

//BROADCAST SERVER

```
#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>

#define MAX_CLIENTS 10 // Maximum number of clients the server can handle
#define DEFAULT_PORT 12345

int main(int argc, char *argv[]) {
    int server_socket;
    struct sockaddr_in server_address, client_address;
    char buf[1024];
    socklen_t clientLength;
    int port = DEFAULT_PORT;
    int broadcastEnable = 1;
    struct sockaddr_in client_addresses[MAX_CLIENTS];
    int num_clients = 0;

    if (argc > 1) {
        port = atoi(argv[1]);
    }

    // Create socket
    server_socket = socket(AF_INET, SOCK_DGRAM, 0);
    if (server_socket == -1) {
        perror("Error: socket creation failed");
        return 1;
    }

    // Set socket options to allow broadcast
    if (setsockopt(server_socket, SOL_SOCKET, SO_BROADCAST, &broadcastEnable,
sizeof(broadcastEnable)) == -1) {
        perror("Error: setsockopt (SO_BROADCAST)");
        return 1;
    }

    // Fill in server's sockaddr_in
    server_address.sin_family = AF_INET;
    server_address.sin_addr.s_addr = INADDR_ANY;
    server_address.sin_port = htons(port);

    // Bind server socket
    if (bind(server_socket, (struct sockaddr *)&server_address, sizeof(server_address)) ==
-1) {
        perror("Error: bind failed");
        return 1;
    }

    printf("Server is running on port %d...\n", port);

    while (1) {
        clientLength = sizeof(client_address);
        int recv_size = recvfrom(server_socket, buf, sizeof(buf), 0, (struct sockaddr
*)&client_address, &clientLength);
        if (recv_size == -1) {
            perror("Error: recvfrom call failed");
            return 1;
        }

        // Null-terminate the received data
        buf[recv_size] = '\0';

        // Check if the client is already stored
        int client_index = -1;
        for (int i = 0; i < num_clients; ++i) {
            if (client_addresses[i].sin_addr.s_addr == client_address.sin_addr.s_addr &&
client_addresses[i].sin_port == client_address.sin_port) {
                client_index = i;
                break;
            }
        }
    }
}
```

```
// If the client is new, store its address and print connection message
if (client_index == -1) {
    if (num_clients < MAX_CLIENTS) {
        client_addresses[num_clients] = client_address;
        client_index = num_clients;
        num_clients++;
        printf("Client%d connected\n", client_index + 1);
    } else {
        printf("Max clients reached. Cannot accept more clients.\n");
        continue;
    }
}

// Print message from the client
printf("Message from client%d: %s\n", client_index + 1, buf);

// Broadcast message to all clients
for (int i = 0; i < num_clients; ++i) {
    sendto(server_socket, buf, strlen(buf), 0, (struct sockaddr
*)&client_addresses[i], sizeof(client_addresses[i]));
}

// Close server socket
close(server_socket);

return 0;
}
```

```
//BROADCAST CLIENT
```

```
#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>

#define DEFAULT_PORT 12345
#define SERVER_IP "127.0.0.1" // Replace with actual server IP address if needed

int main(int argc, char *argv[]) {
    int client_socket;
    struct sockaddr_in server_address;
    char buf[1024];
    int port = DEFAULT_PORT;
    socklen_t server_address_len;
    int recv_size;

    if (argc > 2) {
        port = atoi(argv[2]);
    }

    // Create client socket
    client_socket = socket(AF_INET, SOCK_DGRAM, 0);
    if (client_socket == -1) {
        perror("Error: client socket not created");
        return 1;
    }

    // Fill in server's sockaddr_in
    server_address.sin_family = AF_INET;
    server_address.sin_port = htons(port);
    server_address.sin_addr.s_addr = inet_addr(SERVER_IP);

    // Send message to the server
    printf("Enter message to send: ");
    fgets(buf, sizeof(buf), stdin);
    sendto(client_socket, buf, strlen(buf), 0, (struct sockaddr *)&server_address,
    sizeof(server_address));

    // Receive broadcasted messages from the server
    while (1) {
        server_address_len = sizeof(server_address);
        recv_size = recvfrom(client_socket, buf, sizeof(buf), 0, (struct sockaddr
    *)&server_address, &server_address_len);
        if (recv_size == -1) {
            perror("Error: recvfrom call failed");
            return 1;
        }
        // Null-terminate the received data
        buf[recv_size] = '\0';

        // Print broadcasted message from the server
        printf("Broadcasted message from server: %s\n", buf);
    }

    // Close client socket
    close(client_socket);

    return 0;
}
```

```
rajagiri@ccf001:~/Aadarsh/CN/cycle3/exp7$ ./bclient.out
Enter message to send: hello cl2
Broadcasted message from server: hello cl2

Broadcasted message from server: hello cl3
```

```
rajagiri@ccf001:~/Aadarsh/CN/cycle3/exp7$ ./bclient.out
Enter message to send: hello cl3
Broadcasted message from server: hello cl3
```

```
rajagiri@ccf001:~/Aadarsh/CN/cycle3/exp7$ gcc bclient.c -o bclient.out
rajagiri@ccf001:~/Aadarsh/CN/cycle3/exp7$ ./bclient.out
Enter message to send: hello cl1
Broadcasted message from server: hello cl1

Broadcasted message from server: hello cl2

Broadcasted message from server: hello cl3
```

```
rajagiri@ccf001:~$ cd Aadarsh/CN/cycle3/exp7
rajagiri@ccf001:~/Aadarsh/CN/cycle3/exp7$ gcc bserver.c -o bserver.out
rajagiri@ccf001:~/Aadarsh/CN/cycle3/exp7$ ./bserver.out
Server is running on port 12345...
Client1 connected
Message from client1: hello cl1

Client2 connected
Message from client2: hello cl2

Client3 connected
Message from client3: hello cl3
```