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

#define MAX_CLIENTS 100
#define BUFFER_SZ 2048

static _Atomic unsigned int cli_count = 0;
static int uid = 10;

/* Client structure */
typedef struct{
	struct sockaddr_in address;
	int sockfd;
	int uid;
	char name[32];
} client_t;

client_t *clients[MAX_CLIENTS];

pthread_mutex_t clients_mutex = PTHREAD_MUTEX_INITIALIZER;

void str_overwrite_stdout() {
	printf("\r%s", "> ");
	fflush(stdout);
}

void str_trim_lf (char* arr, int length) {
    int i;
    for (i = 0; i < length; i++) { // trim \n
        if (arr[i] == '\n') {
            arr[i] = '\0';
            break;
        }
    }
}

void print_client_addr(struct sockaddr_in addr){
	printf("%d.%d.%d.%d",
		addr.sin_addr.s_addr & 0xff,
		(addr.sin_addr.s_addr & 0xff00) >> 8,
		(addr.sin_addr.s_addr & 0xff0000) >> 16,
		(addr.sin_addr.s_addr & 0xff000000) >> 24);
}

/* Add clients to queue */
void queue_add(client_t *cl){
	pthread_mutex_lock(&clients_mutex);
	for(int i=0; i < MAX_CLIENTS; ++i){
		if(!clients[i]){
			clients[i] = cl;
			break;
		}
	}

	pthread_mutex_unlock(&clients_mutex);
}

/* Remove clients to queue */
void queue_remove(int uid){
	pthread_mutex_lock(&clients_mutex);

	for(int i=0; i < MAX_CLIENTS; ++i){
		if(clients[i]){
			if(clients[i]->uid == uid){
				clients[i] = NULL;
				break;
			}
		}
	}

	pthread_mutex_unlock(&clients_mutex);
}

/* Send message to all clients except sender */
void send_message(char *s, int uid){
	pthread_mutex_lock(&clients_mutex);

	for(int i=0; i<MAX_CLIENTS; ++i){
		if(clients[i]){
			if(clients[i]->uid != uid){
				if(write(clients[i]->sockfd, s, strlen(s)) < 0){
					perror("ERROR: write to descriptor failed");
					break;
				}
			}
		}
	}

	pthread_mutex_unlock(&clients_mutex);
}

/* Handle all communication with the client */
void *handle_client(void *arg){
	char buff_out[BUFFER_SZ];
	char name[32];
	int leave_flag = 0;

	cli_count++;
	client_t *cli = (client_t *)arg;

	// Name
	if(recv(cli->sockfd, name, 32, 0) <= 0 || strlen(name) <	2 || strlen(name) >= 32-1){
		printf("Didn't enter the name.\n");
		leave_flag = 1;
	} else{
		strcpy(cli->name, name);
		sprintf(buff_out, "%s has joined\n", cli->name);
		printf("%s", buff_out);
		send_message(buff_out, cli->uid);
	}

	bzero(buff_out, BUFFER_SZ);

	while(1){
		if (leave_flag) {
			break;
		}

		int receive = recv(cli->sockfd, buff_out, BUFFER_SZ, 0);
		if (receive > 0){
			if(strlen(buff_out) > 0){
				send_message(buff_out, cli->uid);

				str_trim_lf(buff_out, strlen(buff_out));
				printf("%s -> %s\n", buff_out, cli->name);
			}
		} else if (receive == 0 || strcmp(buff_out, "exit") == 0){
			sprintf(buff_out, "%s has left\n", cli->name);
			printf("%s", buff_out);
			send_message(buff_out, cli->uid);
			leave_flag = 1;
		} else {
			printf("ERROR: -1\n");
			leave_flag = 1;
		}

		bzero(buff_out, BUFFER_SZ);
	}

  /* Delete client from queue and yield thread */
	close(cli->sockfd);
  queue_remove(cli->uid);
  free(cli);
  cli_count--;
  pthread_detach(pthread_self());

	return NULL;
}

int main(int argc, char **argv){
	if(argc != 2){
		printf("Usage: %s <port>\n", argv[0]);
		return EXIT_FAILURE;
	}

	char *ip = "127.0.0.1";
	int port = atoi(argv[1]);
	int option = 1;
```

```
    int listenfd = 0, connfd = 0;
  struct sockaddr_in serv_addr;
  struct sockaddr_in cli_addr;
  pthread_t tid;

  /* Socket settings */
  listenfd = socket(AF_INET, SOCK_STREAM, 0);
  serv_addr.sin_family = AF_INET;
  serv_addr.sin_addr.s_addr = inet_addr(ip);
  serv_addr.sin_port = htons(port);

  /* Ignore pipe signals */
    signal(SIGPIPE, SIG_IGN);

    if(setsockopt(listenfd,                    SOL_SOCKET,(SO_REUSEPORT                    |
SO_REUSEADDR),(char*)&option,sizeof(option)) < 0){
        perror("ERROR: setsockopt failed");
    return EXIT_FAILURE;
    }

    /* Bind */
  if(bind(listenfd, (struct sockaddr*)&serv_addr, sizeof(serv_addr)) < 0) {
    perror("ERROR: Socket binding failed");
    return EXIT_FAILURE;
  }

  /* Listen */
  if (listen(listenfd, 10) < 0) {
    perror("ERROR: Socket listening failed");
    return EXIT_FAILURE;
    }

    printf("=== WELCOME TO THE CHATROOM ===\n");

    while(1){
        socklen_t clilen = sizeof(cli_addr);
        connfd = accept(listenfd, (struct sockaddr*)&cli_addr, &clilen);

        /* Check if max clients is reached */
        if((cli_count + 1) == MAX_CLIENTS){
            printf("Max clients reached. Rejected: ");
            print_client_addr(cli_addr);
            printf(":%d\n", cli_addr.sin_port);
            close(connfd);
            continue;
        }

        /* Client settings */
        client_t *cli = (client_t *)malloc(sizeof(client_t));
        cli->address = cli_addr;
        cli->sockfd = connfd;
        cli->uid = uid++;

        /* Add client to the queue and fork thread */
        queue_add(cli);
        pthread_create(&tid, NULL, &handle_client, (void*)cli);
        /* Reduce CPU usage */
        sleep(1);
    }

    return EXIT_SUCCESS;
}
```