Distributed Sequencer

- Siddhant Dilip Godshalwar

- **Sequencer**

  The task is to create a distributed sequencer service that generates unique and sequentially increasing numerical ids for client requests. Each time a client sends a get_id() request, the sequencer service should provide a new id that is one greater than the previous id, ensuring that each id is unique and higher than the previous one.

- **Implementation**

  I have implemented the code using 3 individual components:- sequencer, server, and client

  Sequencer:-

  - The Sequencer class is responsible for generating unique ids, which are sent to clients upon receiving a "get_id" request.
  - The start method sets up a socket connection and generates an id to be broadcasted to available servers via the broadcast method.
  - The broadcast method uses the concurrent.futures.ThreadPoolExecutor class to send the id to each server in a separate thread using the tell_all function.
  - The tell_all function establishes a socket connection to each server and sends the id, which is sent back to the sequencer and then to the client.
  - This ensures all servers have a consistent view of the latest id.

  Server:-

  - The Server class listens for connections on a specified host and port. Upon receiving a message with an id generated by the sequencer, the id, and the client's address are added to a priority queue.
  - The start method sets up the connection, adds the received id to the queue, and calls the request_handler method.
  - The request_handler sends back the next available id to the client if it matches the one at the top of the queue.

- The method checks if the queue is not empty and if the id at the top matches the next available id.
- The loop continues until the connection is closed.

Client:-

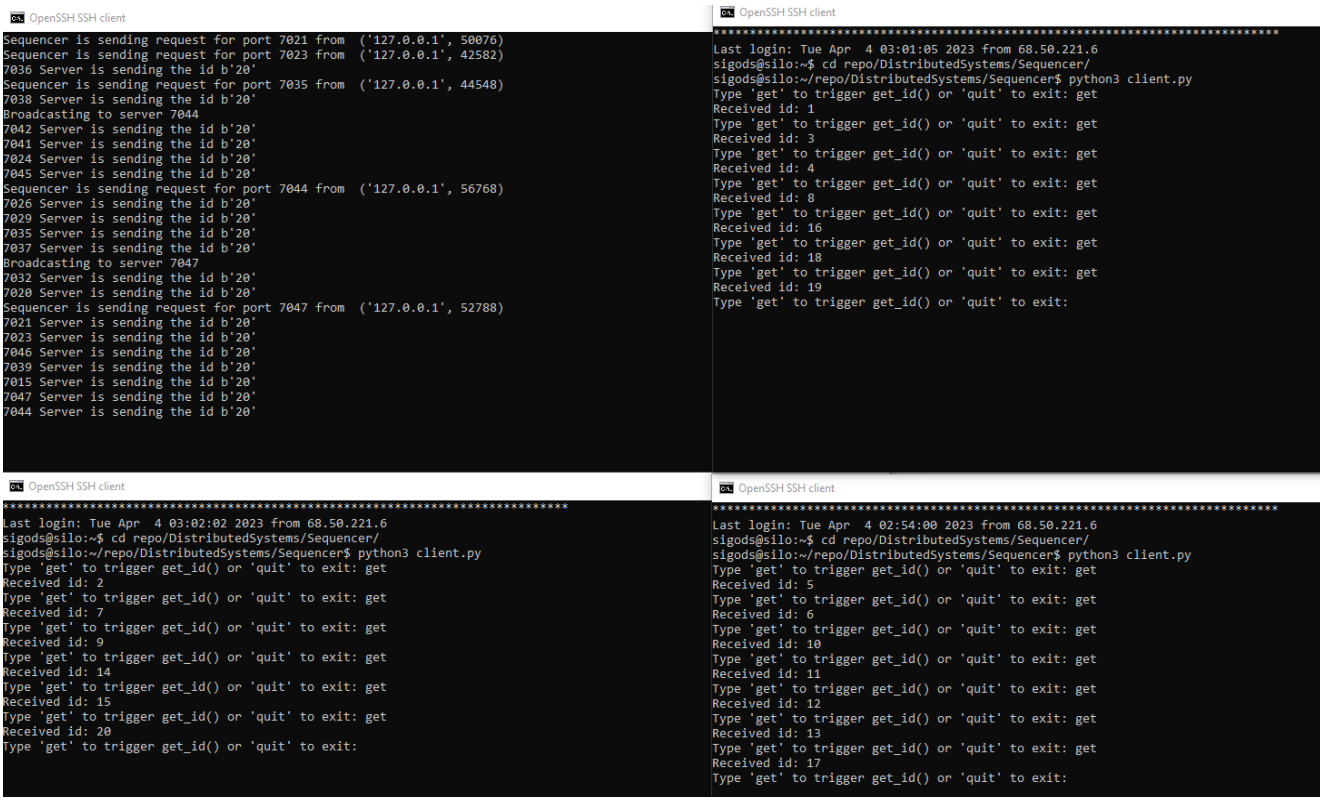
- This script connects to a server using a socket connection and prompts the user to enter 'get' to request a new unique id from the sequencer service, or 'quit' to exit the program.
- If the user enters 'get', the client sends a 'get' message to the server, receives the generated id, and prints it to the console.
- If the user enters 'quit', the program exits the while loop and terminates.

- **<u>Test Case</u>**

  Servers: 48, Clients: 3

  In the screenshot, the top left terminal is for the sequencer.py and the other 3 terminals represent an individual client. As seen, all ids being generated are in handled individually with no repetition.

  This also works for a small number of servers showing same behavior and also for a large number of servers and clients.



- **How to Run**
  Mentioned in the Readme file