```c
//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;
}
```