from the message queue distinctively.

## **Program Code:**

### *sender.c*

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <unistd.h>
#include <sys/msg.h>

#define MAX_TEXT 512

struct my_msg_st{
    long int my_msg_type;
    char some_text[MAX_TEXT];
};

long long int decimalToBinary(int num){
    long long int bin = 0;
    int i = 1;
    while (num != 0){
        int rem = num % 2;
        num /= 2;
        bin += rem * i;
        i *= 10;
    }
    return bin;
}

long long int decimalToOctal(int num){
    long long int octal = 0;
    int i = 1;
    while (num != 0){
        int rem = num % 8;
```

```

    num /= 8;

    octal += rem * i;

    i *= 10;
}

return octal;
}

int main(){
    int running = 1;

    struct my_msg_st some_data_bin;
    struct my_msg_st some_data_oct;
    struct my_msg_st some_data_hex;
    int msgid;
    char buffer[BUFSIZ];

    msgid = msgget((key_t)1234, 0666 | IPC_CREAT);
    if (msgid == -1){
        fprintf(stderr, "msgget failed with error: %d\n", errno);
        exit(EXIT_FAILURE);
    }

    while (running){
        printf("Enter The Decimal Number : ");
        fgets(buffer, BUFSIZ, stdin);

        some_data_bin.my_msg_type = 2;
        some_data_oct.my_msg_type = 8;
        some_data_hex.my_msg_type = 16;

        if (strncmp(buffer, "end", 3) != 0){
            int num = atoi(buffer);

            sprintf(some_data_bin.some_text, "%lld", decimalToBinary(num));

            num = atoi(buffer);
            sprintf(some_data_oct.some_text, "%lld", decimalToOctal(num));

            int quotient, remainder;
            char str[100] = "";

            quotient = atoi(buffer);
            int j = 0;

            if (quotient == 0)

```

```

        strcpy(some_data_hex.some_text, "0");
    else{
        while (quotient != 0){
            remainder = quotient % 16;
            if (remainder < 10)
                str[j++] = 48 + remainder;
            else
                str[j++] = 55 + remainder;
            quotient = quotient / 16;
        }
        int k = 0;
        char hexadecimalnum[100] = "";
        for (int l = j - 1; l >= 0; l--){
            hexadecimalnum[k++] = str[l];
        }
        strcpy(some_data_hex.some_text, hexadecimalnum);
    }
}

else{
    strcpy(some_data_hex.some_text, buffer);
    strcpy(some_data_bin.some_text, buffer);
    strcpy(some_data_oct.some_text, buffer);
}

if (msgsnd(msgid, (void *)&some_data_bin, MAX_TEXT, 0) == -1){
    fprintf(stderr, "msgsnd failed\n");
    exit(EXIT_FAILURE);
}

if (msgsnd(msgid, (void *)&some_data_oct, MAX_TEXT, 0) == -1){
    fprintf(stderr, "msgsnd failed\n");
    exit(EXIT_FAILURE);
}

if (msgsnd(msgid, (void *)&some_data_hex, MAX_TEXT, 0) == -1){
    fprintf(stderr, "msgsnd failed\n");
    exit(EXIT_FAILURE);
}

```

```

    }

    if (strcmp(buffer, "end", 3) == 0){
        running = 0;
    }
}

exit(EXIT_SUCCESS);
}

```

---

### ***binaryreceiver.c***

```

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <unistd.h>
#include <sys/msg.h>

struct my_msg_st{
    long int my_msg_type;
    char some_text[BUFSIZ];
};

int main(){
    int running = 1;
    int msgid;
    struct my_msg_st some_data;
    long int msg_to_receive = 2;
    msgid = msgget((key_t)1234, 0666 | IPC_CREAT);
    if (msgid == -1){
        fprintf(stderr, "msgget failed with error: %d\n", errno);
        exit(EXIT_FAILURE);
    }
    while (running){
        if (msgrcv(msgid, (void *)&some_data, BUFSIZ, msg_to_receive, 0) == -1){
            fprintf(stderr, "msgrcv failed with error: %d\n", errno);
            exit(EXIT_FAILURE);
        }
        if (strcmp(some_data.some_text, "end", 3) == 0){
            running = 0;

```

```

    printf("Program Terminated\n");

    break;

}

printf("Decimal to Binary : %s\n", some_data.some_text);

}

exit(EXIT_SUCCESS);

}

```

---

### *octalreceiver.c*

```

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <unistd.h>
#include <sys/msg.h>

struct my_msg_st{

    long int my_msg_type;

    char some_text[BUFSIZ];

};

int main(){

    int running = 1;

    int msgid;

    struct my_msg_st some_data;

    long int msg_to_receive = 8;

    msgid = msgget((key_t)1234, 0666 | IPC_CREAT);

    if (msgid == -1){

        fprintf(stderr, "msgget failed with error: %d\n", errno);

        exit(EXIT_FAILURE);

    }

    while (running){

        if (msgrcv(msgid, (void *)&some_data, BUFSIZ, msg_to_receive, 0) == -1){

            fprintf(stderr, "msgrcv failed with error: %d\n", errno);

            exit(EXIT_FAILURE);

        }

        if (strcmp(some_data.some_text, "end", 3) == 0){

```

```

        running = 0;

        printf("Program Terminated\n");

        break;

    }

    printf("Decimal to Octal : %s\n", some_data.some_text);

}

exit(EXIT_SUCCESS);

}

```

---

### *hexadecimalreceiver.c*

```

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <unistd.h>
#include <sys/msg.h>

struct my_msg_st{

    long int my_msg_type;

    char some_text[BUFSIZ];

};

int main(){

    int running = 1;

    int msgid;

    struct my_msg_st some_data;

    long int msg_to_receive = 16;

    msgid = msgget((key_t)1234, 0666 | IPC_CREAT);

    if (msgid == -1){

        fprintf(stderr, "msgget failed with error: %d\n", errno);

        exit(EXIT_FAILURE);

    }

    while (running){

        if (msgrcv(msgid, (void *)&some_data, BUFSIZ, msg_to_receive, 0) == -1){

            fprintf(stderr, "msgrcv failed with error: %d\n", errno);

            exit(EXIT_FAILURE);

        }

    }

}

```

```

if (strcmp(some_data.some_text, "end", 3) == 0){
    running = 0;
    printf("Program Terminated\n");
    break;
}

printf("Decimal to Hexadecimal : %s\n", some_data.some_text);
}

if (msgctl(msgid, IPC_RMID, 0) == -1){
    fprintf(stderr, "msgctl(IPC_RMID) failed\n");
    exit(EXIT_FAILURE);
}

exit(EXIT_SUCCESS);
}

```

---

## Screenshot of the Output:

The image displays four terminal windows arranged in a 2x2 grid, showing the output of a program that converts decimal numbers to binary, octal, and hexadecimal. Each window has a title bar 'Exp1a/' and a close button 'x'.

- Top Left Window:** Shows the execution of `./sender.out`. It prompts the user to enter decimal numbers: 10, 0, 20, 9, 171, 60, and 'end'. The program terminates after the 'end' input.
- Top Right Window:** Shows the execution of `./binaryreceiver.out`. It displays the binary representations of the numbers: 1010, 0, 10100, 1001, 10101011, and 111100. The program terminates.
- Bottom Left Window:** Shows the execution of `./octalreceiver.out`. It displays the octal representations of the numbers: 12, 0, 24, 11, 253, and 74. The program terminates.
- Bottom Right Window:** Shows the execution of `./hexadecimalreceiver.out`. It displays the hexadecimal representations of the numbers: A, 0, 14, 9, AB, and 3C. The program terminates.

## **EXPERIMENT NO.- 2**

### **Objective:**

Write C Programs to implement a simple client-server application. A client will send N integers to the server, which will sort the integers and send back to client. The client will print the result. Use Unix File socket for communication.

### **Program Code:**

*client.c*

```
#include <sys/types.h>
#include <sys/socket.h>
#include <stdio.h>
#include <sys/un.h>
#include <unistd.h>
#include <stdlib.h>

int main(){
    int sockfd;
    int len;
    struct sockaddr_un address;
    int result;
    int arr[5], i;

    sockfd = socket(AF_UNIX, SOCK_STREAM, 0);
    address.sun_family = AF_UNIX;
    strcpy(address.sun_path, "server_socket");
    len = sizeof(address);
    result = connect(sockfd, (struct sockaddr *)&address, len);
    if (result == -1){
        perror("oops:client1 ");
        exit(1);
    }

    printf("Enter 5 elements\n");
    for (i = 0; i < 5; i++)
        scanf("%d", &arr[i]);

    write(sockfd, arr, sizeof(arr));
    read(sockfd, arr, sizeof(arr));

    printf("Sorted Array From Server:\n");
    for (i = 0; i < 5; i++)
        printf("%d ", arr[i]);

    close(sockfd);
    exit(0);
}
```

---



## *server.c*

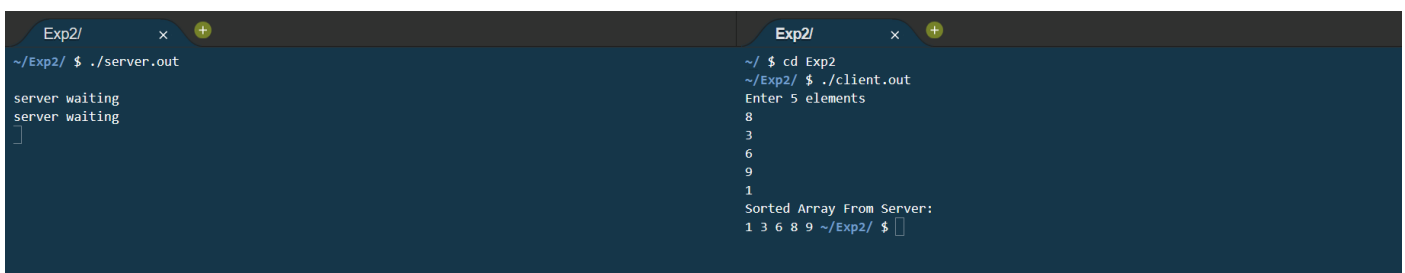
```
#include <sys/types.h>
#include <sys/socket.h>
#include <stdio.h>
#include <sys/un.h>
#include <unistd.h>
#include <stdlib.h>

int main(){
    int server_sockfd, client_sockfd;
    int server_len, client_len;
    struct sockaddr_un server_address;
    struct sockaddr_un client_address;
    int arr[5], i, j, t;

    unlink("server_socket");
    server_sockfd = socket(AF_UNIX, SOCK_STREAM, 0);
    server_address.sun_family = AF_UNIX;
    strcpy(server_address.sun_path, "server_socket");
    server_len = sizeof(server_address);
    bind(server_sockfd, (struct sockaddr *)&server_address, server_len);
    listen(server_sockfd, 5);
    while (1){
        printf("server waiting\n");
        client_len = sizeof(client_address);
        client_sockfd = accept(server_sockfd, (struct sockaddr *)&client_address, &client_len);
        read(client_sockfd, arr, sizeof(arr));
        for (i = 1; i < 5; i++){
            for (j = 0; j < 5 - i; j++){
                if (arr[j] > arr[j + 1]){
                    t = arr[j];
                    arr[j] = arr[j + 1];
                    arr[j + 1] = t;
                }
            }
        }
        write(client_sockfd, arr, sizeof(arr));
        close(client_sockfd);
    }
}
```

---

## Screenshot of the Output:



The screenshot shows two terminal windows side-by-side. The left window, titled 'Exp2/', shows the output of the server program: 'server waiting' followed by a blank line. The right window, also titled 'Exp2/', shows the output of the client program: 'cd Exp2', './client.out', 'Enter 5 elements', followed by the input '8 3 6 9 1', and then the output 'Sorted Array From Server: 1 3 6 8 9'.

```
Exp2/ x +
~/Exp2/ $ ./server.out
server waiting
server waiting
[]

Exp2/ x +
~/ $ cd Exp2
~/Exp2/ $ ./client.out
Enter 5 elements
8
3
6
9
1
Sorted Array From Server:
1 3 6 8 9 ~/Exp2/ $ []
```

# **EXPERIMENT NO.- 3A**

## **Objective:**

Write a C Program to implement TCP socket. The client will take a bit-stream from the user and send it to the server. The server will implement bit stuffing and send the stream back to the client. The client will print it.

## **Program Code:**

### *client.c*

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

int main()
{
    int sockfd;
    int len;
    struct sockaddr_in address;
    int result;
    int st[8] = {0};
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    address.sin_family = AF_INET;
    address.sin_addr.s_addr = inet_addr("127.0.0.1");
    address.sin_port = 9734;
    len = sizeof(address);
    result = connect(sockfd, (struct sockaddr *)&address, len);
    if (result == -1)
    {
        perror("oops:client1");
        exit(1);
    }
    printf("Frame Size is Eight. Enter a bit sequence: ");
    for (int i = 0; i < 8; i++)
    {
        scanf("%d", &st[i]);
    }
    write(sockfd, st, sizeof(st));
    read(sockfd, st, sizeof(st));
    printf("Received Sequence: \n");
    for (int i = 0; i < 8; i++)
    {
        printf("%d", st[i]);
    }
    printf("\n\n");
}
```

```
    exit(0);
}
```

---

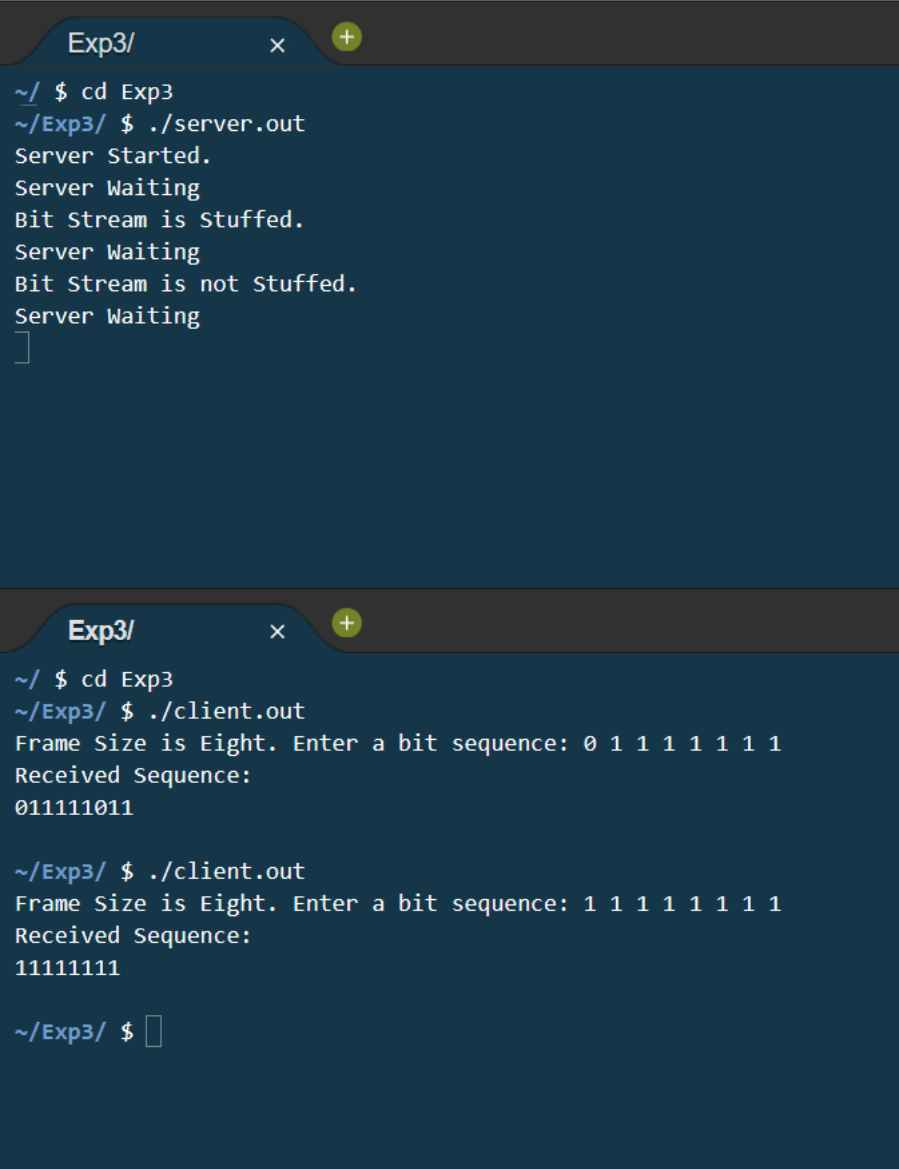
### *server.c*

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

int main()
{
    int client_sockfd, server_sockfd;
    int server_len, client_len;
    struct sockaddr_in server_address;
    struct sockaddr_in client_address;
    server_sockfd = socket(AF_INET, SOCK_STREAM, 0);
    server_address.sin_family = AF_INET;
    server_address.sin_addr.s_addr = inet_addr("127.0.0.1");
    server_address.sin_port = 9734;
    server_len = sizeof(server_address);
    bind(server_sockfd, (struct sockaddr *)&server_address, server_len);
    listen(server_sockfd, 5);
    printf("Server Started.\n");
    while (1)
    {
        int st[8];
        int i = 0;
        printf("Server Waiting\n");
        client_len = sizeof(client_address);
        client_sockfd = accept(server_sockfd, (struct sockaddr *)&client_address, &client_len);
        read(client_sockfd, st, sizeof(st));
        if ((st[i] == 0) && (st[i + 1] == 1) && (st[i + 2] == 1) && (st[i + 3] == 1) && (st[i + 4] == 1) && (st[i
+ 5] == 1))
        {
            st[i + 5] = 10;
            printf("Bit Stream is Stuffed.\n");
        }
        else
        {
            printf("Bit Stream is not Stuffed.\n");
        }
        write(client_sockfd, st, sizeof(st));
    }
}
```

---

## Screenshot of the Output:



The image shows two terminal windows, both titled 'Exp3/'. The top window shows the execution of a server program, and the bottom window shows the execution of a client program.

```
~/ $ cd Exp3
~/Exp3/ $ ./server.out
Server Started.
Server Waiting
Bit Stream is Stuffed.
Server Waiting
Bit Stream is not Stuffed.
Server Waiting

```

```
~/ $ cd Exp3
~/Exp3/ $ ./client.out
Frame Size is Eight. Enter a bit sequence: 0 1 1 1 1 1 1 1
Received Sequence:
011111011

~/Exp3/ $ ./client.out
Frame Size is Eight. Enter a bit sequence: 1 1 1 1 1 1 1 1
Received Sequence:
11111111

~/Exp3/ $ 
```

# **EXPERIMENT NO.- 3B**

## **Objective:**

Create a multi-client TCP server. The client will take a dataword and divisor from the user and send them to the server. The server will find out the codeword using CRC and send it to the client.

## **Program Code:**

### *client.c*

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#define MAXSIZE 15

int main()
{
    char val[MAXSIZE];
    int sockfd, result, len;
    struct sockaddr_in address;

    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    address.sin_family = AF_INET;
    address.sin_addr.s_addr = inet_addr("127.0.0.1");
    address.sin_port = 9734;
    len = sizeof(address);
    result = connect(sockfd, (struct sockaddr *)&address, len);
    if (result == -1)
    {
        perror("Client Error!");
        exit(1);
    }
    printf("Enter the dataword: ");
    fgets(val, MAXSIZE, stdin);
    write(sockfd, val, sizeof(val));

    printf("Enter the divisor: ");
    fgets(val, MAXSIZE, stdin);
    write(sockfd, val, sizeof(val));

    read(sockfd, val, sizeof(val));
    printf("Codeword from Server = %s", val);
    close(sockfd);
    return 0;
```

```
}
```

---

### *server.c*

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <pthread.h>
#include <math.h>
#define MAXCLIENT 3
#define MAXSIZE 15

void *serve(void *arg)
{
    int client_sockfd, client_len;
    struct sockaddr_in client_address;
    char data[MAXSIZE], tmp[MAXSIZE];
    char val[MAXSIZE], div[MAXSIZE];
    int i, j, end, start, data_len = 0, div_len = 0;
    int socket = *((int *)arg);
    client_len = sizeof(client_address);
    client_sockfd = accept(socket, (struct sockaddr *)&client_address, &client_len);
    read(client_sockfd, val, sizeof(val));
    i = 0;
    while (val[i] != 10)
    {
        data_len++;
        i++;
    }
    strcpy(data, val);
    strcpy(tmp, val);
    printf("Dataword Received!\n");
    read(client_sockfd, val, sizeof(val));
    i = 0;
    while (val[i] != 10)
    {
        div_len++;
        i++;
    }
    strcpy(div, val);
    i = data_len;
    data_len = data_len + div_len - 1;
    while (i < data_len)
    {
        data[i] = '0';
        i++;
    }
}
```

```

}
data[i] = 10;
start = 0;
end = div_len - 1;
while (end < data_len)
{
    if (data[start] == '1')
    {
        for (i = start, j = 0; i <= end, j < div_len; i++, j++)
        {
            if (data[i] == div[j])
                data[i] = '0';
            else
                data[i] = '1';
        }
    }
    start++;
    end++;
}
i = 0;
end = div_len - 1;
while (end)
{
    tmp[data_len - 1 - i] = data[data_len - 1 - i];
    end--;
    i++;
}
tmp[data_len] = 10;
write(client_sockfd, tmp, sizeof(tmp));
return NULL;
}

```

```

int main()
{
    int server_sockfd, server_len, temp;
    struct sockaddr_in server_address;
    pthread_t th[MAXCLIENT];
    server_sockfd = socket(AF_INET, SOCK_STREAM, 0);
    server_address.sin_family = AF_INET;
    server_address.sin_addr.s_addr = inet_addr("127.0.0.1");
    server_address.sin_port = 9734;
    server_len = sizeof(server_address);
    bind(server_sockfd, (struct sockaddr *)&server_address, server_len);
    listen(server_sockfd, MAXCLIENT);
    printf("Server Started\n");
    while (1)
    {
        temp = 0;
        while (temp < MAXCLIENT)
        {
            int *pserver = malloc(sizeof(int));

```

```

    *pserver = server_sockfd;
    pthread_create(&th[temp], NULL, serve, (void *)pserver);
    temp++;
}
temp = 0;
while (temp < MAXCLIENT)
{
    printf("Server Waiting\n");
    pthread_join(th[temp], NULL);
    temp++;
}
}
return 0;
}

```

## Screenshot of the Output:

The screenshot displays four terminal windows arranged in a 2x2 grid, all with the title bar 'Exp3b/'.

- Top-left window:** Shows the server's execution. It starts with `~/ $ cd Exp3b` and `~/Exp3b/ $ ./server.out`. The output includes 'Server Started', followed by a loop of 'Server Waiting' and 'Dataword Received!' messages.
- Top-right window:** Shows the first client's execution. It starts with `~/ $ cd Exp3b` and `~/Exp3b/ $ ./client1.out`. The user enters '1010101010' for the dataword and '11001' for the divisor. The output is 'Codeword from Server = 10101010100010'.
- Bottom-left window:** Shows the second client's execution. It starts with `~/ $ cd Exp3b` and `~/Exp3b/ $ ./client2.out`. The user enters '100100' for the dataword and '1101' for the divisor. The output is 'Codeword from Server = 100100001'.
- Bottom-right window:** Shows the third client's execution. It starts with `~/ $ cd Exp3b` and `~/Exp3b/ $ ./client3.out`. The user enters '100101' for the dataword and '1101' for the divisor. The output is 'Codeword from Server = 100101100'.



# **EXPERIMENT NO.- 4**

## **Objective:**

Take a IPV4 address as input. Write a C program to check in which class does it belong. Also print special comment for network ID and broadcast ID and Default Mask ID.

## **Program Code:**

*ipv4.c*

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

char findClass(char str[])
{
    char arr[4];
    int i = 0;
    while (str[i] != '.')
    {
        arr[i] = str[i];
        i++;
    }
    i--;

    int ip = 0, j = 1;
    while (i >= 0)
    {
        ip = ip + (str[i] - '0') * j;
        j = j * 10;
        i--;
    }

    if (ip >= 1 && ip <= 126)
        return 'A';

    else if (ip >= 128 && ip <= 191)
        return 'B';

    else if (ip >= 192 && ip <= 223)
        return 'C';

    else if (ip >= 224 && ip <= 239)
        return 'D';

    else
        return 'E';
}

void separate(char str[], char ipClass)
{
    char network[12];
```

```

for (int k = 0; k < 12; k++)
    network[k] = '\0';

if (ipClass == 'A')
{
    int i = 0, j = 0;
    while (str[j] != '.')
        network[i++] = str[j++];
    printf("Network ID is %s.0.0.0\n", network);
    printf("Broadcast ID: %s.255.255.255\n", network);
    printf("Default Mask ID is 255.0.0.0\n");
}

else if (ipClass == 'B')
{
    int i = 0, j = 0, dotCount = 0;
    while (dotCount < 2)
    {
        network[i++] = str[j++];
        if (str[j] == '.')
            dotCount++;
    }

    printf("Network ID is %s.0.0\n", network);
    printf("Broadcast ID: %s.255.255\n", network);
    printf("Default Mask ID is 255.255.0.0\n");
}

else if (ipClass == 'C')
{
    int i = 0, j = 0, dotCount = 0;
    while (dotCount < 3)
    {
        network[i++] = str[j++];
        if (str[j] == '.')
            dotCount++;
    }

    printf("Network ID is %s.0\n", network);
    printf("Broadcast ID: %s.255\n", network);
    printf("Default Mask ID is 255.255.255.0\n");
}

else
    printf("In this Class, IP address is not divided into Network and Host ID\n");
}

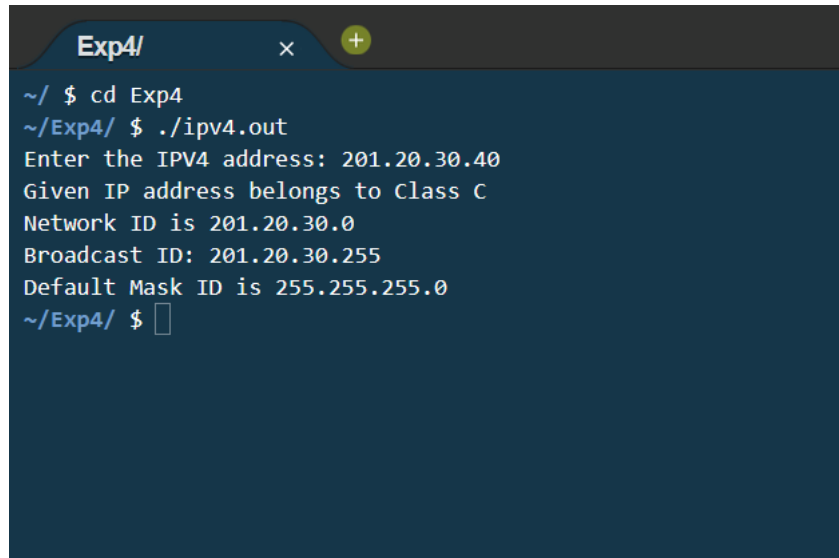
int main()
{
    char str[15];
    printf("Enter the IPV4 address: ");

```

```
fgets(str, 15, stdin);
char ipClass = findClass(str);
printf("Given IP address belongs to Class %c\n", ipClass);
separate(str, ipClass);
return 0;
}
```

---

### Screenshot of the Output:



```
Exp4/ x +
~/ $ cd Exp4
~/Exp4/ $ ./ipv4.out
Enter the IPV4 address: 201.20.30.40
Given IP address belongs to Class c
Network ID is 201.20.30.0
Broadcast ID: 201.20.30.255
Default Mask ID is 255.255.255.0
~/Exp4/ $
```

# **EXPERIMENT NO.- 5**

## **Objective:**

Write client-server programs using UDP socket. The client will take a data word from the user and send it to the server. The server will find the codeword (use Hamming code error correction technique) and send it back to the client.

## **Program Code:**

### *client.c*

```
#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;
    struct sockaddr_in servaddr;
    if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0)
    {
        perror("socket creation failed");
        exit(EXIT_FAILURE);
    }
    memset(&servaddr, 0, sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    servaddr.sin_port = htons(PORT);
    servaddr.sin_addr.s_addr = INADDR_ANY;
    int arr[4];
    printf("Enter four data bits: ");
    for (int i = 0; i < 4; i++)
        scanf("%d", &arr[i]);
    sendto(sockfd, &arr, 4 * sizeof(int), MSG_CONFIRM, (const struct sockaddr *)&servaddr,
    sizeof(servaddr));
    printf("Data sent\n");
    int ans[7], len;
    len = sizeof(servaddr);
    recvfrom(sockfd, &ans, 7 * sizeof(int), MSG_WAITALL, (struct sockaddr *)&servaddr, &len);
    printf("Received Dataword: ");
    for (int i = 0; i < 7; i++)
        printf("%d", ans[i]);
    close(sockfd);
    return 0;
}
```

```
}
```

---

---

### *server.c*

```
#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>
#include <math.h>

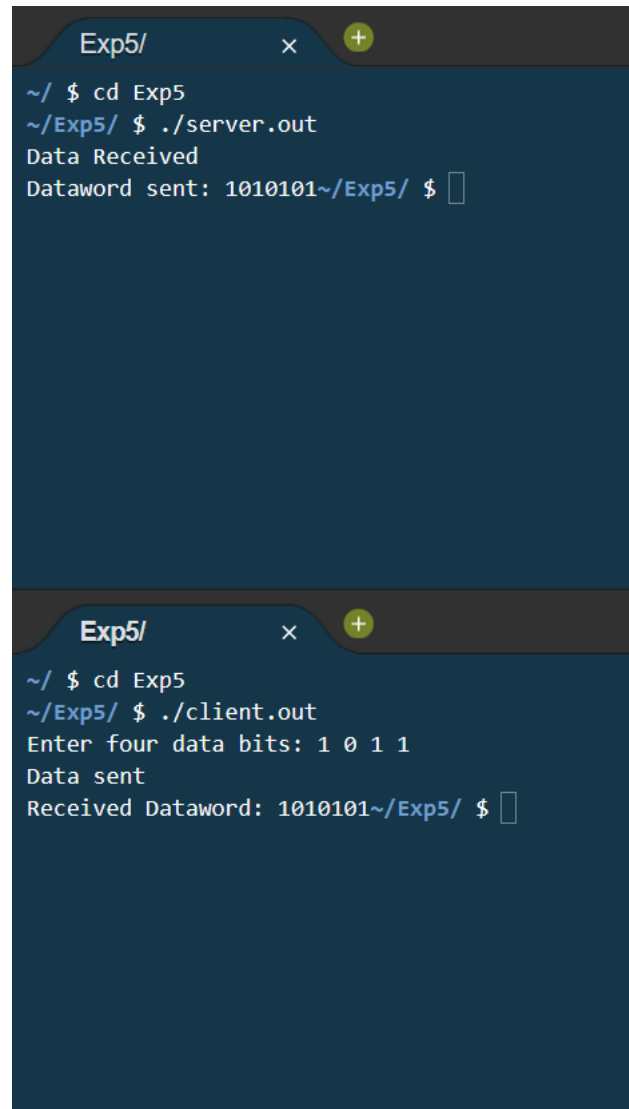
#define PORT 8080

int main()
{
    int sockfd;
    struct sockaddr_in servaddr, cliaddr;
    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));
    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr = INADDR_ANY;
    servaddr.sin_port = htons(PORT);
    if (bind(sockfd, (const struct sockaddr *)&servaddr, sizeof(servaddr)) < 0)
    {
        perror("bind failed");
        exit(EXIT_FAILURE);
    }
    int len;
    len = sizeof(cliaddr);
    int arr[4];
    recvfrom(sockfd, &arr, 4 * sizeof(int), MSG_WAITALL, (struct sockaddr *)&cliaddr, &len);
    printf("Data Received \n");
    int ans[7];
    ans[0] = arr[0];
    ans[1] = arr[1];
    ans[2] = arr[2];
    ans[4] = arr[3];
    ans[3] = ans[0] ^ ans[1] ^ ans[2];
    ans[5] = ans[0] ^ ans[1] ^ ans[4];
    ans[6] = ans[0] ^ ans[2] ^ ans[4];
    sendto(sockfd, &ans, 7 * sizeof(int), MSG_CONFIRM, (const struct sockaddr *)&cliaddr, len);
    printf("Dataword sent: ");
    for (int i = 0; i < 7; i++)
```

```
    printf("%d", ans[i]);  
    return 0;  
}
```

---

## Screenshot of the Output:



The screenshot displays two terminal windows, both titled 'Exp5/'. The top window shows the execution of the server program. The user navigates to the 'Exp5' directory and runs './server.out'. The program outputs 'Data Received' and 'Dataword sent: 1010101'. The bottom window shows the execution of the client program. The user navigates to the 'Exp5' directory and runs './client.out'. The program prompts the user to 'Enter four data bits: 1 0 1 1', outputs 'Data sent', and then displays 'Received Dataword: 1010101'.

```
Exp5/ x +  
~/ $ cd Exp5  
~/Exp5/ $ ./server.out  
Data Received  
Dataword sent: 1010101~/Exp5/ $   
  
Exp5/ x +  
~/ $ cd Exp5  
~/Exp5/ $ ./client.out  
Enter four data bits: 1 0 1 1  
Data sent  
Received Dataword: 1010101~/Exp5/ $   

```

# **EXPERIMENT NO.- 6**

## **Objective:**

Write C programs to implement a simple chat server (single client, single server) by using TCP Socket.

## **Program Code:**

### *client.c*

```
#include <stdio.h>
#include <string.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <unistd.h>
#include <pthread.h>
#include <termios.h>
#include <stdlib.h>
void *recv_thread(void *a);
void *send_thread(void *a);
int cfd;
int main()
{
    struct sockaddr_in cl;
    int n;
    pthread_t snd, rcv;
    cfd = socket(AF_INET, SOCK_STREAM, 0);
    cl.sin_family = AF_INET;
    inet_aton("127.0.0.1", &(cl.sin_addr));
    cl.sin_port = htons(8760);
    connect(cfd, (struct sockaddr *)&cl, sizeof(cl));
    pthread_create(&snd, NULL, send_thread, NULL);
    pthread_create(&rcv, NULL, recv_thread, NULL);
    pthread_join(snd, NULL);
    pthread_join(rcv, NULL);
    close(cfd);
    return 0;
}
void *send_thread(void *a)
{
    int n;
    char str[200];
    while (1)
    {
        fgets(str, 200, stdin);
        write(cfd, (void *)str, sizeof(str));

        if (strncmp(str, "bye", 3) == 0)
            exit(0);
    }
}
```

```

    }
    pthread_exit(NULL);
}
void *recv_thread(void *a)
{
    int n;
    char str[200];
    while (1)
    {
        n = read(cfd, (void *)str, sizeof(str));
        if (n > 0)
        {
            printf("Server: %s", str);
            fflush(stdout);
        }
        if (strncmp(str, "bye", 3) == 0)
            exit(0);
    }
    pthread_exit(NULL);
}

```

---

### ***server.c***

```

#include <stdio.h>
#include <string.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <unistd.h>
#include <pthread.h>
#include <termios.h>
#include <stdlib.h>
void *recv_thread(void *a);
void *send_thread(void *a);
int cfd;
int main()
{
    struct sockaddr_in ser, cl;
    int sfd, len, i, n;
    pthread_t snd, rcv;
    len = sizeof(cl);
    sfd = socket(AF_INET, SOCK_STREAM, 0);
    ser.sin_family = AF_INET;
    inet_aton("127.0.0.1", &(ser.sin_addr));
    ser.sin_port = htons(8760);
    n = bind(sfd, (struct sockaddr *)&ser, sizeof(ser));
    n = listen(sfd, 1);
    cfd = accept(sfd, (struct sockaddr *)&cl, &len);
    pthread_create(&snd, NULL, send_thread, NULL);
    pthread_create(&rcv, NULL, recv_thread, NULL);
}

```



```

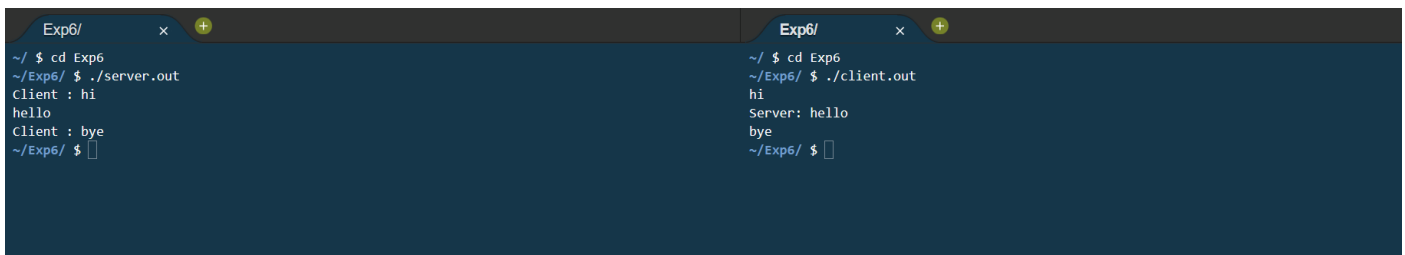
pthread_join(snd, NULL);
pthread_join(rcv, NULL);
close(cfd);
return 0;
}
void *send_thread(void *a)
{
    int n;
    char str[200];
    while (1)
    {
        fgets(str, 200, stdin);
        write(cfd, (void *)str, sizeof(str));
        if (strncmp(str, "bye", 3) == 0)
            exit(0);
    }
    pthread_exit(NULL);
}
void *recv_thread(void *a)
{
    int n;
    char str[200];
    while (1)

    {
        n = read(cfd, (void *)str, sizeof(str));
        if (n > 0)
        {
            printf("Client : %s", str);
            fflush(stdout);
        }
        if (strncmp(str, "bye", 3) == 0)
            exit(0);
    }
    pthread_exit(NULL);
}

```

---

## Screenshot of the Output:



```

Exp6/ x +
~/ $ cd Exp6
~/Exp6/ $ ./server.out
Client : hi
hello
Client : bye
~/Exp6/ $

Exp6/ x +
~/ $ cd Exp6
~/Exp6/ $ ./client.out
hi
Server: hello
bye
~/Exp6/ $

```

# **EXPERIMENT NO.-7**

## **Objective:**

Write C programs to implement a multi-client chat server by using TCP Socket.

## **Program Code:**

*client.c*

```
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>
#define MAX_SIZE 50
char msg[MAX_SIZE];
void *recvmg(void *my_sock)
{
    int sock = *((int *)my_sock);
    int len;
    while ((len = recv(sock, msg, MAX_SIZE, 0)) > 0)
    {
        msg[len] = '\0';
        fputs(msg, stdout);
    }
}
int main(int argc, char *argv[])
{
    pthread_t recvt;
    int sock_desc;
    struct sockaddr_in serv_addr;
    char client_name[MAX_SIZE];
    char recvmg[MAX_SIZE];
    strcpy(client_name, argv[1]);
    if ((sock_desc = socket(AF_INET, SOCK_STREAM, 0)) < 0)
        printf("Failed creating socket\n");
    bzero((char *)&serv_addr, sizeof(serv_addr));
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr = inet_addr("127.0.0.1");
    serv_addr.sin_port = htons(3000);

    if (connect(sock_desc, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0)
    {
        printf("Failed to connect to server\n");
        return -1;
    }
    pthread_create(&recvt, NULL, (void *)recvmg, &sock_desc);
```

```

char data[MAX_SIZE];
char sendMSG[MAX_SIZE];
while (1)
{
    fgets(data, MAX_SIZE, stdin);
    strcpy(sendMSG, client_name);
    strcat(sendMSG, ": ");
    strcat(sendMSG, data);
    write(sock_desc, &sendMSG, sizeof(sendMSG));
}
exit(0);
close(sock_desc);
return 0;
}

```

---

### ***server.c***

```

#include <stdio.h>
#include <sys/socket.h>
#include <string.h>
#include <arpa/inet.h>
#include <pthread.h>
pthread_mutex_t mutex;
int clients[20];
int n = 0;
void sendtoall(char *msg, int curr)
{
    int i;
    pthread_mutex_lock(&mutex);
    for (i = 0; i < n; i++)
    {
        if (clients[i] != curr)
        {
            if (send(clients[i], msg, strlen(msg), 0) < 0)
            {
                printf("sending failure \n");
                continue;
            }
        }
    }
    pthread_mutex_unlock(&mutex);
}
void *recvmg(void *client_sock)
{
    int sock = *((int *)client_sock);
    char msg[500];
    int len;
    while ((len = recv(sock, msg, 500, 0)) > 0)

```

```

    {
        msg[len] = '\0';
        fputs(msg, stdout);
        sendtoall(msg, sock);
    }
}

int main()
{
    struct sockaddr_in ServerIp;
    pthread_t recvt;
    int sock = 0, Client_sock = 0;

    ServerIp.sin_family = AF_INET;
    ServerIp.sin_port = htons(3000);
    ServerIp.sin_addr.s_addr = inet_addr("127.0.0.1");
    sock = socket(AF_INET, SOCK_STREAM, 0);

    if (bind(sock, (struct sockaddr *)&ServerIp, sizeof(ServerIp)) == -1)
        printf("cannot bind, error!! \n");
    else
        printf("Server Started...\n");

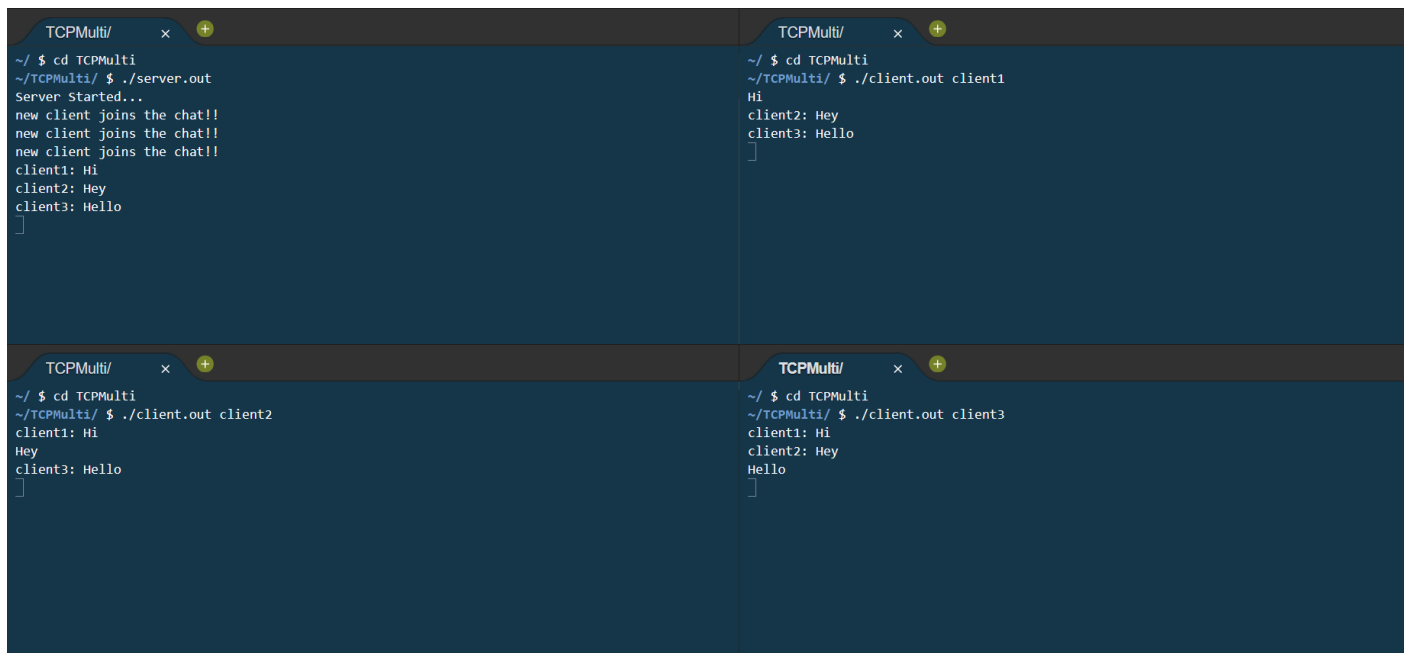
    if (listen(sock, 20) == -1)
        printf("listening failed!! \n");

    while (1)
    {
        if ((Client_sock = accept(sock, (struct sockaddr *)NULL, NULL)) < 0)
        {
            printf("accept failed!!\n");
        }
        else
        {
            printf("new client joins the chat!!\n");
        }
        pthread_mutex_lock(&mutex);
        clients[n] = Client_sock;
        n++;
        pthread_create(&recvt, NULL, (void *)recvmg, &Client_sock);
        pthread_mutex_unlock(&mutex);
    }
    return 0;
}

```

---

## Screenshot of the Output:



The screenshot displays four terminal windows arranged in a 2x2 grid, all titled 'TCPMulti/'. Each window shows the output of a different command or set of commands related to a TCP-based chat application.

**Top-Left Terminal:** Shows the server starting and receiving three client connections. The output is as follows:

```
~/ $ cd TCPMulti
~/TCPMulti/ $ ./server.out
Server Started...
new client joins the chat!!
new client joins the chat!!
new client joins the chat!!
client1: Hi
client2: Hey
client3: Hello
]
```

**Top-Right Terminal:** Shows the output of the client1.out script, which sends 'Hi' to the server and receives 'client2: Hey' and 'client3: Hello' in response. The output is:

```
~/ $ cd TCPMulti
~/TCPMulti/ $ ./client.out client1
Hi
client2: Hey
client3: Hello
]
```

**Bottom-Left Terminal:** Shows the output of the client2.out script, which sends 'Hi' to the server and receives 'Hey' and 'client3: Hello' in response. The output is:

```
~/ $ cd TCPMulti
~/TCPMulti/ $ ./client.out client2
client1: Hi
Hey
client3: Hello
]
```

**Bottom-Right Terminal:** Shows the output of the client3.out script, which sends 'Hi' to the server and receives 'client2: Hey' and 'Hello' in response. The output is:

```
~/ $ cd TCPMulti
~/TCPMulti/ $ ./client.out client3
client1: Hi
client2: Hey
Hello
]
```

# **EXPERIMENT NO.-8**

## **Objective:**

Write C programs to implement a simple chat server (single client, single server) by using UDP Socket.

## **Program Code:**

### *client.c*

```
#include <stdio.h>
#include <string.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <unistd.h>
#include <pthread.h>
#include <termios.h>
#include <stdlib.h>
void *recv_thread(void *a);
void *send_thread(void *a);
int cfd;
int main()
{
    struct sockaddr_in cl;
    int n;
    pthread_t snd, rcv;
    cfd = socket(AF_INET, SOCK_STREAM, 0);
    cl.sin_family = AF_INET;
    inet_aton("127.0.0.1", &(cl.sin_addr));
    cl.sin_port = htons(8760);
    connect(cfd, (struct sockaddr *)&cl, sizeof(cl));
    pthread_create(&snd, NULL, send_thread, NULL);
    pthread_create(&rcv, NULL, recv_thread, NULL);
    pthread_join(snd, NULL);
    pthread_join(rcv, NULL);
    close(cfd);
    return 0;
}
void *send_thread(void *a)
{
    int n;
    char str[200];
    while (1)
    {
        fgets(str, 200, stdin);
        write(cfd, (void *)str, sizeof(str));

        if (strncmp(str, "bye", 3) == 0)
            exit(0);
    }
}
```

```

    }
    pthread_exit(NULL);
}
void *recv_thread(void *a)
{
    int n;
    char str[200];
    while (1)
    {
        n = read(cfd, (void *)str, sizeof(str));
        if (n > 0)
        {
            printf("Server: %s", str);
            fflush(stdout);
        }
        if (strncmp(str, "bye", 3) == 0)
            exit(0);
    }
    pthread_exit(NULL);
}

```

---

### ***server.c***

```

#include <stdio.h>
#include <string.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <unistd.h>
#include <pthread.h>
#include <termios.h>
#include <stdlib.h>
void *recv_thread(void *a);
void *send_thread(void *a);
int cfd;
int main()
{
    struct sockaddr_in ser, cl;
    int sfd, len, i, n;
    pthread_t snd, rcv;
    len = sizeof(cl);
    sfd = socket(AF_INET, SOCK_STREAM, 0);
    ser.sin_family = AF_INET;
    inet_aton("127.0.0.1", &(ser.sin_addr));
    ser.sin_port = htons(8760);
    n = bind(sfd, (struct sockaddr *)&ser, sizeof(ser));
    n = listen(sfd, 1);
    cfd = accept(sfd, (struct sockaddr *)&cl, &len);

```

```

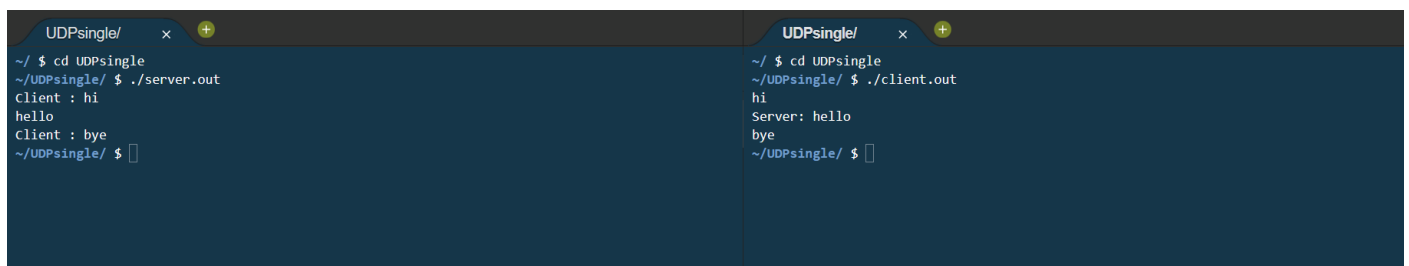
pthread_create(&snd, NULL, send_thread, NULL);
pthread_create(&rcv, NULL, recv_thread, NULL);
pthread_join(snd, NULL);
pthread_join(rcv, NULL);
close(cfd);
return 0;
}
void *send_thread(void *a)
{
    int n;
    char str[200];
    while (1)
    {
        fgets(str, 200, stdin);
        write(cfd, (void *)str, sizeof(str));
        if (strncmp(str, "bye", 3) == 0)
            exit(0);
    }
    pthread_exit(NULL);
}
void *recv_thread(void *a)
{
    int n;
    char str[200];
    while (1)

    {
        n = read(cfd, (void *)str, sizeof(str));
        if (n > 0)
        {
            printf("Client : %s", str);
            fflush(stdout);
        }
        if (strncmp(str, "bye", 3) == 0)
            exit(0);
    }
    pthread_exit(NULL);
}

```

=====

## Screenshot of the Output:





# **EXPERIMENT NO.-9**

## **Objective:**

Write C programs to implement a multi-client chat server by using UDP Socket.

## **Program Code:**

*client.c*

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.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(EXIT_FAILURE);
    }

    memset(&servaddr, 0, sizeof(servaddr));

    servaddr.sin_family = AF_INET;
    servaddr.sin_port = htons(PORT);
    servaddr.sin_addr.s_addr = INADDR_ANY;

    int n;

    socklen_t len;

    while (1)
    {
        printf("Enter message : ");
        fgets(buffer, MAXLINE, stdin);
        sendto(sockfd, (const char *)buffer, strlen(buffer),
            MSG_CONFIRM, (const struct sockaddr *)&servaddr,
            sizeof(servaddr));
        printf("Message sent.\n");
    }
}
```

```

n = recvfrom(sockfd, (char *)buffer, MAXLINE,
             MSG_WAITALL, (struct sockaddr *)&servaddr,
             &len);
buffer[n] = '\0';
printf("Server : %s\n", buffer);

if ((strncmp(buffer, "bye", 3)) == 0)
{
    printf("Client Exit...\n");
    break;
}
}
}

```

=====

### *server.c*

```

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

#define PORT 8080
#define MAXLINE 1024

int main()
{
    int sockfd;
    char buffer[MAXLINE];
    struct sockaddr_in servaddr, cliaddr;

    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));

    servaddr.sin_family = AF_INET; // IPv4

```

```

servaddr.sin_addr.s_addr = INADDR_ANY;
servaddr.sin_port = htons(PORT);

if (bind(sockfd, (const struct sockaddr *)&servaddr,
    sizeof(servaddr)) < 0)
{
    perror("bind failed");
    exit(EXIT_FAILURE);
}

int len, n;

len = sizeof(cliaddr); // len is value/result

while (1)
{
    n = recvfrom(sockfd, (char *)buffer, MAXLINE,
        MSG_WAITALL, (struct sockaddr *)&cliaddr,
        &len);
    buffer[n] = '\0';
    printf("Client : %s\n", buffer);
    printf("Enter message : ");
    fgets(buffer, MAXLINE, stdin);
    sendto(sockfd, (const char *)buffer, strlen(buffer),
        MSG_CONFIRM, (const struct sockaddr *)&cliaddr,
        len);
    printf("message sent to client.\n");
}

return 0;
}

```

=====

## Screenshot of the Output:

<pre>UDPMulti/ x + ~/ \$ cd UDPMulti ~/UDPMulti/ \$ ./server.out Client : hi it's client 1  Enter message : hello client 1 message sent to client. Client : hey it's client 2  Enter message : hi client 2 message sent to client. Client : hello it's client 3  Enter message : hey client 3 message sent to client. _</pre>	<pre>UDPMulti/ x + ~/ \$ cd UDPMulti ~/UDPMulti/ \$ ./client.out Enter message : hi it's client 1 Message sent. Server : hello client 1  Enter message : _</pre>
<pre>UDPMulti/ x + ~/ \$ cd UDPMulti ~/UDPMulti/ \$ ./client.out Enter message : hey it's client 2 Message sent. Server : hi client 2  Enter message : _</pre>	<pre>UDPMulti/ x + ~/ \$ cd UDPMulti ~/UDPMulti/ \$ ./client.out Enter message : hello it's client 3 Message sent. Server : hey client 3  Enter message : _</pre>