# COMP90015: Distributed Systems – Assignment 1
# Multi-threaded Dictionary Server (15 marks)

## Problem Description

Using a client-server architecture, design and implement a multi-threaded server that allows concurrent clients to search the meaning(s) of a word, add a new word, and remove an existing word.

This assignment has been designed to demonstrate the use of two fundamental technologies that have been discussed during the lectures:

- *Sockets*
- *Threads*

Hence, the assignment must make an **EXPLICIT** use of the two above. By explicit, we mean that in your application, sockets and threads must be the lowest level of abstraction for network communication and concurrency.

## Architecture

- The system will follow a client-server architecture in which multiple clients can connect to a (single) multi-threaded server and perform operations concurrently.

- The multi-threaded server may implement a thread-per-request, thread-per-connection, or worker pool architecture. This is a design decision for you to make.

## Interaction

- All communication will take place via sockets. These sockets can be either TCP or UDP, however, keep in mind that all communication between clients and server is required to be reliable.

- You are responsible for designing your own message exchange protocol. Some data formats that you may use include XML, JSON, Java Object Serialization, or you may choose to design your own.

## Failure Model

- All communication between components must be reliable. If you are using TCP, then the reliability guarantees offered by the protocol are sufficient. If you decide to use UDP, then you must implement an infrastructure that offers reliable communication over UDP. For those of you with previous experience using TCP, using UDP may be a rewarding challenge.

- It is expected that, on both the server and the client side, errors (by means of exception handling) are properly managed. The errors include the following:

- Input from the console for what concerns the parameters passed as command line.
- Network communication (server/address not reachable, bad data...).
- I/O to and from disk (cannot find the dictionary file, error reading the file, etc...).
- Other errors you might come up with.

The application will be tested and validated against all these errors.

## Functional Requirements

- **Query the meaning(s) of a given word**

  The client should implement a function that is used to query the dictionary with the following minimum (additional input/output parameters can be used as required) input and output:

  Input: Word to search

  Output: Meaning(s) of the word

  Error: The client should clearly indicate if the word was not found or if an error occurred. In case of an error, a suitable description of the error should be given to the user.

- **Add a new word**

  Add a new word and one or more of its meanings to the dictionary. For the word to be added successfully it should not exist already in the dictionary. Also, attempting to add a word without an associated meaning should result in an error. A new word added by one client should be visible to all other clients of the dictionary server. The minimum input and output parameters are as follows:

  Input: Word to add, meaning(s)

  Output: Status of the operation (e.g., success, duplicate)

  Error: The user should be informed if any errors occurred while performing the operation.

- **Remove an existing word**

  Remove a word and all of its associated meanings from the dictionary. A word deleted by one client should not be visible to any of the clients of the dictionary server. If the word does not exist in the dictionary then no action should be taken. The minimum input and output parameters are as follows:

  Input: Word to remove

  Output: Status of the operation (e.g., success, not found)

Error: The user should be informed if any errors occurred while performing the operation.

- **Adding additional meaning to an existing word**

  If multiple meanings already exist, you need to add this new one to the existing list (of course, **if the same meaning already exists, it should NOT be added**). Update made by one client should be visible to any of the clients of the dictionary server. If the word does not exist in the dictionary, then no action should be taken. The minimum input and output parameters are as follows:

  Input: Existing Word, new meaning

  Output: Status of the operation (e.g., success, not found)

  Error: The user should be informed if any errors occurred while performing the operation.

- **Updating existing meaning of an existing word by new one**

  If the word and specified meaning already exists, then replace that exiting meaning by new one. Otherwise, no action and flag appropriate message on the client side. Updates made by one client should be visible to any of the clients of the dictionary server. If the word does not exist in the dictionary, then no action should be taken. The minimum input and output parameters are as follows:

  Input: Existing Word, existing meaning, new meaning

  Output: Status of the operation (e.g., success, not found)

  Error: The user should be informed if any errors occurred while performing the operation.

## User Interface

A Graphical User Interface (GUI) is required for this project. You are free to design it and to use any existing tools and libraries for this purpose.

## Implementation Guidelines

These are guidelines only, you are allowed and encouraged to come up with your own design and interfaces (of course, **do not hardcode port number/IP address**).

- When the server is launched, it loads the dictionary data from a file containing the initial list of words and their meanings. This data is maintained in memory in a structure that enables an efficient word search. When words are added or removed, the data structure is updated to reflect the changes.

  A sample command to start the server is:

```
> java –jar DictionaryServer.jar <port> <dictionary-file>
```

Where `<port>` is the port number where the server will listen for incoming client connections and `<dictionary-file>` is the path to the file containing the initial dictionary data.

- When the client is launched, it creates a TCP socket bound to the server address and port number. This socket remains open for the duration of the client-server interaction. All messages are sent/received through this socket.

  A sample command to start the client is:
  ```
  > java –jar DictionaryClient.jar <server-address> <server-port>
  ```

## Implementation Language

The assignment should be implemented in **Java**. Utilization of technologies such as RMI and JMS are **not allowed**.

## Report

You should write a report describing your system and discussing your design choices. Your report should include:

- The problem context in which the assignment has been given.
- A brief description of the components of the system.
- An overall class design and an interaction diagram.
- A critical analysis of the work done followed by the conclusions.

Please note that the report is a **WRITTEN** document, do not put only graphs. A report without any descriptive text addressing the problem, architecture, protocols, and the analysis of the work done will not be considered valid.

The length of the report is not fixed. A good report is auto consistent and contains all the required information for understanding and evaluating the work done. Given the level of complexity of the assignment, a report in the range of 4 to 6 pages is reasonable. Please remember that the length of the report is simply a guideline to help you to avoid writing an extremely long or incomplete report.

It is important to put your details (name, surname, student id) in:

- The head page of the report.
- As a header in each of the files of the software project.

This will help to avoid any mistakes in locating the assignment and its components on both sides.

## Submission

You need to submit **your own work (originally done by you for this subject as per https://academicintegrity.unimelb.edu.au/)** the following via LMS:

1. **Your report is in PDF format *only*.**
2. Two *executable jar* files (DictionaryClient.jar and DictionaryServer.jar) that will be used to run your system on the day of the demo.
3. Your source files are in a .ZIP or .TAR archive *only*.

(**These above items should be submitted each one individually** – NOT as one zip file containing all. Otherwise, there will be **a penalty of -1 mark**).

Submissions will be due on: **April 14, 2025; Monday (5 pm).**

## Marking

The marking process will be structured by first evaluating whether the assignment (application + report) is compliant with the specification given. This implies the following:

- Use of TCP/UDP sockets and threads for the application.
- proper use of concurrency in the server.
- proper handling of the errors.
- a report with the requested content and format (PDF).
- timeliness in submitting the assignment in the proper format (names, directory structure, etc.).
- Client-side GUI.

All of the above elements will earn you a maximum of 10 (out of 15) marks. The rest of the marks are assigned to two elements:

- *Excellence (3 marks: 2 for unique implementation aspects, 1 for report with detailed analysis)*. The assignment has not only been implemented by meeting the minimum requirements, but the design has been well thought out and a proper interaction with the user has been provided (notification of errors, which really help to understand what went wrong). For what concerns the report excellence, the report is complete and well written with the **proper graphs** and a discussion and **analysis of the system implemented** including **advantages and disadvantages of your design choices**. You need to explicitly write in the Report all **Excellence elements** *(point-by-point listing with brief illustration/discussion included) in a separate section.*

- *Creativity Elements (2 marks)*. Any additional implementation such as a GUI for managing the server or other enhancements to the code introducing advanced features (such as processing of user requests within a server in scalable, reliable, and efficient manner; and retry mechanism to reach out the server if access fails in the first instance) will be considered a plus. **For example, using your own implementation of a thread pool (not Java's) would earn you these marks**. You need to explicitly write in the Report all ***Creativity elements*** *(point-by-point listing with brief illustration/discussion included) in a separate section.*

**NOTE 1:** Don't document anything you haven't implemented in the report. This is misconduct and will result in severe penalties.

## Demonstration Schedule and Venue

You are required to provide a demonstration of the working application and will have the opportunity to discuss with the tutors the design and implementation choices made during the demo.

You are free to develop your system where you are more comfortable (at home, on one pc, on your laptop, in the labs...) but keep in mind that the assignment is meant to be a distributed system that works on at least two different machines in order to separate the client from the server.

**We will announce the demo date, time, and venue closer to the due date**. Each tutor will hold 2-3 demo sessions and you will be required to showcase your system in one of the sessions held by the tutor of the workshop in which you are enrolled.

If you need any clarification on the assignment, kindly ask questions during the tutorials or in the LMS forum, so that all students benefit from it.

**NOTE 2: Demo is going to be campus-based, your Tutor will announce the demo schedule. DEMO is mandatory and if you DO NOT demonstrate, there will be penalties of -5 Marks. For the demo, you need to bring your own computer/laptop.**

## Penalties for late submissions of assignments

Assignments submitted late will be penalized in the following way:

- 1 day late: -1 mark
- 2 days late: -2 marks
- 3 days late: -3 marks
- 4 days late: -4 marks
- etc.