Design Documentation

Assignment: Multiple Clients and Server

1. Brief overview of your implementation

The skeleton code supported communication between a **single server and a single client**. It relied on the `select` system call to monitor both `STDIN` and the socket for incoming data. Messages typed on the client were displayed on the server, and vice versa.

We extended this implementation to support **multiple clients** using two different models:

(a) Thread-based server (`server_thread.c`)

- The server uses the standard socket sequence:

```
socket();
bind();
listen();
accept();
```

- After every successful `accept`, a **new thread** is created using:

```
pthread_create(&tid, NULL, client_thread, new_s);
```

- Each thread runs the function `event_loop(client_sock)`.

  This function uses `select()` to watch for:

  - input from the connected client socket
  - input from the server's `STDIN`
  - a 30-second timeout

- **Key code fragment (handling client activity):**

```
if (FD_ISSET(client_sock, &readfds)) {
    int bytes = recv(client_sock, buf, sizeof(buf), 0);
    if (bytes <= 0) {
        printf("[Client %d] Disconnected.\n", client_sock);
        close(client_sock);
        return;
    }
    printf("[Client %d]: %s\n", client_sock, buf);
}
```

This ensures that any message received from the client is printed on the server terminal.

- **Key code fragment (handling inactivity):**

```
else if (activity == 0) {
    printf("[Client %d] No activity for 30 seconds.\n", client_sock);
    continue;
}
```

This shows if a client has been idle for 30 seconds.

Thus, each client is handled by a separate thread, and multiple clients can work in parallel.

(b) Event-based server (`server_event.c`)

- Instead of threads, the server maintains an array `client_socks[MAX_CLIENTS]` to store all connected clients.
- It uses a **single `select` loop** to monitor:
  - The listening socket (for new connections)
  - The server's `STDIN`
  - All active client sockets
- **Accepting a new client:**

```
if (FD_ISSET(listen_sock, &readfds)) {
    int new_sock = accept(listen_sock, (struct sockaddr *)&sin, &addr_len);
    printf("New client connected. Socket: %d\n", new_sock);
    // add to client_socks array
}
```

This ensures new clients can connect while the server is running.

- **Handling messages from clients:**

```
int bytes = recv(sock, buf, sizeof(buf), 0);
if (bytes <= 0) {
    printf("[Client %d] Disconnected.\n", sock);
    close(sock);
    client_socks[i] = -1;
} else {
    printf("[Client %d]: %s\n", sock, buf);
}
```

- **Sending messages from server STDIN to one client:**

```
if (FD_ISSET(STDIN_FILENO, &readfds)) {
    fgets(buf, sizeof(buf), stdin);
    for (int i = 0; i < MAX_CLIENTS; i++) {
        if (client_socks[i] != -1) {
            send(client_socks[i], buf, strlen(buf) + 1, 0);
            break;  // only send to one client
        }
    }
}
```

Here, only one active client (first available) receives the message from the server terminal, as required by the assignment.

---

2. Are messages sent by all four clients displayed on the server's terminal?

Yes.

- When four clients are connected, each can type messages independently.
- All such messages are displayed on the server's terminal, with their socket number or thread tag (depending on the implementation).
- Messages typed on the server's terminal are forwarded to one client.

Example observed behavior:

- **Client messages received by server:**

  ```
  [Client 5]: aditya dixit
  [Client 6]: pranjal
  [Client 7]: kanav meena
  [Client 4]: grp-13, hw3
  ```

- **Server messages delivered to client:**

  ```
  (Server typed message → displayed on one client terminal)
  ```

Thus, the system works correctly for at least four clients.

---

3. Name and roll number of the group members

- **PRANJAL BHARTI — 2021080**
- **ADITYA DIXIT — 2022030**
- **KANAV MEENA — 2023266**
