Client/Server Chat Program Design

Wilson Sue & Charmie Jung
November 18, 2023

Structures

State

Command

Component	Purpose
`main`	Entry point for the program. Parses command-line arguments and starts the server or client based on the mode.
`struct ClientInfo`	Structure to hold information about a client, including its socket, index, and the array of connected clients.
`start_server`	Main function for starting the server. Initializes the server socket and handles client connections in a loop.
`handle_client`	Function executed in a separate thread for each connected client. Manages communication with a specific client and broadcasts messages to others.
`create_socket`	Creates a socket for the server or client.
`configure_socket`	Configures socket options, such as setting `SO_REUSEADDR` for the server socket or `FD_CLOEXEC` for the client socket.
`bind_socket`	Binds the server socket to a specific address and port.
`listen_socket`	Listens for incoming connections on the server socket.
`accept_connection`	Accepts a new connection request from a client, returning the new client socket and address.
`select_sockets`	Uses `select` to wait for activity on sockets and handle new connections or console input.
`handle_new_connection`	Handles a new client connection, assigns it an index, and creates a thread to handle its communication.

`broadcast_to_clients`	Broadcasts a message from the server to all connected clients.
`start_client`	Main function for starting the client. Initializes the client socket and handles communication with the server in a loop.
`create_socket`	Creates a socket for the server or client.
`configure_socket`	Configures socket options, such as setting `FD_CLOEXEC` for the client socket.
`connect_to_server`	Initiates a connection to the server.
`receive_from_server`	Receives data from the server and handles disconnection.
`send_to_server`	Sends user input to the server.
`read_console_input`	Reads user input from the console.

Finite State Machine

State Table

State	Input	Next State	Action
Start	-	Binding	Initialize server socket, bind, and listen.
Bind		Binding	Binding to the port and address

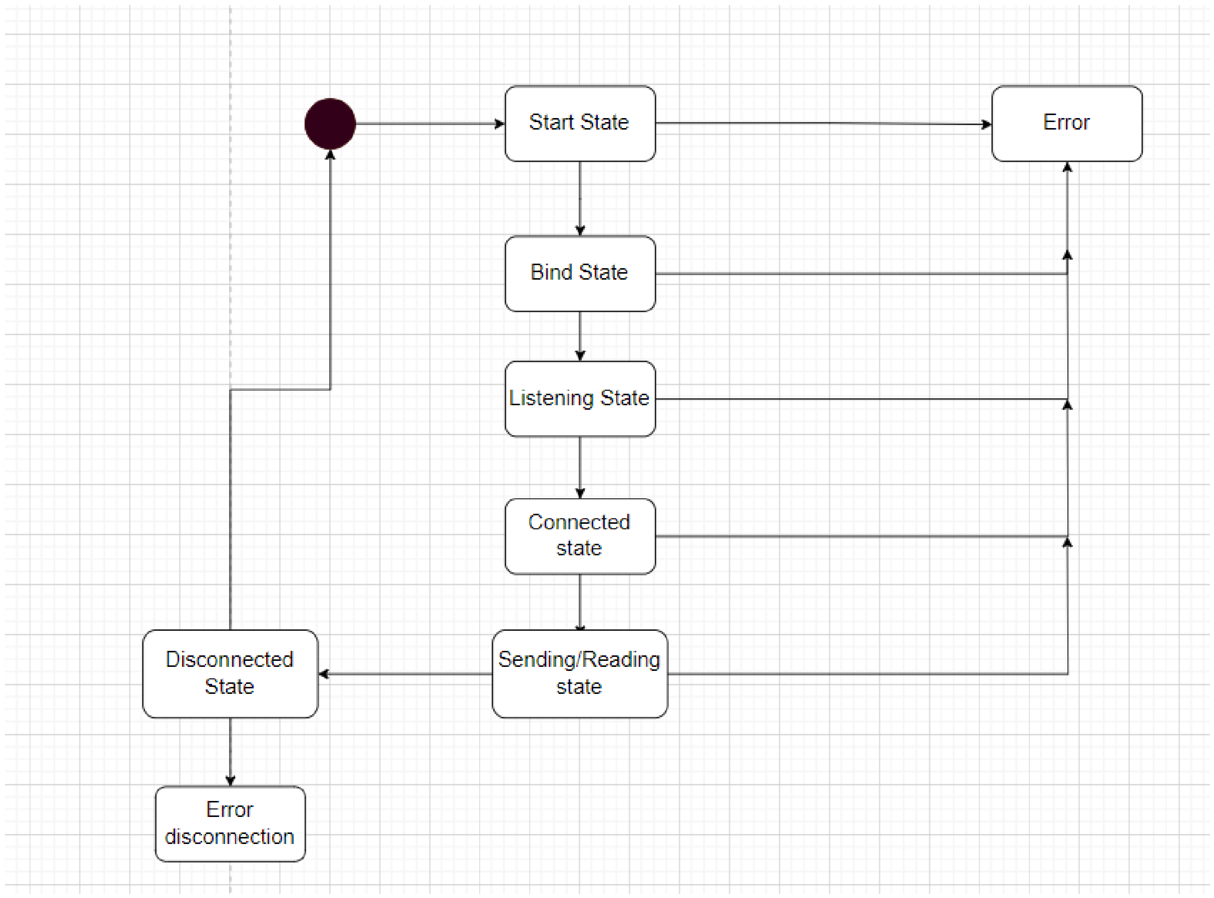
Listening	Connection request received	Connected	Accept new client connection.
Connected	Data received from client	Broadcasting	Handle client messages and broadcast to others.
Broadcasting	Console input available	Connected	Read console input and broadcast to clients.
Connected	Client disconnects	Listening/Disconnected	Handle client disconnect, go back to listening.
Listening/	Console input available	Broadcasting	Read console input and broadcast to clients.
Disconnected			Close sockets, cleanup, and exit.

Client State table

State	Input	Next State	Action
Start	-	Connecting	Initialize client socket and connect to server.
Connecting	Connection successful	Connected	Connection established, start chat loop.
Connecting	Connection failed	Disconnected	Connection failed, close socket and exit.
Connected	Data received from server	Connected	Handle server messages and display.
Connected	Console input available	Connected	Read console input and send to server.

Connected	Ctrl-D (EOF) detected	Disconnected	Close socket, cleanup, and exit.

State Transition Diagram



Functions

handle_client

start_server

start_client

Purpose

Handling the client server

Starting the server for the client to connect to
Starting the client for the peers to send and read

Parameters

Arguments, address and port

Return

Type	Next State
Static void	`Executed in a separate thread for each connected client`
Static void	`Starting the server to the initialised server socket`
Static void	`Start the client to the initialised client socket`
Failure	`ERROR`

Pseudocode

```
// Constants
MAX_CLIENTS = 10
BUFFER_SIZE = 1024
UINT16_MAX = 65535

// Structures
struct ClientInfo {
  int client_socket
  int client_index
  int clients[MAX_CLIENTS]
}

// Functions
function handle_client(arg):
  buffer[BUFFER_SIZE]
  client_info = arg
  client_socket = client_info.client_socket
  client_index = client_info.client_index
  clients = client_info.clients

  loop:
     bytes_received = recv(client_socket, buffer,
sizeof(buffer), 0)
     if bytes_received <= 0:
     print("Server closed the connection.")
     close(client_socket)
     clients[client_index] = 0
     free(client_info)
     exit_thread()

     buffer[bytes_received] = '\0'
     print("Received from Client ", client_index, ": ",
buffer)

     // Broadcast the message to all other connected clients
     for i = 0 to MAX_CLIENTS - 1:
     if clients[i] != 0 and i != client_index:
     send(clients[i], buffer, strlen(buffer), 0)

function start_server(address, port):
  server_socket
  client_socket
  server_addr
  client_addr
  clients[MAX_CLIENTS] = {0}
  optval = 1

  // Socket creation
  server_socket = create_socket()

  // Socket configuration
  configure_socket(server_socket)

  // Bind
  bind_socket(server_socket, address, port)

  // Listen
  listen_socket(server_socket)

  print("Server listening on ", address, ":", port)

  loop:
     // Select activity on sockets
     activity = select_sockets(server_socket, clients)
     // New connection
     if server_socket in activity:
     client_socket, client_addr =
accept_connection(server_socket)

     // Handle new connection
     handle_new_connection(client_socket, client_addr,
clients)

     // Check for console input
     if stdin in activity:
     server_buffer = read_console_input()
     broadcast_to_clients(server_buffer, clients)

function start_client(address, port):
  client_socket
  server_addr

  // Socket creation
  client_socket = create_socket()

  // Socket configuration
  configure_socket(client_socket)

  // Connect to the server
  connect_to_server(client_socket, address, port)

  print("Connected to the server. Type your messages and press
Enter to send. "
     "Press Ctrl-Z to exit or Ctrl-D to close the Server
Connection.")

  // Chat loop
  loop:
     // Select activity on socket and user input
     activity = select_sockets(client_socket, stdin)

     // Check for server message
     if client_socket in activity:
     server_buffer = receive_from_server(client_socket)
     print("Received: ", server_buffer)

     // Check for user input
     if stdin in activity:
```

```
     client_buffer = read_console_input()
     send_to_server(client_socket, client_buffer)
```

read_commands

Purpose

Read a command line from stdin.

Parameters

The state to store the command line into.

Return

Read	Next State
Chat -a <ip address> <port>	`Binds to the socket with Ip address and port.`
Chat -c <ip address> <port>	`Connect to the socket with the same ip address and port.`
Chat -c <ip address> <port> < <file name.txt>	`Reads the content of the file.`
Failure	`ERROR`

Pseudocode

```
// Entry point
function main(argc, argv):
  // Check command line arguments
  if argc != 4:
     print("Usage: ", argv[0], " [-a/-c] <address> <port>")
     exit_failure()

  // Parse command line arguments
  mode = argv[1]
  address = argv[2]
  port = parse_port(argv[3])

  // Start server or client based on mode
  if mode == "-a":
     start_server(address, port)
  elif mode == "-c":
     start_client(address, port)
  else:
     print("Invalid mode. Use -a for the server or -c for the
client.")
     exit_failure()
```

handle_run_error

Purpose

Display the error message when a process fails

Parameters

The error object
The command that executed

Return

Error	Message
Server error	Socket creation failed
Set Socket Options error	Setsockopt failed
Bind error	Bind failed
Listening for connections error	Listen failed
Accept connection error	Accept failed
Thread Creation for Handling Clients error	Thread creation failed
Client Socket Creation error	Socket creation failed
Connect to server error	Connection failed
Select for Socket Activity error	Select error
Receiving Data from Server/Client error	Server closed the connection.

Sending data error	Error sending message

Pseudocode

```
server_socket = create_socket(AF_INET, SOCK_STREAM, 0)
if server_socket == -1
  print_error("Socket creation failed")
  exit_with_failure
if set_socket_option(server_socket, SOL_SOCKET, SO_REUSEADDR,
&optval, sizeof(optval)) == -1
  print_error("Setsockopt failed")
  close_socket(server_socket)
  exit_with_failure
if bind_socket(server_socket, (struct sockaddr *)&server_addr,
sizeof(server_addr)) == -1
  print_error("Bind failed")
  close_socket(server_socket)
  exit_with_failure
if listen_socket(server_socket, valueNew) == -1
  print_error("Listen failed")
  close_socket(server_socket)
  exit_with_failure
client_socket = accept_connection(server_socket, (struct sockaddr
*)&client_addr, (socklen_t *)&client_len)
if client_socket == -1
  print_error("Accept failed")
  close_socket(server_socket)
  exit_with_failure
if create_thread(&tid, NULL, handle_client, (void *)client_info)
!= 0
  print_error("Thread creation failed")
  close_socket(server_socket)
  free_memory(client_info)
  exit_with_failure
client_socket = create_socket(AF_INET, SOCK_STREAM, 0)
if client_socket == -1
  print_error("Socket creation failed")
  exit_with_failure
activity = select_sockets(max_sd + 1, &readfds, NULL, NULL, NULL)
if connect_to_server(client_socket, (struct sockaddr
*)&server_addr, sizeof(server_addr)) == -1
  print_error("Connection failed")
  exit_with_failure

if activity < 0
  print_error("Select error")
  close_socket(server_socket)
  exit_with_failure
bytes_received = receive_data(client_socket, buffer,
sizeof(buffer) - 1, 0)
if bytes_received <= 0
  print_message("Server closed the connection.")
  break_from_loop
if send_data(clients[i], buffer, strlen(buffer), 0) == -1
  print_error("Error sending message")
  break_from_loop
```

do_exit

Purpose

Close the connection between sockets and addresses

Parameters

input_key

Return

`Closed connection`

Pseudocode

```
while (program_running) {
  // Check for keyboard input without blocking
  if (is_keyboard_input_available()) {
    char input_key = read_keyboard_input();

    // Check for Ctrl-z
    if (input_key == CTRL_z) {
      // Perform cleanup actions if needed
      close_resources();

      // Exit the program
      exit_successfully();
    }

    // Handle other keys or continue program logic
    process_key(input_key);
  }

  // Other program logic goes here
  perform_other_tasks();
}
```