DCU School of Computing Assignment Submission

Student Name(s):	Killian Byrne & Alaaldin Afana
Student Number(s):	16781985 & 16395986
Programme:	Bachelor of Science (Hons) in Computer Applications
Project Title:	Assignment 2 - Client-Server text system
Module code:	CA4006
Lecturer:	Dr Rob Brennan
Project Due Date:	13/04/2020

Declaration

I declare that this material, which I now submit for assessment, is entirely my own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my work. I understand that plagiarism, collusion, and copying is a grave and serious offence in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion, or copying. I have read and understood the Assignment Regulations set out in the module documentation. I have identified and included the source of all facts, ideas, opinions, viewpoints of others in the assignment references. Direct quotations from books, journal articles, internet sources, module text, or any other source whatsoever are acknowledged and the source cited are identified in the assignment references.

I have not copied or paraphrased an extract of any length from any source without identifying the source and using quotation marks as appropriate. Any images, audio recordings, video or other materials have likewise been originated and produced by me or are fully acknowledged and identified.

This assignment, or any part of it, has not been previously submitted by me or any other person for assessment on this or any other course of study. I have read and understood the referencing guidelines found at http://www.library.dcu.ie/citing&refguide08.pdf and/or recommended in the assignment guidelines.

I understand that I may be required to discuss with the module lecturer/s the contents of this submission.

I/me/my incorporates we/us/our in the case of group work, which is signed by all of us.

Signed: *Killian Byrne & Alaaldin Afana*

Date: 13/04/2020

Assignment 2 report

Table of contents:

(1) Problem break down:

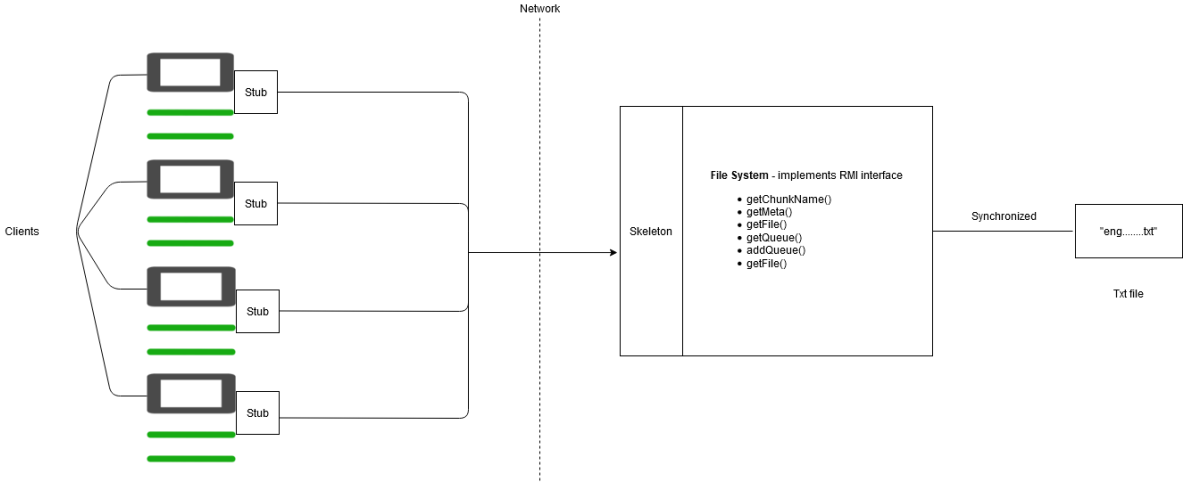
What are the main requirements of this assignment?

(a) Create a client-server application that allows clients to perform different functionalities on text files.

(b) The application must be able to run on two distinct machines connected by a network.

(c) The server must process and manage requests from multiple clients concurrently. The server must also manage it's own hard disk-space and perform multiple functionalities of it's own.

(2) System overview:



(3) Design Documentation:

(A) Distributed Server:

- We implemented our server using the Java Remote Method Invocation (RMI) library. RMI provides a mechanism to create distributed applications in java.

- To create the server, there are two steps involved, creating an interface that defines the client/server contract, and creating an implementation of the interface. In this case, the interface is *FileInterface*. FileServer instantiates the FileSystem object. This FileSystem object (the stub) is then bound to a given namespace within the RMI registry and mapped to a specific port. The server then listens on the specified port for any incoming traffic. Now the server can accept many clients on different ports.

(B) Server Functionality:

- The FileServer class carries out the functionality required. The server prompts the user to enter a start character and an end character for the chunk of letters. The server then passes the characters to getChunked() where a hash map (String, LinkedList) called chunked, is added to with all the characters that exist in that chunk eg a chunk of a - d, would return a hash map of {'a'=[], 'b'=[], 'c'=[], 'd'=[]}. The default txt file is then read in line by line, and if the first letter of the 'word' entry in the txt file matches the given letter from the chunked hash map, then the line is added to the linked list.
- Using this hash map, metadata is then gathered. The metadata returned is the number of words in the chunk e.g. chunk 'ab' - {a: [and, alone], b: [busy]} would return 3 as the total number of words, the sum of all the occurrences of the given words in the chunk is also returned.
- An output file is also generated, that contains the words in the given chunk, in a random order, occurring the number of times that they occur in the txt file. This file is then made available for download.
- To estimate the time of an output file to be generated, we could get a good estimate for the generation of files because of how the method getChunked() reads in the txt file. We know that getChunked() reads the txt file line by line in a for loop and therefore we know that a fixed number of lines would read in, every time. Therefore, if we timed how long it would take for that method to execute, we can save this time as a fixed variable. To make an estimate for the time it takes to generate a given file, we can multiply the number of items ahead of a client's chunk by the average time taken to execute.

(C) Managing disk space:

- In order for the server to protect itself against filling its own disk, we needed to be able to determine how much free space there is currently, check if it is close to our (artificial) limit, and take measures accordingly. To get the disk space of the server, we used a built in java function called ".getFreeSpace()". We then set an artificial limit variable. When the function getChunked() is called, the current disk space is checked against the limit before any other analysis is carried out. If the limit is reached, then an error is returned to the user to inform them that the disk space is full and no more files can be generated.

(D) Client functionality:

- The method getQueue() displays the current state of the queue of chunk requests to the client. The user is prompted in the menu if they want to display the queue or not.
- After the user has specified a chunk, metadata is returned to them about the chunk of letters they have specified. The user is then prompted to decide to download the output file or not. If the user does not cancel the generation of the file, the contents of the hash map, chunked from getChunked() method is used to get the words and total occurrences of each word. The words and occurrences are then used to write to the output in a random order of words. The generated file is named in the format: [chunk letter 1 + chunk letter 2].txt e.g. ab.txt, where a and b are the chunked letters.

(4) Concurrency strategy:

- We synchronize our implementation by making our critical rmi functions synchronised, these functions are functions that use global variables , it is important to have locks on these functions as only one client at a time can change the global variables. We did not synchronize any functions that were reading from the file as the file never changes.

(5) Client & Server failure handling:

- RemoteException that is thrown by accessing a remote object, is handled in a higher level of the client. If a remote exception is thrown the user is prompted to repeat the same procedure they were doing. We also added error handling for server disk space and did not allow itself to fill up its disk space. We done that by creating a conditional statement based on free space in the survey .

(6) Scalability performance study for varying numbers of clients:

- For the scalability study for a number of clients we couldn't automate it as our machines at home are very old and couldn't handle the load. Instead of automation we deployed our server on an ec2 instance and made multiple requests from multiple machines. The server was able to handle 6 different machines running requests on it .

How to run:

To run server : ./run s 0.0.0.0 9999

To run client : ./run c 0.0.0.0 9999