Lab 3

1. Explain how the TCP connection is established between the client and server. How does the server handle incoming connections?

 In the code,  a simple TCP server is implemented that listens for incoming connections and responds to messages.

Steps for TCP connection are established between the client and server as listed below.

* The server starts by calling  net.Listen ( “ tcp ” , “ 8080 ” ), which creates a TCP listener on port 8080.
* The server enters an infinite loop with for { . . . }.Within this loop, it calls listener.Accept(), which blocks the execution until a client connects to the server.
* When a client attempts to connect to the server, a TCP handshake occurs.

Steps for how the server handles incoming connections :

* After successfully accepting a connection (conn, err := listener.Accept()),  the server uses go handleClient(conn) to handle the client in a new goroutine. This allows the server to continue listening for other incoming connections without blocking on the current one. Each client can be served concurrently.
* Within the handleClient function, defer conn.Close( ) is used to ensure that the connection is closed properly when the function exits, releasing any resources associated with that connection.
* The server reads messages from the connected client using bufio.NewReader(conn) .ReadString( ‘\n’ ).
* After reading the message, the server prints it to the console and sends a response back to the client with conn.Write( [ ] byte (‘Message received: “ + message) ).  This sends a confirmation message to the client, indicating that the server has received the message.

1. What challenge does the server face when handling multiple clients, and how does Go’s concurrency model help solve this problem?

Challenges are :

* Concurrency Issues : When multiple clients connect and interact with the server, there are potential concurrency issues, especially when reading from and writing to shared resources.
* Managing Client Connections : The server must maintain an accurate list of connected clients, ensuring that when one client disconnects, it is removed from the list.
* Handling Disconnections: If a client disconnects while the server is trying to read from or write to that client, it can cause errors that need to be managed.

In summary, the server faces challenges related to concurrency, resource management, and communication when handling multiple clients

Go’s concurrency model helps to solve it by applying the concepts of :

* Goroutines
* Mutex for Synchronization

Go’s concurrency model, with its goroutines and synchronization mechanisms like mutexes, allows the server to effectively manage these challenges.

1. How does the server assign tasks to the clients? What real-world distributed systems scenario does this model resemble?

The assign tasks to the clients by  taking the following steps :

1. Listening for Client Connections
2. Handling Client Connections
3. Generating and Sending Tasks
4. Receiving Results from Clients
5. Task Interval

Real-world distributed systems scenario which resembles this model will be :

**Load Balancing:**

* The server acts as a load balancer by distributing tasks among multiple clients to prevent any single client from being overwhelmed. This can improve the overall efficiency and responsiveness of the system.