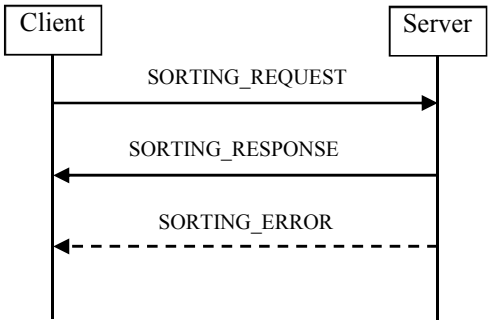
Computer Network Lab (CS 3272)

Assignment 3 - Extension: Application development using TCP Socket

Time: 1 week

Sorter Application:

Problem Description: This assignment aims to implement a TCP Server application that sorts a set of elements provided by the Clients. Initially, a Client sends a SORTING_REQUEST message to the Server. On receiving the message, the Server decodes the message and performs the sanity checking. Once it finds a well-formed packet, it performs *Insertion Sort (or any other kind of sorting)* using the unsorted list of elements. Next, it responds with the SORTING_RESPONSE message to the Client. However, if it finds a malformed SORTING_REQUEST message, it responds with a SORTING_ERROR message with an error code. Upon receiving the SORTING_RESPONSE message, the Client prints the sorted list. If it gets a SORTING_ERROR message, it prints the error message for debugging or rectifying the Client code. The following diagram captures a brief overview of message exchanges.



The message format of this sorter application's Protocol Data Unit (PDU) follows TLV (Type, Length, and Value) format, as mentioned below.

```
|-----------|-----------|----------|
|  Message  |  Message  | Message  |
|   Type    |  Length   |  Value   |
|  (1 byte) | (2 bytes) | (P bytes)|
|-----------|-----------|----------|
```

Field Description:

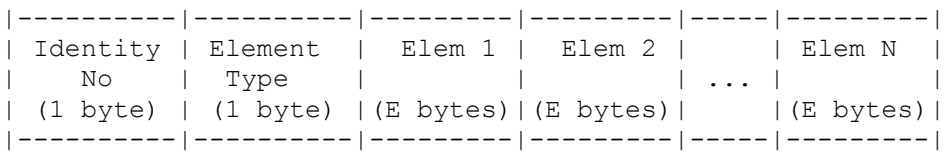
1. **MessageType:** There could be of three kinds of message type as mentioned below.

```
typedef enum MessageType {
      SORTING_REQUEST = 0,
      SORTING_RESPONSE,
      SORTING_ERROR,
} MessageType;
```

2. **MessageLength:** The message length field specifies the length of the entire message, that is, the length of the PDU.

3. **MessageValue**: This field specifies message specific information for the respective message type.

- The MessageValue for the SORTING_REQUEST and SORTING_RESPONSE messages are identical and as stated below –



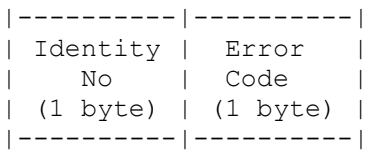
  - **IdentityNo:** The correlation between the SORTING_REQUEST and SORTING_RESPONSE (and SORTING_ERROR) message is maintained using the Identity No field. At the Client, the Identity No field is used to understand for which request the response (or error) is received. This can be a monotonically increasing value starting from 0, 1, 2 … 255 set at Client side. Note that a Client can send many SORTING_REQUEST messages to the Server without waiting for a response from the Server.

  - **ElementType:** This represents different data types of the elements to be sorted. Note that the size of each element E is the same as the size of the data types. For example, E = 1 for *char*, E = 2 for *short*, E = 4 for *int*, etc.

```
typedef enum ElementType {
    CHAR = 0,
    SHORT,
    INT,
    FLOAT,
} ElementType;
```

  - **Elem:** Elem field represent respective elements of the element set.

- The MessageValue field for the SORTING_ERROR message is as stated below -



  - **ErrorCode:** There could be two possible error codes. If the SORTING_REQUEST packet is formatted incorrectly, the Server should respond with the error code MALFORMED_PACKET. The overall size of the SORTING_REQUEST should be at most 100 bytes. In case it exceeds, the Server responds with an error code as TOO_BIG_PACKET.

```
typedef enum ErrorCode {
    MALFORMED_PACKET = 0,
    TOO_BIG_PACKET,
} ErrorCode;
```

Suggestions: Implement the Sorter application, keeping the following aspects in mind.

- The Server can accept many TCP connections from multiple Clients. However, the Server should act as a single-threaded application. It can be achieved using a select() system call.
- Once the TCP connection is established with the Server, a Client can send multiple SORTING_REQUEST without waiting for the response from the Server.
- At the same time, many Clients can send many SORTING_REQUEST simultaneously.
- However, the Server always handles SORTING_REQUEST first come, first serve basis. Hence, all other requests will be waited till the time processing of the current request not completed.
- The function template can be used to implement a shorting technique for different data types like char, short, int, float etc.
- Encoding and decoding of messages should be done as per network byte order.