# CAB403 Problem Solving Task

**Sean Betts:** n8865434

**Michael De Closey:** n9165924

# Contribution

|  |  |
| --- | --- |
| **Student** | **Contribution** |
| Sean Betts | 50% |
| Michael De Closey | 50% |

# Instructions to compile and run the program

To compile the server application, the following files must be in the same directory:

* ClientLeaderboardFunctions.c
* ClientLeaderboardFunctions.h
* ServerCommunication.c
* ServerCommunication.h
* HangmanServer.c
* HangmanServer.Makefile
* TextProcessor.c
* TextProcessor.h

The program can be compiled by opening a command line terminal and navigating it to the directory that these files are located. Within the command line, the following command is given:

make -f HangmanServer.Makefile

The server program can then be called within the command line terminal using the following command. By default, the port number for the server is 12345.

./HangmanServer

If the user wishes to input a custom port number for the server to use then the server must be started with the following command.

./HangmanServer <Port Number>

In the command line, <Port Number>, is replaced with the desired port number. For example, if the user wants to use port 54321, then following command is given.

./HangmanServer 54321

To exit out of the program, press Ctrl+C.

# Client Compilation

To compile the client application, the following files must be in the same directory:

* HangmanClient.c
* ClientCommunication.c
* ClientCommunication.h
* HangmanClient.Makefile

The program can be compiled by opening a new command line terminal and navigating it to the directory that these files are located. Within the command line, the following command is given:

make -f HangmanClient.Makefile

The client program can then be called from within the command line terminal using the following command. The command line must include the IP address for the server as well as the port number that the server is using.

./HangmanClient <Server IP Address> <Port Number>

If the user wishes to connect to the server on the local computer, and the port number is 12345, the following command is given:

./HangmanClient localhost 12345

If the server is on a different computer on the same network it, the client can connect to the server by providing the server’s IP address. If the server’s IP address is 192.168.1.13 and the port number is 12345, then the following command is given:

./HangmanClient 192.168.1.13 12345

To exit out of the program, press Ctrl+C.

# Leaderboard Description

A struct array was used as the main data structure for the leader board. This struct, that is defined as Leaderboard, was used to hold information including a client’s ID, username, games played, games won, and percentage. Clients are added to the struct as soon as they have been logged into to the Hangman Server for the first time. The entry is initialised with the client’s ID and username, and all game variables are set to zero.

The sorting of the leader board in ascending order was performed after a client had completed their game of Hangman. The ordering was done by making use of a sorting algorithm that would compare each client against each other. It will first check the number of games won between to sequentially placed clients, and if they are equal, it will check the percentage of games won by the two clients. If the percentages are equal, it will check which username comes first on the alphabet. The sorting is done by using a nested for-loop that iterates over each pair of clients twice.

The server then sends the username, games won, and games played to the client. This is done by iterating through each client record in the array and sending off the necessary information for that client to the client application.

As this data structure may also include clients that have not played any games, a conditional statement has been added to the function that sends the leader board to the user to ignore any clients with no games played.

|  |
| --- |
| struct \_leaderboard{  int clientId  char \*username;  int gamesPlayed;  int gamesWon;  float percentage;  } temp;  typedef struct \_leaderboard Leaderboard; |

# Critical-Section Solution

The critical-section within the Hangman program was located at operations where the leader board was being updated with new values. This would also cause a race condition to occur when another client would attempt to read the leader board while another tried to update. As such this problem is similar to a reader-writer problem.

As the Hangman program could allow as many reading operations on the leader board. However, only a single write operation on the leader board at any time.

As such a reader-writer solution was implemented with the ability for multiple clients to read the leader board at one time and only one client being able to write to the leader board.

Three pthread\_mutex objects were used to implement the reader-writer solution. Two of these mutex lock were used to control reading access to the leader board. One mutex, the readerMutex, is used to control the counting variable that counts the number of clients reading the leader board. It also locks the writerMutex so that no write function can be performed while the read counter is not zero. This mutex is locked when the leaderboardReadLock() function has been called. It will also lock the readerCounterMutex, and the writerMutex if the readCounter is equal to 1. It will then be unlocked after the readCounter has been updated.

The readerCounterMutex is used to control access to the readerCounter variable which is used to determine the number of clients reading the leader board. This mutex is used so that only one client can update the readerCounter at a time. The readerCounterMutex is locked within the leaderboardReadUnlock() function. It will then decrement the readerCounter before unlocking the readerCounterMutex. If the readCounter is equal to 0, it will unlock the writerMutex.

The writerMutex is used to ensure that only one client can update the leader board at a given time. Other than when a reading function is performed, the writerMutex will be locked when the leaderboardWriteLock() function is called. This function will lock the writerMutex and readerMutex, preventing other clients from reading or writing while it is locked. It is unlocked, along with the readerMutex when the leaderboardWriteUnlock() function is called.

The leaderboard mutex functions were declared on Line 315 to line 360 in the ClientLeaderboardFunctions.c file.

The reader functions were implemented on line 127 and 129 of the HangmanServer.c file.

The writer functions were implemented on line 104 to 106 and line 236 to 238 of the HangmanServer.c file. Lines 104 to 106 were implemented to control the updating of the leader board with new data. The lines from 236 to 238 were used to control access so that a thread could add a new client to the leader board.

# Thread Pool creation and management

The threadpool was created using pthread objects. A pthread array was created and initialised to contain a maximum of 10 different threads. Each thread is used to handle a single client connection. The threads were initialised within the createThreads() function in the HangmanServer.c file.

Client connections are handled through the use of a linked list struct. This struct, the connection struct, holds the port number of a pending client connection, and a pointer to the next struct. Two global linked lists are used to add and get the next available client connection. Two global integer variables are used to count the number of pending connections and the number of active connections. If the number of active connections is equal to 10, any more pending connections will be closed.

When the threads are created, they are given a pointer to the threadConnectionHandler() function, and a thread ID. The thread will enter a continuous while loop that will check the number of pending connections, if there are no pending connections, it will wait for the pthread\_cond to signal that a new connection has been made. If a connection is present, the thread will retrieve the connection from the linked list. It will link the thread to that connection by setting the socketId and the threadId in a struct known as the thread struct. The thread will then enter the handleConnection() function which performs the main logic of the Hangman game.

A pthread\_mutex and a pthread\_cond to control access to the linked list. When a connection has been made with the server, the server will pass the socketId to the addConnection() function. The pthread\_mutex is locked and adds the connection to the linked list. It will then unlock the pthread\_mutex, and signal the pthread\_cond.

When the command to close the program has been issued, the server will call the clearThreads() and clearConnections() functions. The clearThreads() function closes all active connections and frees the dynamic memory allocated to the threads struct. The clearConnections() function closes all pending connections and frees the dynamic memory allocated to the connections and lastConnection struct.