CS 454
ASSIGNMENT 3

RPC DOCUMENTATION AND SYSTEM MANUAL

GROUP MEMBERS

- Mahammad Ismayilzada [20496519]
- Aydin Aliyev [20495793]

ARCHITECTURE AND DESIGN CHOICES

This RPC library and binder are completely written in C++. The language choice is because of C++'s object oriented capabilities and built in standard data structures. Therefore, the system is based on several classes that manipulate specific parts of the system and rpc functions use those classes to implement the library calls.
Main classes are the followings:

- **SEGMENT -** this class represents a TCP segment that follows the protocol of the exchange messages and have 3 main fields namely **length, type** and **MESSAGE.** Its main methods are **encapsulate** and **decapsulate.**
- **MESSAGE -** this class represents the base class for different kinds of messages exchanged between binder, client and servers. It does **marshalling** and **unmarshalling** of the messages which are virtual methods overridden by its children:
  - **REQ_REG_MESSAGE -** register request message.
  - **RES_REG_SUCCESS_MESSAGE -** register success response message
  - **REQ_LOC_MESSAGE -** location request message
  - **RES_LOC_SUCCESS_MESSAGE -** location success response message
  - **REQ_EXEC_MESSAGE -** execute request message
  - **RES_EXEC_SUCCESS_MESSAGE -** execute success response message
  - **REQ_TERM_MESSAGE -** terminate request message
  - **RES_FAILURE_MESSAGE -** failure response message

- **SOCK -** this class represents the socket class and handles listening to and accepting and selecting connections. This class has a virtual method called **handle_request** which is overridden by its children to handle different kinds of requests when connection is accepted and/or socket selected. It has two children:
    - **SERVER_SOCK -** this class handles server socket and its connections.
    - **BINDER_SOCK -** this class handles binder socket and its connections
- **FUNC_SIGNATURE -** represents the function signature and has **operator<** and **operator==** overloaded for comparison. Comparison handles function overloading as well.
- **LOCATION -** represents a host location and overloads **operator==** as well.
- **SCHEDULER -** represents simple scheduler to schedule **NUM_THREADS** threads and distribute jobs among them. Server socket uses this to support multithreading.

SENDING/RECEIVING MESSAGES

- **sendSegment -** this method encapsulates the segment into a message and sends it. This method in turn calls the marshalling method of the relevant message. Then it calculates the buffer size and allocates it. Then copies length, type and marshalled message into the buffer and sends it.
    - **marshall -** this method marshalls the arguments depending on its class type. This is a polymorphic method call. It first calculates the buffer length to be allocated and then copies the data in sequence to the buffer.
- **recvSegment -** this method decapsulates the message into a segment and returns it. This method in turn calls the unmarshalling method of the relevant message. It first receives length of 4 bytes, type of 4 bytes and message of (length - 4) bytes and constructs a SEGMENT object and returns it.
    - **unmarshall -** this method unmarshalls the message into arguments depending on its class type. This is a static method call.

BINDER

Binder creates a new BINDER_SOCK and runs it to receive connections and handle them accordingly. The binder database which is called **funcmap** is of type map with keys being of type **FUNC_SIGNATURE** and values being of type **queue of LOCATIONs**. This database supports following main functions:

- **registerLocation -** registers a function signature and location pair in the database. It checks if this signature is already exists or not. This check uses the overloaded operator< and operator== methods which in turn handles function overloading as well. Function overloading is handled by checking if the names, argument types and their order are the same. If the signature exists, it checks if it has the location already register. If so, **EDUPLOCATION** is returned, otherwise location is added to the queue of locations. If the signature is not found, new entry is created in the database with this signature and locations which just contains given location.
- **getLocation -** this method returns a location according to a function signature. If signature not found **ENOLOCATION** is returned, otherwise the location is selected based on round-robin fashion. Round robin is accomplished by selecting location from the front of the queue and pushing it to the back of the queue for all signatures.
- **removeLocation -** this method removes a location from the database when that location is detected to be down. It also checks if the removed location is the last one for a function signature, then it removes the whole entry as well for space efficiency purposes.

Binder also maintains another database which is called **servermap** of type map between **socket fd (int)** and **LOCATION.** This database is used to terminate servers upon terminate request of the client. It does this by calling **terminateLocations** on the database.

ERROR CODES

```
#define ESOCKET         -5      // error opening socket
#define ENOHOST         -6      // error no host found
#define ECONNECT        -7      // error connecting
#define EUNKNOWN        -8      // unknown error
#define ENOSERVER       -9      // error no available server
#define ETHREAD         -10     // error creating thread
#define EACCEPT         -11     // error accepting connection
#define EBIND           -12     // error binding socket
#define ENOFUNCTION     -13     // error no function found
#define EIBINDER        -14     // error invalid binder
#define ENOLOCATION     -15     // error no location found
#define EDUPLOCATION    -16     // error duplicate location
```

WARNING CODES

```
#define WDUPREG          10     // warning duplicate registration
#define WNOLOCATION      11     // warning no location found
```

Note: Bonus functionality has not been implemented.