

Reflection Report:

1. Explain how you implemented **concurrency** to handle multiple clients.

To support multiple clients at the same time, I used goroutines. Every time the a client connects, the server calls `go handleConnection(conn)` which creates a separate lightweight thread for each client allowing them to read/write on the server concurrently.

2. Describe the challenges you faced and how you solved them.

Handling concurrent access to the shared data store as multiple clients connect and modify the data at the same time, I would say was challenging because I was seeing unexpected behavior and it occasionally freezes. I had to rerun the client and server file many times. But I implemented mutex to ensure that one operation could modify the store at a time(preventing race condition). Other thing is I tried monitoring client connection and commands that helped me to debug issues in the code to ensure smooth communication between the server and the client.

3. Discuss how you ensured **data consistency** when multiple clients accessed the store concurrently.

Since multiple clients may update or read keys at the same time, a race condition can happen, but to prevent this I used `sync.Mutex`, before modifying the shared map, the server locks the mutex which ensures only one goroutine writes at a time(preventing corrupted data)