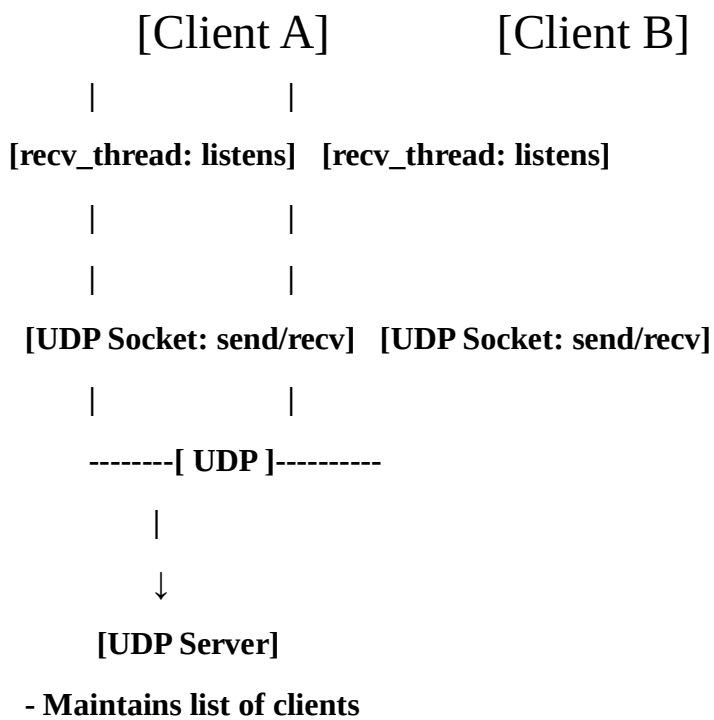
□ UDP Chatbox



- Maintains list of clients

- Broadcasts messages

- Prints locally

- Responds to commands

□ Modular Structure

□ `server.c`

❿ Global Data

- ❿ `clients[]`: Stores info of active clients
- ❿ `server_socket`: UDP socket
- ❿ `pthread_mutex_t lock`: Protects client list

❿ Core Functions

- ❿ `broadcast()`: Sends message to all clients (except sender)
- ❿ `server_task()`: Thread to receive packets, update clients, call `broadcast`
- ❿ `main()`: Creates socket, binds port, spawns thread, handles server's terminal input

□ client.c

❿ Global Data

- ❿ `client_socket`: UDP socket
- ❿ `server_addr`: Destination server IP + port
- ❿ `username[]`: Name of user

❿ Core Functions

- ❿ `send_event()`: Sends a packet to the server (JOIN, MSG, LEAVE)
- ❿ `receive_task()`: Thread to receive broadcasted packets and display them
- ❿ `main()`: Parses args, starts socket + threads, handles user input

□ Packet Flow (State Machine Style)

Step	Sender	Packet Type	Receiver	Action
1	Client	`EVENT_JOIN`	Server	Adds client to list, prints joined
2	Client	`EVENT_MSG`	Server	Prints msg, broadcasts to all clients
3	Server	`EVENT_MSG`	Other Clients	Displays message via `recv_thread`
4	Client	`/exit` → `LEAVE`	Server	Removes client from list, prints left

□ Thread Roles

Thread	Where	Role
`recv_thread`	Client	Listens for server messages
`server_task`	Server	Listens for any client packets
`main thread`	Both	Reads user input, sends messages

□ Data Format: `ChatPacket`

```
typedef struct {
    ChatEventType type;          // JOIN, MSG, LEAVE
    char username[32];           // Sender's name
    char message[1024];          // Text to send
} __attribute__((packed)) ChatPacket;
```

FOLDER LAYOUT:

```
udp-chatbox/
 ├──── server.c
 ├──── client.c             # (optional if you want to split header)
 ├──── README.md            # (optional build file)
```