Assignment: Process Management and Distributed Computing

Due Date: **18th October 2017**
Weighting: **40%**
Group or Individual: **Individual/Group of Two**

1 Assignment Description

Task 1: Client-Server Computing

<u>You only need to attempt this task if you are aiming to achieve a mark up to 64% for this assignment.</u>

You have been commissioned to develop a client/server system for an online games provider to expand their current offerings to registered clients. The company wishes to offer to their current clients the game of Hangman, which is a word guessing game.

In the game of Hangman the server randomly selects a two-word phrase from a given text file. The word phrase is represented by a row of dashes representing each letter of the word on the client machine. If the client selects a correct letter the letter should then appear in all the correct places in which it occurs.

The client has a number of turns in which to guess all the letters in the two-word phrase. The number of turns the client receives is proportional to the length of the two word phrase. If the client guesses all the letters before running out of turns the client wins the game. If the client fails to guess all the letters within the number of turns provided the client loses the game.

All the words that are to be used in the game have been provided by the company and they have insisted that this list is *<u>not</u>* to be changed. The information stored in this text file is in the form of **object, object type (**e.g. beach, place**)** – each line in the text file represents a pair of words which are both used in a single game of Hangman.

The client and server will be implemented in the C programming language using BSD sockets on the Linux operating system. This is the same environment used during the weekly practicals. The programs (clients and server) are to run in a terminal reading input from the keyboard and writing output to the screen.

You will have acquired all the necessary knowledge and skills to complete the assignment by attending all the lectures and practicals, and completing the associated reading from the text book.

Server

The server will automatically load the *hangman_text.txt* file which is to be located in the same directory as the server binary file. The server will take only one command line parameter that indicates which port the server is to listen on. If no port number is supplied the default port of 12345 is to be used by the server. The following command will run the server program on port 12345.

```
./Server 12345
```

The server is responsible for ensuring only registered clients of the system can play Hangman. A file named *Authentication.txt* contains the names and the passwords of all registered clients. This file should be located in the same directory as the server binary file. This file is *not* to be changed.

The server should not be able to take any input from the terminal once it is running. The only way that the server is to exit is upon receiving a SIGINT from the operating system (an interrupt signal). SIGINT is a type of signal found on POSIX systems and in the Linux environment this means ctrl+c has been pressed at the server terminal. You need to implement a signal handler and ensure the server exits cleanly. This means the server needs to deal elegantly with any threads that have been created as well as any open sockets, dynamically allocated memory and/or open files. When the server receives a SIGINT, it is to immediately commence the shutdown procedure even if there are currently connected clients.

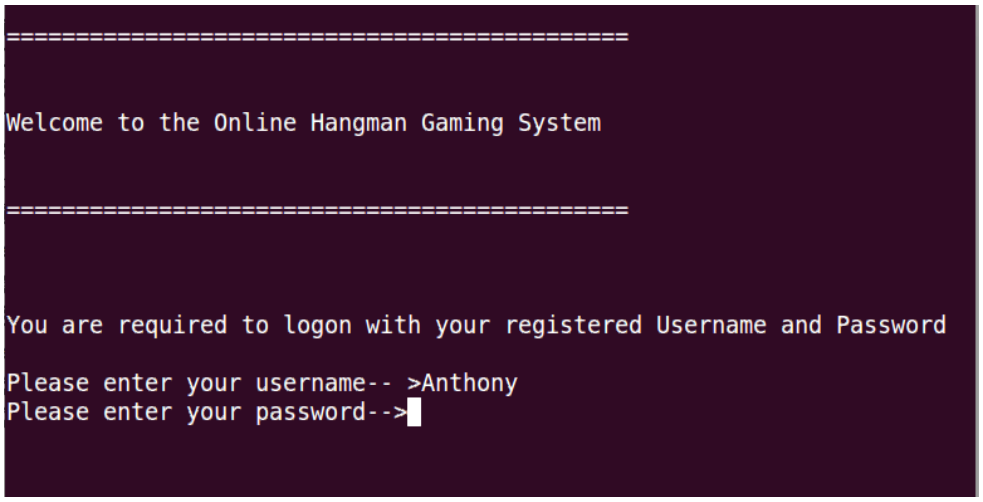
In the 9th edition of the textbook, Section 4.6.2 Signal Handling on page 183 has an introduction to signal handlers and Section 18.9.1 Synchronisation and Signals on page 818 has a detailed description of signals on Linux.

Client

The client will take two (2) command line parameters: hostname and port number. The following command will run the client program connecting to the server on port 12345.

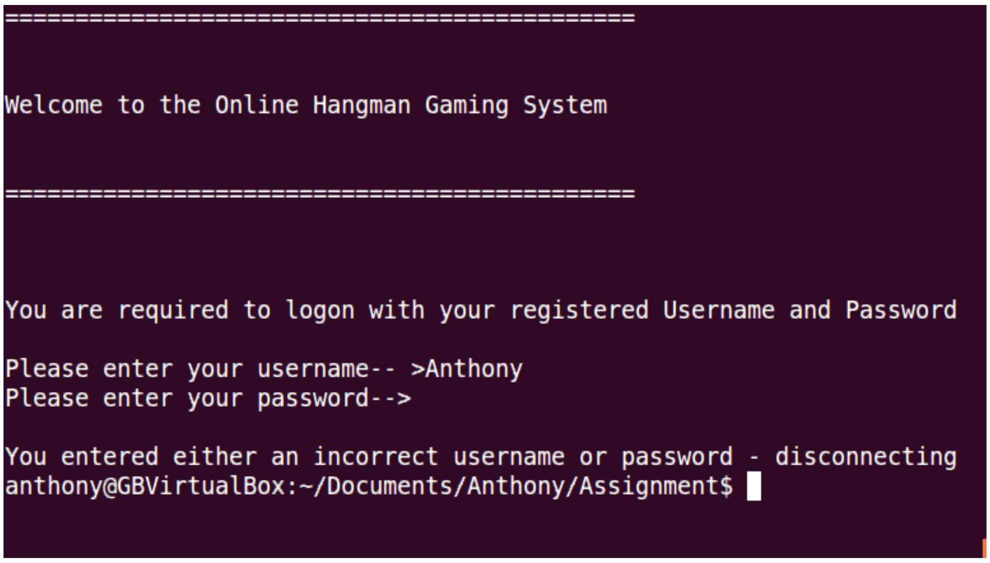
```
./Client server_IP_address 12345
```

The game of Hangman is only available to registered clients of the company's online gaming system. Once the client program starts the client must authenticate by entering a registered name and password which appear in the *Authentication.txt* file located on the server. After the ***server*** validates the client's credentials the client program can proceed to the main menu. If the client does not authenticate correctly the socket is to be immediately closed. (See Figure 1 & Figure 2)



```
=============================================


Welcome to the Online Hangman Gaming System


=============================================



You are required to logon with your registered Username and Password

Please enter your username-- >Anthony
Please enter your password-->
```

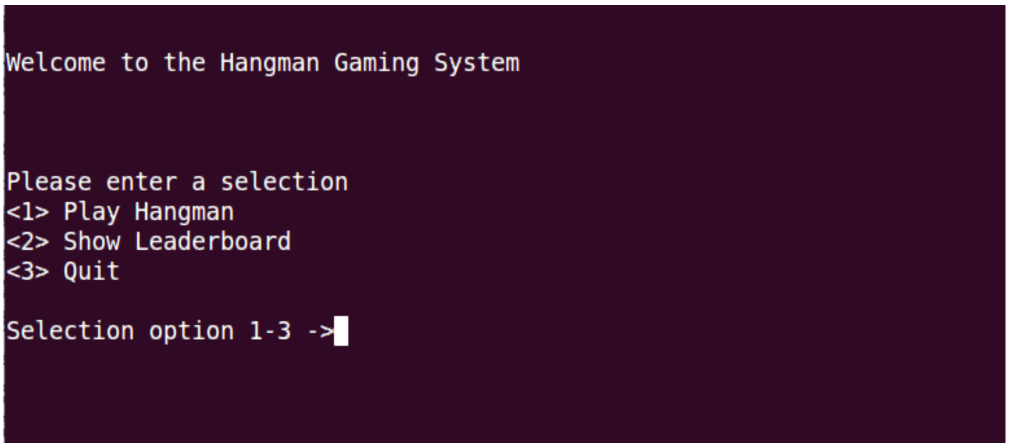
Figure 1. Client logon screen



```
=============================================


Welcome to the Online Hangman Gaming System


=============================================



You are required to logon with your registered Username and Password

Please enter your username-- >Anthony
Please enter your password-->

You entered either an incorrect username or password - disconnecting
anthony@GBVirtualBox:~/Documents/Anthony/Assignment$
```

Figure 2. Unsuccessful Client logon

The client program should be a menu driven application. The client program will have three options: *Play Hangman, Show Leader Board, and Exit.* (Figure 3)



```
Welcome to the Hangman Gaming System


Please enter a selection
<1> Play Hangman
<2> Show Leaderboard
<3> Quit

Selection option 1-3 ->
```

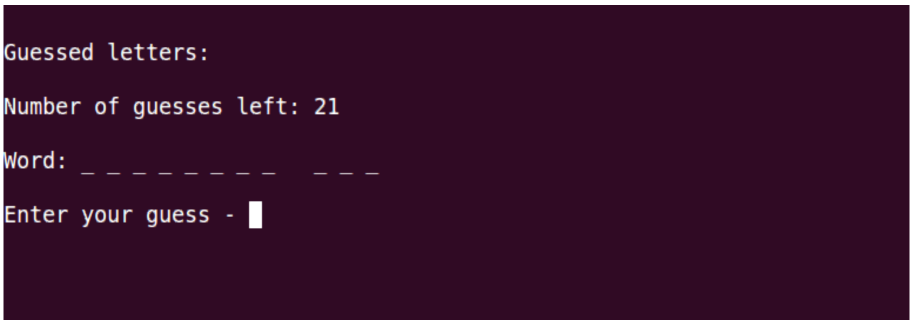
Figure 3. Client Main Menu

If the Client elects to play a game of Hangman, the ***server*** is to select a random pair of words from the *hangman_text.txt* file (e.g. beach, place) to use in the game of Hangman. **Remember each line in the text file represents a pair of words, <u>both</u> of which are to be used in a single game of Hangman**. Once the pair of words has been selected by the server, the client can begin the game of Hangman.

It is not a requirement to draw the typical Hangman stick figure to represent the number of incorrect guesses. Instead a simple interface should be used by the client program to represent:

- Letters that have been already guessed
- Number of guesses left
- The pair of words to be guessed by the client

An initial setup of how the interface should appear on the client when the game first starts is shown in Figure 4. The letters for each pair of words that need to be guessed are initially represented by underscores. The first word represents the **object type (e.g. animal)** and the second word represents the **object (e.g. donkey)**. The Guessed Letters list is empty at the start of the game.



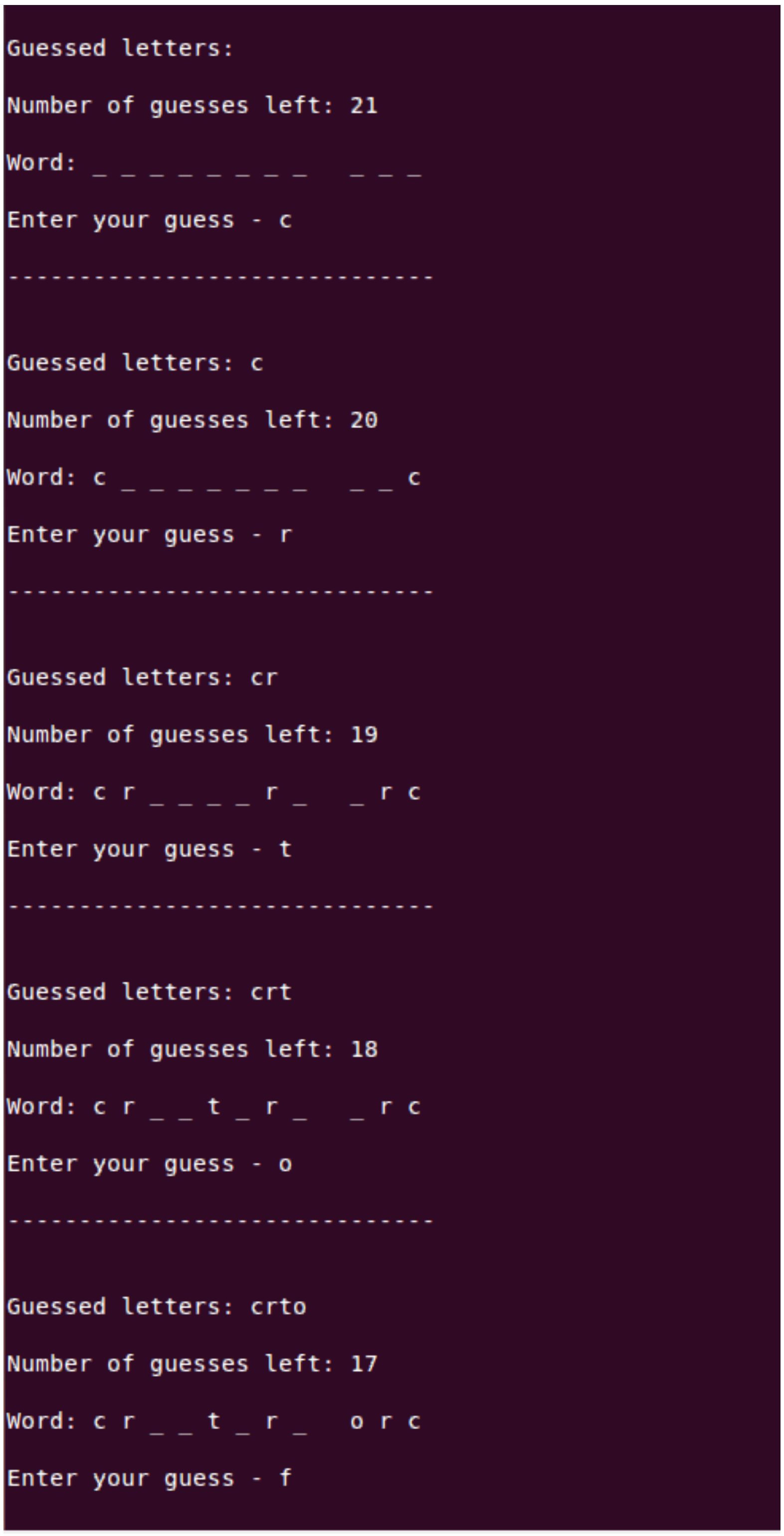
```
Guessed letters:

Number of guesses left: 21

Word: _ _ _ _ _ _ _ _  _ _ _

Enter your guess -
```

Figure 4. Initial Hangman Screen

The number of guesses is dependent on the length of the two (2) words and can be calculated by using the formula:

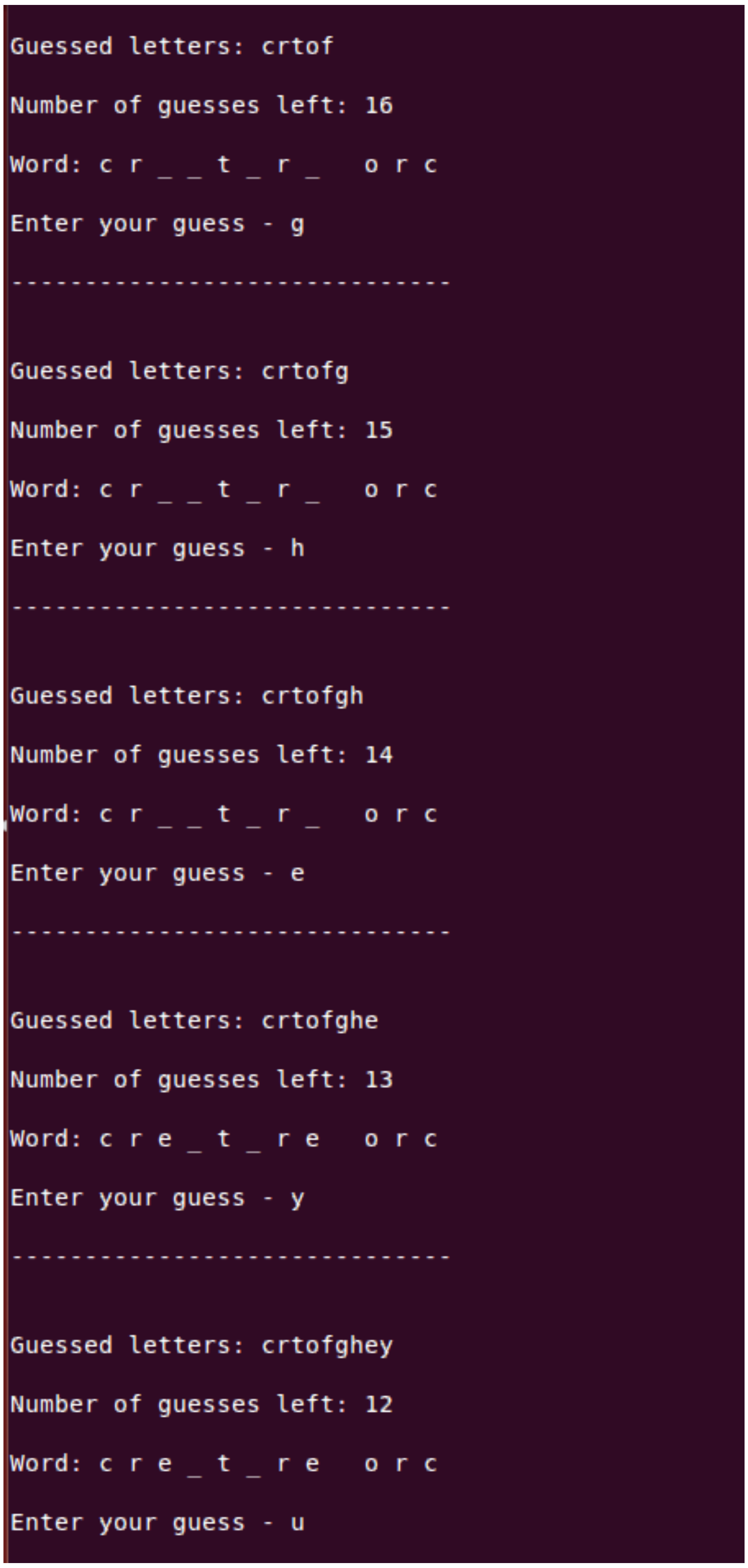
Number of guesses = min{length of word 1 + length of word 2 + 10, 26}

The client then begins the game of Hangman by guessing the letters that exist in the two-word phrase. The letters that have been previously guessed must be updated after each guess is made and the number of remaining guesses needs to decrease by one (1) for each letter guessed. If the client guesses the same letter twice, this is counted as a valid guess. If the client guesses a correct letter, it must be placed in the appropriate position and the underscore removed. An example of how the game should play is shown in Figure 5, 6 and 7.



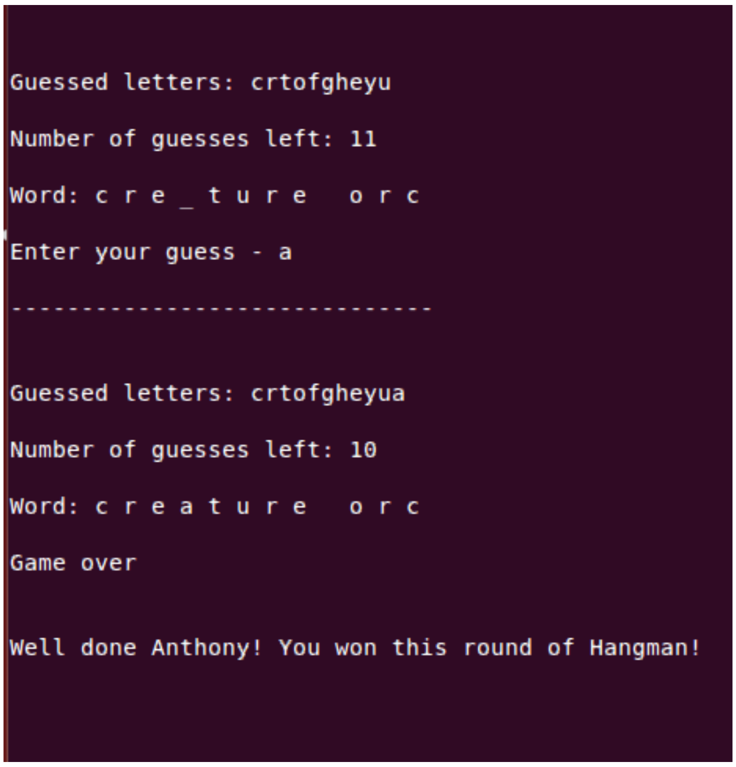
```
Guessed letters:

Number of guesses left: 21

Word: _ _ _ _ _ _ _ _   _ _ _

Enter your guess - c

------------------------------


Guessed letters: c

Number of guesses left: 20

Word: c _ _ _ _ _ _ _   _ _ c

Enter your guess - r

------------------------------


Guessed letters: cr

Number of guesses left: 19

Word: c r _ _ _ _ r _   _ r c

Enter your guess - t

------------------------------


Guessed letters: crt

Number of guesses left: 18

Word: c r _ _ t _ r _   _ r c

Enter your guess - o

------------------------------


Guessed letters: crto

Number of guesses left: 17

Word: c r _ _ t _ r _   o r c

Enter your guess - f
```

Figure 5. As each guess is made the list of guessed letters is updated as well as the number of remaining guesses.



```
Guessed letters: crtof

Number of guesses left: 16

Word: c r _ _ t _ r _   o r c

Enter your guess - g

-----------------------------

Guessed letters: crtofg

Number of guesses left: 15

Word: c r _ _ t _ r _   o r c

Enter your guess - h

-----------------------------

Guessed letters: crtofgh

Number of guesses left: 14

Word: c r _ _ t _ r _   o r c

Enter your guess - e

-----------------------------

Guessed letters: crtofghe

Number of guesses left: 13

Word: c r e _ t _ r e   o r c

Enter your guess - y

-----------------------------

Guessed letters: crtofghey

Number of guesses left: 12

Word: c r e _ t _ r e   o r c

Enter your guess - u
```

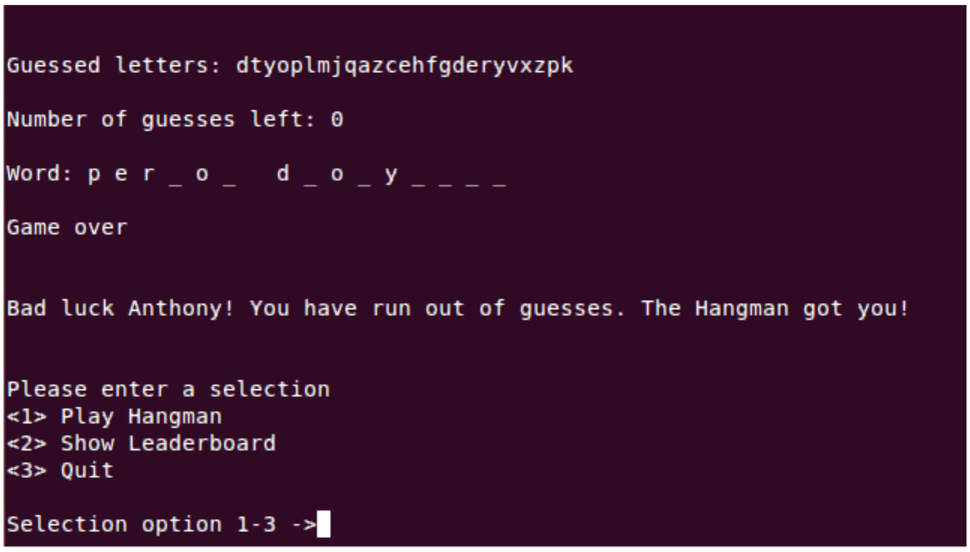
Figure 6. Game of Hangman (continued)



```
Guessed letters: crtofgheyu

Number of guesses left: 11

Word: c r e _ t u r e   o r c

Enter your guess - a

------------------------------

Guessed letters: crtofgheyua

Number of guesses left: 10

Word: c r e a t u r e   o r c

Game over


Well done Anthony! You won this round of Hangman!
```

Figure 7. Client has successfully guessed the two word phrase

If the client does not successfully guess the two-word phrase within the allocated number of guesses the client loses the game. The phrase should *not* be revealed to the client, if they lose the game. Figure 8 shows how the game will end in the case where the client runs out of guesses.

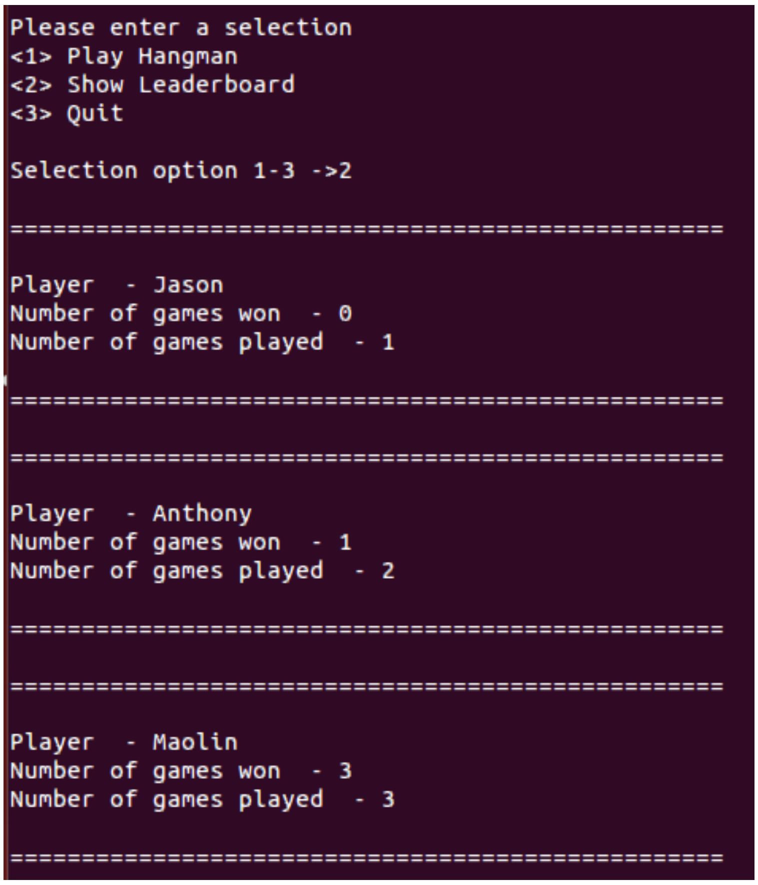


```
Guessed letters: dtyoplmjqazcehfgderyvxzpk

Number of guesses left: 0

Word: p e r _ o _   d _ o _ y _ _ _ _

Game over


Bad luck Anthony! You have run out of guesses. The Hangman got you!


Please enter a selection
<1> Play Hangman
<2> Show Leaderboard
<3> Quit

Selection option 1-3 ->
```

Figure 8. Client has run out of guesses and loses the game

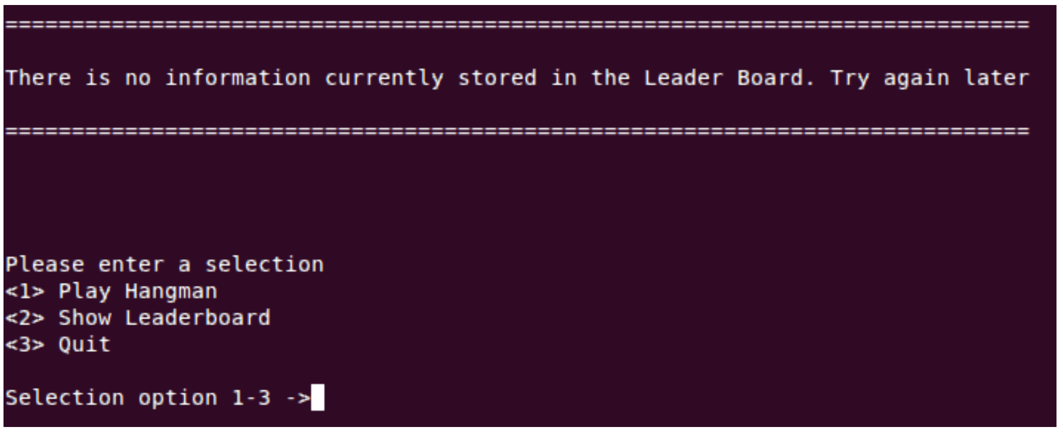
Once the game has been won or lost, the client should return to the main menu.

A Leader Board needs to be maintained for all the registered clients of the Hangman game system. A registered user can request an up-to-date report of the current statistics for all players. The output for the Leader Board can be seen in Figure 9. Only those clients which have played a game should have their statistics shown to the requesting client. The statistics for the clients should be displayed in *ascending* order of *games won*. If two or more players have the same number of games won then the player with the highest percentage of games won should be displayed last (percentage is the number of games won divided by the number of games played). If two or more players have the same number of games won and the same percentage of games won then display those players in alphabetical order. If a client has not logged on and played a game then no statistics should be displayed for this client. If there has been no games played by *any* client then no statistics should be displayed for any player as shown in Figure 10. The lifetime of the Leader Board is only to be maintained for the duration of the server runtime. This information does not need to be persistent between server instances.



```
Please enter a selection
<1> Play Hangman
<2> Show Leaderboard
<3> Quit

Selection option 1-3 ->2

==================================================

Player  - Jason
Number of games won  - 0
Number of games played  - 1

==================================================

==================================================

Player  - Anthony
Number of games won  - 1
Number of games played  - 2

==================================================

==================================================

Player  - Maolin
Number of games won  - 3
Number of games played  - 3

==================================================
```

Figure 9. Example output of the Leader Board if application is multithreaded – if you only complete Task 1 there will only be statistics for the current player)


```
=============================================================================

There is no information currently stored in the Leader Board. Try again later

=============================================================================




Please enter a selection
<1> Play Hangman
<2> Show Leaderboard
<3> Quit

Selection option 1-3 ->
```
Figure 10. No statistics for any players – no game has been played since server started.

IMPORTANT NOTE: The company has specifically requested that the words to be used in a game of Hangman must **never** be sent to the client to prevent unscrupulous users of the Hangman Game system from cheating. After each guess is made on the client the letter is sent to the server for processing. This means the majority of the game logic needs to be implemented on the server side. How this is accomplished is an implementation decision for you as the programmer to decide.

The client/server system is all console based. No bonuses will be given for complex user interfaces. The implementation for Task 1 does not need to handle concurrent connections and therefore does not need to be multithreaded.

Task 2: Multithreaded Programming & Process Synchronisation

You only need to attempt this task if you are aiming to achieve a mark up to 84% for this assignment.

The server you have implemented in Task 1 can only handle a single request at a time. You are required to extend the server to accept multiple concurrent connections. As each new connection is made, a new thread is created to handle the client request.

The server will allow multiple clients to use the system at the same time. You will need to implement a system whereby the server can accept multiple concurrent connections. When a client exits the thread allocated to the client is destroyed.

One potential problem that you must handle in this multithreaded programming task is a so-called Critical-Section Problem because the Leader Board is a database that may be shared among multiple concurrent connections, each of which may read or update the database. You must make sure that there will no connection that is updating the Leader Board while other connections are reading it. However, it is fine to allow multiple connections to read the Leader Board concurrently.

This critical-section problem is similar to the Readers-Writers Problem discussed in Section 5.7.2. Thus, you may reuse the solution to the Readers-Writers Problem. In this critical-section problem, each client (connection) may play the role of Reader or Writer at a time. While the client (connection) plays the Hangman game, its role is Writer as at the end of the game, it will update the Leader Board; while the client (connection) displays the Leader Board, its role is Reader as it only read data from the Leader Board, but never update the Leader Board.

Task 3: Thread Pool

You only need to attempt this task if you are aiming to achieve a mark up to 100% for this assignment.

The server will allow up to 10 clients to use the system at the same time. After the maximum of 10 clients have connected, if a new client tries to connect either the new client will have to wait for another user to exit or the connection may be dropped.

You will need to implement a thread pool where each incoming connection is handled in a separate thread. A thread pool reduces the cost of constantly creating and destroying threads to handle requests. Instead, threads are reused to handle connections. It also limits the number of incoming connections to guarantee the performance of the server.

You can implement your own Thread Pool reusing the producer consumer pattern from the weekly practicals, where the main thread accepts incoming sockets and places them in a queue (the producer) and 10 threads remove sockets from the queue and process the connection (the consumer). You may also use or adapt the complete Thread Pool implementation provided in the weekly practicals.

2 Submission

Your assignment solution will be submitted electronically via Blackboard before 11:59pm on 18th October 2017. If you work in a group, you only need to submit one copy of your group assignment. Your submission should be a zip file and the file name should contain the student number(s) and include the following items:

• All source code of your solution.

• The ***make*** file that you used for generating the executable file, if any.

• A report in Word Document Format or PDF which includes:

a) A statement of completeness indicating the tasks you attempted, any deviations from the assignment specification, and problems and deficiencies in your solution.
b) Information about your team, including student names and student numbers, if applicable
c) Statement of each student's contributions to the assignment. If all members of the group contributed equally, then it is sufficient to state this explicitly.
d) Description of the data structure that is used for the Leader Board in your implementation.
e) Description of how the critical-section problem is handled in Task 2, if applicable.
f) Description of how the thread pool is created and managed in Task 3, if applicable.
g) Instructions on how to compile and run your program.

Note: If the program does not compile on the command line in the Linux environment used during each weekly practical, your submission will be heavily penalised. Implementations written in Visual Studio or other IDE's that do not compile on the Linux command line will receive a mark of zero (0).