Leader Board Structure:

The leader board structure used in this implementation is a linked list within the server. This list contains the name of the player, the number of games played and the number of games won. Each node also points to the next node. The required functionality for board management, such as creating new nodes, updating existing nodes and reordering the nodes are called in separate functions. Reordering the board is completed as score updates are made.

To display the information for the client, the client will first send the request to the server, who then cycles through each node, sending the respective name and scores.

Critical Section management:

The critical section in this implementation is handled using mutex locks, surrounding the sections associated with editing and displaying the scoreboard. Once a user has completed a game and the sever goes to make an update to the scoreboard (or add a new entry), a mutex lock placed around this section of the scoreboard thread as well as around the function to print scoreboard information. Within this lock, the scoreboard is then updated using the scoreboard functions. Once these operations have been completed, the lock is released from both section, allowing users to either update the board or display the information. These locks are also placed when a call to the display functions are made, and released upon completion of the function. This ensures that the scoreboard is only updated or displayed when another user is not already accessing the data.

Thread Pool management:

The thread pool for this assessment is complete by creating a collection of threads that await a client. The main server function initially creates this series of 10 threads within an array. Checks are repeatedly called in a loop to see if there are any requests for the server, and if there are, one of the unused threads in the pool is assigned to that request. This thread then initiates the server actions with a file descriptor unique to that thread. Within the threads, if the client actions are completed, the socket is closed.

Each active thread in the pool is also freed upon receiving the termination signal on the server.

How To Run.

1. Within two Linux Terminals, go to the folder containing all source files.

```
i.e.

cd Assessment
```

2. When in the Assessment folder, run the included Makefile

```
i.e.

make
```

3. After compiling, run the outputted hangman_server file. You can specify a port if you would like, if not, port defaults to 12345.

```
i.e.

./hangman_server

OR

./hangman_server PORT_NUMBER
```

4. After running the server, run the outputted hangman_client file, specified to the server IP and port.

```
i.e.

./hangman_client

OR

./hangman_ client PORT_NUMBER
```

5. Repeat step 4 in a new terminal for each user (up to 10 users).