**CS4103 – Practical 1 Report**

**Functionalities Implemented**

Communication Setup

A game server was setup to service client requests from prisoners in Java. An interface called () represented the Prosecutor service and it was added to allow for Remote Procedure Calling using Java’s Remote Method Invocation (RMI) framework. Once the basic skeleton of the system was developed, the client could ping the system using (Test connection method) to check if a connection was established. This was all performed in the local registry.

Single to multiple client implementation

After ensuring a basic communication setup between the client and the server, the prisoner could play the game via the command line interface by either cooperating with a second prisoner or betraying them. This resulted in a reduction of their prison sentence depending on the decision of both players as illustrated in the table below:

Initially the decision from the second prisoner was randomly generated at startup however, support for multiple clients was added to allow a second prisoner to log their decision. Once all the response was logged, the system calculated the sentence reduction and updated the sentence. The prisoner already connected to the system would simply poll the server every 10 seconds until a response was received.

Data manager – Multiple case support

The functionality of servicing multiple cases was added by implementing a HashMap where all the cases were stored. This centralised data manager also enabled resource sharing between services.

Appeal Services (Extension)

An appeal service was also added as an extra functionality. The prisoner could appeal for a 1 year sentence reduction which was randomly decided by the appealing system. If successful this service would update the prisoner’s sentence in the central data structure.

**Design**

A 2-Tier Client-Server interface with the middleware handling the connections. The server was responsible for controlling all aspects regarding client requests servicing as well as maintaining the locally stored data. There is support for multiple clients accessing the system in a controlled and thread-safe manner. With regards to the client, some input verification was performed on the client side to reduce communication traffic and also the number of invalid messages that could be transmitted. The instructions were loaded via a text file to simplify maintenance and to reduce the traffic that would be required if the instructions had to be fetched from the system.

A HashMap was used to store all the case information since it allows for a quick look up and it is not of fixed size. This is highly important since multiple clients could simultaneously attempt to access and use this structure which can corrupt it. This is a bottleneck in the system and minimising the time spent here is extremely important for better performance. Additionally, the HashMap increases scalability and flexibility as more cases can be easily added.

The middleware was implemented using Java Remote Method Invocation (RMI) which allows for multiple clients to access the RMI registry and use the methods publicly available via the () provided.

Only a small majority of the methods is synchronised to ensure that assets that are shared cannot be corrupted by effectively locking out the other client. This could have been implemented using asynchronous method calling however, for the purposes of this application, synchronised methods were used simply due to its simple implementation and the fact that the client can wait indefinitely until it receives a response. The polling time was set to 10 seconds to reduce traffic on the server.

The data handling class was created in order to centralise the data storage and allow all the services available to access the data in a controlled manner. However, the Appeal service implemented was simple and it only alters the data of the client currently requesting it, which reduces the chances of data corruption. It was also assumed that this service could only be used if the Prosecutor service (responsible for implementing the prisoner’s dilemma) was used previously.

**Testing (Adding screenshots)**

The system was manually tested and its main functionalities are displayed in the screenshots below. Other scenarios proved more difficult to test such as race conditions since they required two clients accessing the server and using the same resources at exactly the same time. Unit testing could have been implemented for certain blocks of the code developed that were highly independent and modular. This would increase confidence in the robustness of the system.

Furthermore, some scenarios were identified as potential error inducing states however, due to the limited scope of the coursework they were not tested. These include:

* Users with the same login details accessing the system
  + One of the users should be locked out or their connection address checked
* Server or Client post-crashing behaviour
  + Server should disconnect clients gracefully and the client should inform the user of the connection loss instead of hanging.
* The impact of the synchronisation locks put in place on performance when the number of users currently accessing the system increases.
* File loading for instructions.
  + Checking for handling of race conditions using
* Testing for any race conditions that are present via connecting multiple clients accessing shared cases simultaneously.

**Evaluation**

Java RMI is an API that is simple to use and straightforward to setup. It allows for multiple clients to access a registry by default which is extremely useful when developing a client-server system and it can be ran from most systems. Clients’ requests were outsourced to registry which is helpful because there are less operations the developer has to consider when developing applications based on Java RMI. However, this loss of control can be seen as an obstacle since client and server disconnections are harder to manage/keep track.

The application could be improved by implementing packages to help ensure the client and the server files are decoupled and that resources are not incorrectly shared via the application.

The *java.rmi.Naming* class should be used instead of the *java.rmi.registry.LocateRegistry* since the LocateRegistry can only operate on the local host.

However, there are flaws when using Java RMI as a middleware. One of the concerns is security since this middleware requires serialisation to pass objects across the network. The burden of serialisation of the class is placed on the developer who has to consider the issues regarding this process (object destruction during serialisation and re-construction) which has implications on the system. Access restriction to a subset of the objects cannot be enforced after the object has been serialised which can violate the privacy guarantees expected of the object (private, public). On the other hand, encryption and decryption are handled by the lower transport network which facilitates development.

Additionally, a codebase setup should have created to prevent remote code loading (a known exploit= and increase the level of security of the system. The application developed lacks a security policy to enforce.

**Maybe MOM might be better . client does’t need to do much**

Below are identified errors/areas that can improve the system’s performance and robustness:

* The client should only be able to log their decision once. If restarted, the client can change their decision.
* Any users with the same login details using the system should be flagged and access permitted once they have been cleared (Client using the same address).
* Having a centralised data handler creates a bottleneck in the system. Any processes using synchronised methods will lock the object to other threads which impacts performance.
* A timeout for disconnections should be added on the client side to prevent the client clogging the system with polling requests.
* Encryption/Decryption of user details.
* Creating session with all resources required and pass the object to the user instead of constantly passing unencrypted user details (security flaw)

Running and Compiling

More details in readMe text file in submission folder.