Nanyang Technological University
School of Computer Science & Engineering

CZ4013 Distributed Systems
Group Project

Mao Jiawei (U1720139L) — 30%
Wei Zi Yun Mark (U1721491K) — 30%
Yik Jia Ler (U1721233K) — 40%

1. Overall Architecture

1.1 Repository Structure

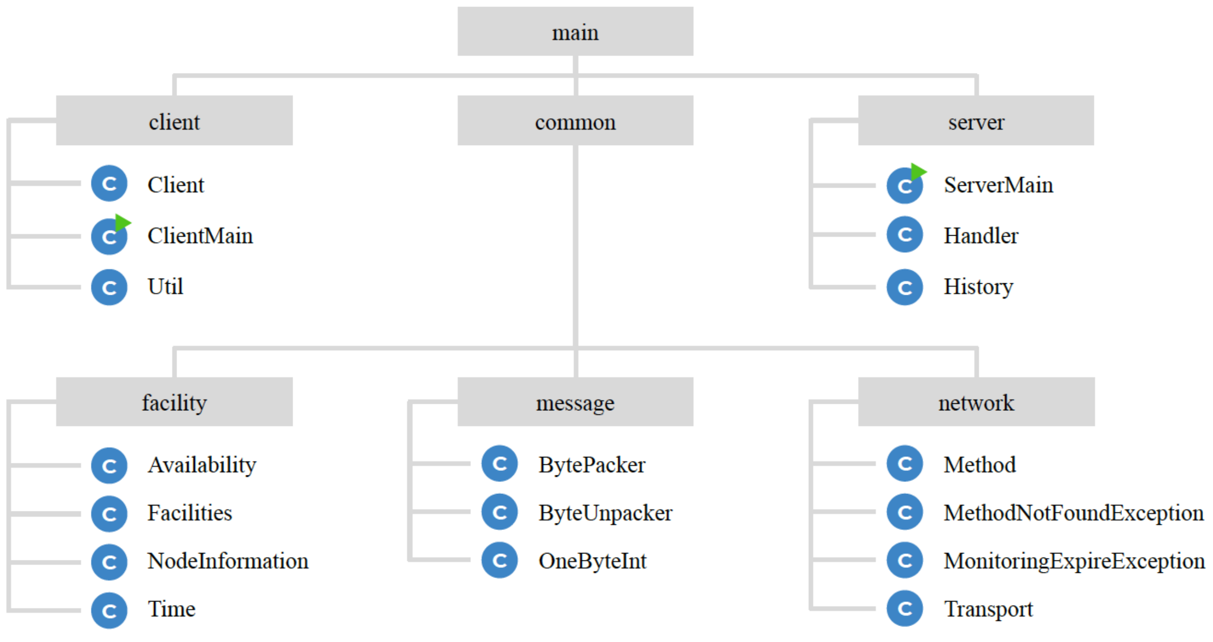


Figure 1: Overall repository structure

Figure 1 above shows the breakdown of this project repository, the structure of folders and corresponding java files.

1.2 Client-Server Communication

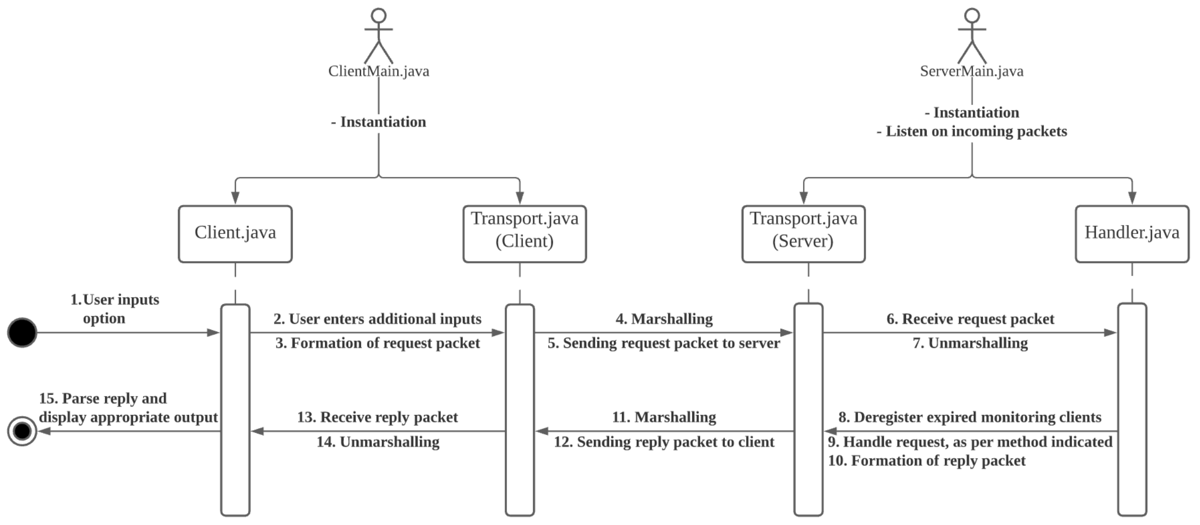

ClientMain.java
- Instantiation
ServerMain.java
- Instantiation
- Listen on incoming packets
Client.java
Transport.java (Client)
Transport.java (Server)
Handler.java
1. User inputs option
2. User enters additional inputs
3. Formation of request packet
4. Marshalling
5. Sending request packet to server
6. Receive request packet
7. Unmarshalling
8. Deregister expired monitoring clients
9. Handle request, as per method indicated
10. Formation of reply packet
11. Marshalling
12. Sending reply packet to client
13. Receive reply packet
14. Unmarshalling
15. Parse reply and display appropriate output

Figure 2: Client-server communication structure

The above UML diagram in Figure 2 depicts the overall logic and information flow within the application. The specifics of the individual functions (e.g. marshalling/unmarshalling, handling of request etc.) would be elaborated further in detail in the following sections.

1.3 Facility Implementation

The class Facilities in Facilities.java houses the bulk of the server-side storage and manipulation of data of our Facility Booking System. In our implementation, we have only allowed bookings for 4 facilities - 2 Lecture Theatres (LT) and 2 Meeting Rooms (MR). The code snippet below shows the 4 facilities implemented.

```
public enum Types {
   LT1,
   LT2,
   MR1,
   MR2
}
```

The local variables in the Facilities class, shown in the screenshot below, are responsible for caching information.

```
private final HashMap<Types, Availability> availability;
private final HashMap<Types, ArrayList<UUID>> bookings;
private final PriorityQueue<LocalDateTime> timePQ;
private final HashMap<LocalDateTime, NodeInformation> timeMap;
private final HashMap<Types, ArrayList<NodeInformation>> monitors;
```

The variable `availability` is a Hashmap that takes in a facility as the key and an Availability object as the value. The Availability class itself contains a `bookings` variable - a Hashmap of UUID and a Pair of Start Time and End Time, corresponding to a particular booking.

```
private final HashMap<UUID, Pair<Time, Time>> bookings;
```

The variable `bookings` is a Hashmap that takes in a facility as the key and an ArrayList of UUIDs. UUID represents the unique ID corresponding to a particular booking made by a client. The Time class simply contains a `day` variable, an `hour` variable and a `minute` variable. It also handles the various arithmetic operations. Essentially, the `availability` Hashmap variable keeps track of the availability status of each facility.

The following 3 variables are responsible for keeping track of monitoring clients.

The variable `timePQ` is a PriorityQueue, which orders LocalDateTime from earliest to latest. The LocalDateTimes are added to the queue whenever a booking is made, and it refers to the DateTime when a client should be deregistered.

Likewise, the variable `timeMap` is a HashMap that takes in a LocalDateTime as the key, and the NodeInformation object corresponding to the client with this particular LocalDateTime, as explained above. The NodeInformation object contains information of a particular client relating to this monitoring session, namely its InetSocketAddress and facility type registered.

Lastly, the variable `monitors` is a HashMap that takes in a facility as the key and an Arraylist of NodeInformation objects as the value. This variable represents the information of clients monitoring each facility, respectively.

2. Additional Services Implemented

2.1 Idempotent — CancelBooking

An idempotent operation proposed is the cancellation of active bookings. The cancellation function takes in the UUID returned to the client during the initial booking and finds the appropriate mapping of the booking to be deleted from the HashMap stored. There is no need to avoid the server executing the operation more than once for this request because for repeated requests, there will not be a wrong booking cancelled as the lookup is done with the UUID. If no UUID is found, the server would simply assume that the active booking has already been deleted and reply to the client accordingly.

2.2 Non-Idempotent — ExtendBooking

A non-idempotent operation proposed is the extension of active bookings. For this operation, there is a need for the server to identify the extendBooking messages sent by the client and filter out the duplicates to avoid executing this process from the same client more than once. This is because extension acts on adding an input time interval (in hours) from the original endTime of the booking. This would result in increments of time being added every time the duplicate message is received.

3 Message Structure

3.1 Client Request Message Structure

Client request messages are standardised to start with:

- `int SERVICE_ID`
- `int MESSAGE_ID`

The subsequent parts of the request messages will contain the necessary arguments required for each service.

For example, a client request message for querying the availability of LT1 would have the following message content:

```
int SERVICE_ID = 2;
int MESSAGE_ID = 0; //Assuming it is the first message sent
String FACILITY = "LT1";
```

Each type of service corresponds to an integer. The types of services are as follows:

```
public static int PING = 0;
public static int QUERY = 1;
public static int ADD = 2;
public static int CHANGE = 3;
public static int MONITOR = 4;
public static int CANCEL = 5;
public static int EXTEND = 6;
```

The different arguments required for each service are defined below:

```
public enum Query {
   FACILITY
}

public enum Add {
   STARTDAY,
   STARTHOUR,
   STARTMIN,
   ENDDAY,
   ENDHOUR,
   ENDMIN,
   FACILITY
}

public enum Change {
   UUID,
   OFFSET
}

public enum Monitor {
   FACILITY,
   INTERVAL
}

public enum Extend {
   UUID,
   EXTEND
}

public enum Cancel {
   UUID
}

public enum Ping {
   PING,
}
```

3.2 Server Response Message Structure

For response messages from the server, the structures are standardised to contain 2 components:

- `int MESSAGE_ID`
- `String REPLY`

MESSAGE_ID is sent back to the client to ensure that the message that the server has received and replied to is the same as what the client has sent initially and that the server is not replying to the wrong type of request.

REPLY is a string that contains the response from the server. For instance, when the client successfully sends a request to add a new booking, the server returns the UUID of that booking to the client in the form of a String. In the case when clients query the the availability of facilities, the ArrayList which stores the bookings of the specified facility will be returned in the form of a String, parsed using the function below:

```java
private static String parseBookingsToString(ArrayList<Pair<Time, Time>> bookings) {
   StringBuilder sb = new StringBuilder();
   for (Pair<Time, Time> b : bookings) {
       Time start = b.getKey();
       Time end = b.getValue();
       sb.append(start.getDayAsName());
       sb.append("/");
       sb.append(start.hour);
       sb.append(":");
       sb.append(start.minute);
       sb.append("-");
       sb.append(end.getDayAsName());
       sb.append("/");
       sb.append(end.hour);
       sb.append(":");
       sb.append(end.minute);
       /** delimiter **/
       sb.append(Method.DELIMITER);
   }
   return sb.toString();
}
```

For the other types of service requests, success and error messages will be returned in the REPLY body from the server back to the client and logged on the console.

3.3 Marshalling

Marshalling of the messages is done in the file `BytePacker.java`. The class is able to marshal 3 types of input: `int`, `String` and `double.`

```java
private int intToByte(int i, byte[] buffer, int index) {

   byte[] temp = new byte[4];
   ByteBuffer.wrap(temp).putInt(i);
   for(byte b: temp){
       buffer[index++] = b;
   }

   return index;
}

private int stringToByte(String s, byte[] buffer, int index) {
   for (byte b : s.getBytes()) {
       buffer[index++] = b;
   }
   return index;
}

private int doubleToByte(Double d, byte[] buffer, int index) {
   byte[] temp = new byte[8];
   ByteBuffer.wrap(temp).putDouble(d);
   for(byte b: temp){
       buffer[index++] = b;
   }
   return index;
}
```

The marshalled inputs are then placed in a `Byte[] byteArray` to be returned to the client class and sent across to the server to be unmarshalled.

3.4 Unmarshalling

Unmarshalling of the messages is done in the file `ByteUnpacker.java`. The class is able to unmarshal the 3 types of input: `int`, `String` and `double` sent across from the client.

```java
private String parseString(byte[] data, int offset, int length) {
   try{
       StringBuilder sb = new StringBuilder();
       for(int i=0;i<length;i++,offset++){
           sb.append((char)data[offset]);
       }
       return sb.toString();
   } catch(IndexOutOfBoundsException e){
       return null;
   }
}

private Double parseDouble(byte[] data, int offset) {
   int doubleSize = 8;
   byte[] temp = new byte[doubleSize];
   for(int i =0;i<doubleSize;i++){
       temp[i] = data[offset+i];
   }
   double value = ByteBuffer.wrap(temp).getDouble();
   return value;
}

private Integer parseInt(byte[] data, int offset) {
   int intSize = 4;
   byte[] temp = new byte[intSize];
   for(int i=0;i<intSize;i++){
       temp[i] = data[offset+i];
   }

   int value = ByteBuffer.wrap(temp).getInt();
   return value;
}
```

4. Invocation Semantics

Invocation semantics can be selected when first running the `server.ServerMain` program. Users can choose between At-Least-Once (ALO) or At-Most-Once (AMO) invocation semantics, after which the server will handle duplicate messages (demarcated by duplicate `messageIds)` as per the invocation semantics selected.

To properly test the invocation semantics during ordinary use, timeout functionality is implemented in the `Client` program to simulate packet loss when transporting packets between clients and server programs. Users first enter a preferred packet reception failure probability (`failureProbability`), and when a client method is called (e.g. `client.addBooking`), the method will fail based on the probability entered, where a higher probability indicates a higher chance of packet loss. If the client 'fails' to receive a reply from the server, a duplicate request will then be sent to the server. An example of this implementation can be in the following code snippet:

```
ByteUnpacker.UnpackedMsg unpackedMsg;
this.transport.send(this.serverAddr, packer);
System.out.println("message sent to server");

while (true) {

   if (this.random.nextDouble() >= failureProbability) {
       unpackedMsg = transport.receivalProcedure(message_id);
       break;
   } else {
       System.out.println("Simulating packet loss");
       Thread.sleep(timeout);
       this.transport.send(this.serverAddr, packer);
   }
}
```

4.1 At-Least-Once

For At-Least-Once (ALO) invocation semantics, duplicate messages from the client are not filtered by the server, and consequently, all client requests received are executed. Because duplicate messages are not filtered under the ALO implementation, non-idempotent function calls will suffer from erroneous execution. For example, repeated client messages for `extendBooking` will result in the specified booking being extended multiple times, compromising the application. Idempotent operations, such as `addBooking`, will be unaffected by duplicate messages.

In the source code, idempotent functions, such as `queryAvailability`, only have At-Least-Once implementation, while non-idempotent functions, such as `extendBooking`, receive a flag to determine At-Least-Once vs At-Most-Once execution.

4.2 At-Most-Once

For At-Most-Once (AMO) invocation semantics, the server instantiates the `History` class, which implements 3 key objects:

1. The `ClientRecord` class: Each client is assigned a specific ClientRecord that stores their `INetSocketAddress` as their identifier
2. A `clientRecordList` ArrayList: Contains a list of all ClientRecords to be searched when a message is received by the client
3. A `messageIdToReplyMap` HashMap: Stores up to 10 server replies for individual messageIds the first time an operation for the client request is executed on the server. Each time a client request is received, the server searches the map for a corresponding

An example of this implementation in the server method handling functions can be seen as follows:

```
ByteUnpacker.UnpackedMsg unpackedMsg = unpacker.parseByteArray(data);
int messageId = unpackedMsg.getInteger(MESSAGE_ID);

// check for duplicates
if(atMostOnce) {
   BytePacker historicalReply = client.findDuplicateMessage(messageId);
   if (historicalReply != null) {
       server.send(clientAddr, historicalReply);
       break outerloop;
   }
}

/* ...
   function handling logic
   ...
*/

// if no duplicate request is found
if (atMostOnce) {
   client.addReplyEntry(messageId, replyMessageClient);
   System.out.println("New reply record added for current client!");
   server.send(clientAddr, replyMessageClient);
}
```

As can be seen in the example above, the `findDuplicateMessage()` method searches through `messageIdToReplyMap` for the client in question to check for previously stored replies to client requests of the same message ID. If no historical reply is found, a new entry is added to the HashMap to be retrieved and sent to the client should a duplicate request be received by the server.

5. Simulation Results

In this section, the servers will be invoked with ALO and AMO semantics respectively, and their results for each function will be compared. To simulate packet loss, the client would request for a failureProbability input shown below.

```
double failureProbability = safeReadDouble("Enter preferred server reply failure probability (0.0
- 1.0): ");
```

To more easily simulate error conditions leading to the sending of duplicate client request messages, three test functions are implemented in the client program, and can be accessed and ran through the client's main menu interface. The individual functions can be found under the `Client` class file, and are listed below:

1. `sendDuplicateExtendsToServer`: Sends multiple booking extension requests for the same booking to the server, with the same messageId. This is a non-idempotent operation.
2. `sendDuplicateCancelsToServer`: Sends multiple booking cancellation requests for the same booking to the server, with the same messageId. This is an idempotent operation.
3. `sendDuplicateChangesToServer`: Sends multiple booking change (offset) requests for the same booking to the server, with the same messageId. This is a non-idempotent operation.

For the report, the results of the above 3 functions will be used to showcase the difference in results between ALO and AMO servers on idempotent and non-idempotent functions.

For this simulation, we will follow the following steps:

1. We first add a new booking for the LT1 facility on Monday, 13:30 - 15:30
2. Using the UUID received, we run `sendDuplicateChangesToServer` to postpone the booking by 30 minutes, with a repeated message count = 5
3. Similarly, we run `sendDuplicateExtendsToServer` to extend the booking by 2 hours with a repeated message count = 5
4. After viewing the results, we then run `sendDuplicateCancelsToServer` to cancel our previous booking with a repeated message count = 5

The results for the simulation can be seen below:

Simulation Results		
	Operation Performed	**Client Booking Output**
ALO	`Start Application`	`null`
	`addBooking()`	`Facility: LT1 | Start: MONDAY 13:30 | End: MONDAY 15:30`
	`sendDuplicateChangesToServer(), offset = 30min`	`Facility: LT1 | Start: MONDAY 16:00 | End: MONDAY 18:00`
	`sendDuplicateExtendsToServer(), extension = 2h`	`Facility: LT1 | Start: MONDAY 16:00 | End: TUESDAY 04:00`
	`sendDuplicateCancelsToServer()`	`null`
AMO	`Start Application`	`null`
	`addBooking()`	`Facility: LT1 | Start: MONDAY 13:30 | End: MONDAY 15:30`
	`sendDuplicateChangesToServer(), offset = 30min`	`Facility: LT1 | Start: MONDAY 14:00 | End: MONDAY 16:00`
	`sendDuplicateExtendsToServer(), extension = 2h`	`Facility: LT1 | Start: MONDAY 14:00 | End: MONDAY 18:00`
	`sendDuplicateCancelsToServer()`	`null`

Table 1: Invocation semantics simulation results

For full screenshots of the simulation, please refer to Appendices A-1 to B-3.

From the results in Table 1, we can see that while idempotent operations such as `sendDuplicateCancelsToServer` execute successfully regardless of the type of invocation semantics used, non-idempotent operations such as `sendDuplicateChangesToServer` see erroneous executions if duplicate messages are not handled correctly. Consequently, the client's booking outputs for the ALO server are erroneous, while outputs for the AMO server are the same as expected.

End of Report

6. Appendix

Appendix A-1: ALO Simulation, sendDuplicateChangesToServer

```
Booking confirmed! UUID of booking: 6b7ef13a-06b0-4f28-bf50-4efdc8a89d3b | Facility: LT1 | Start:
MONDAY 13:30 | End: MONDAY 15:30

Choose your function (Enter '6' for menu): 3

Please enter the confirmation ID of the booking: 6b7ef13a-06b0-4f28-bf50-4efdc8a89d3b

Please enter the offset desired for this booking in minutes (negative for advancement, positive
for postponement)
(1 => 1min, 60 => 1hour, 3600 => 1 day): 30
Status: 0
Response from server: UUID: 6b7ef13a-06b0-4f28-bf50-4efdc8a89d3b. LT1. Success! Booking updated.
Status: 0
Response from server: UUID: 6b7ef13a-06b0-4f28-bf50-4efdc8a89d3b. LT1. Success! Booking updated.
Status: 0
Response from server: UUID: 6b7ef13a-06b0-4f28-bf50-4efdc8a89d3b. LT1. Success! Booking updated.
Status: 0
Response from server: UUID: 6b7ef13a-06b0-4f28-bf50-4efdc8a89d3b. LT1. Success! Booking updated.
Status: 0
Response from server: UUID: 6b7ef13a-06b0-4f28-bf50-4efdc8a89d3b. LT1. Success! Booking updated.

Choose your function (Enter '6' for menu): 5

(MAIN MENU) Your choice of service ('9' for MANUAL): 1
----------------------------------------------------------------
Please choose a facility by typing [1-4]:
1: LT1
2: LT2
3: MR1
4: MR2
0: Exit and perform another query

Your choice of facility: 1
message sent to server
Status: 0
Response from server:
For LT1:
---------------Booking 1---------------
Start Time: MONDAY/16:0
End Time: MONDAY/18:0
```

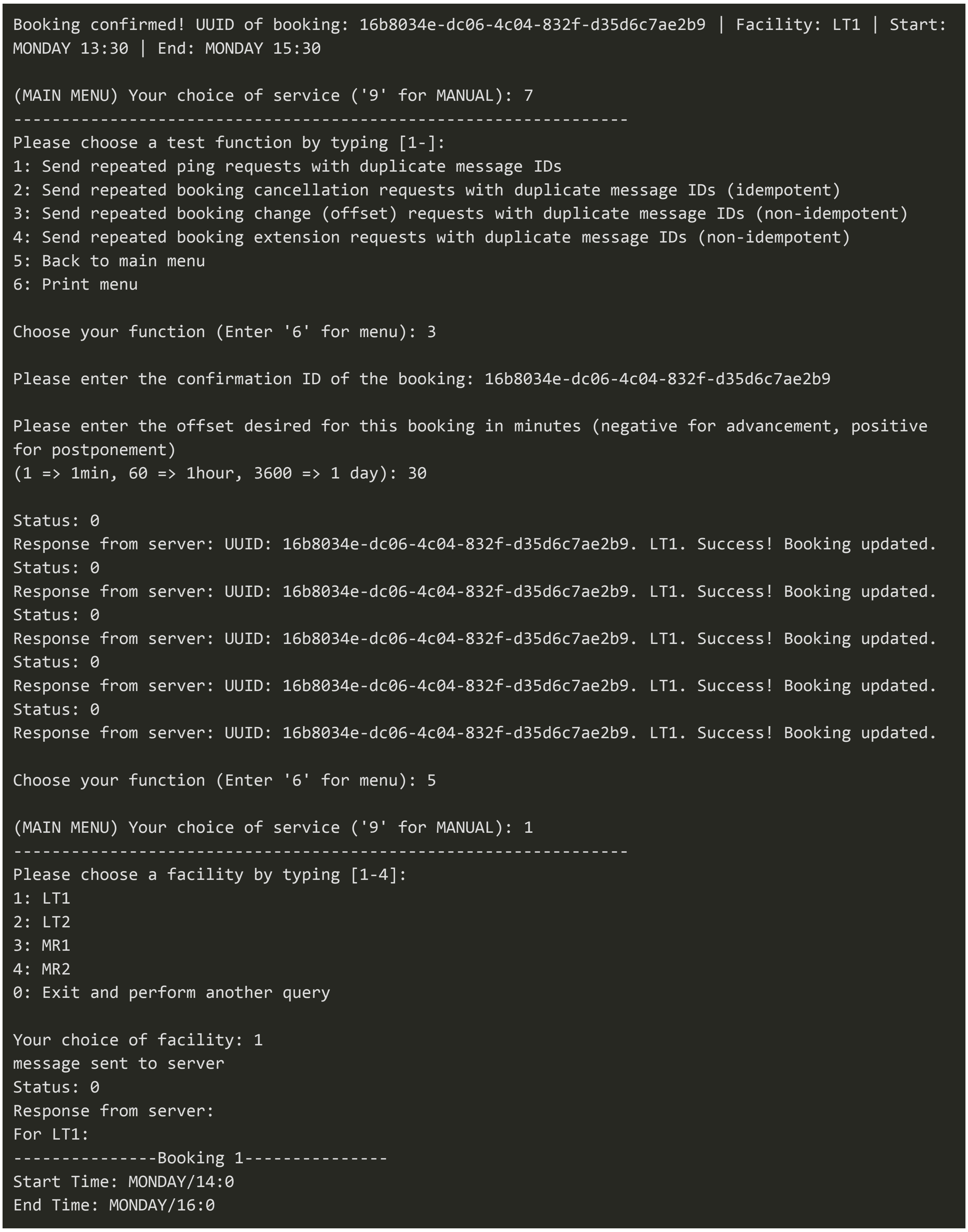
Appendix A-2: ALO Simulation, sendDuplicateExtendsToServer

```
Please enter the confirmation ID of the booking: 6b7ef13a-06b0-4f28-bf50-4efdc8a89d3b

Please enter the extension desired for this booking in hours (30-minute block)
(i.e. 30-minute => 0.5, 2-hours => 2): 2
Status: 0
Response from server: UUID: 6b7ef13a-06b0-4f28-bf50-4efdc8a89d3b. LT1. Success! Booking updated.
Status: 0
Response from server: UUID: 6b7ef13a-06b0-4f28-bf50-4efdc8a89d3b. LT1. Success! Booking updated.
Status: 0
Response from server: UUID: 6b7ef13a-06b0-4f28-bf50-4efdc8a89d3b. LT1. Success! Booking updated.
Status: 0
Response from server: UUID: 6b7ef13a-06b0-4f28-bf50-4efdc8a89d3b. LT1. Success! Booking updated.
Status: 0
Response from server: UUID: 6b7ef13a-06b0-4f28-bf50-4efdc8a89d3b. LT1. Success! Booking updated.


Choose your function (Enter '6' for menu): 5

(MAIN MENU) Your choice of service ('9' for MANUAL): 1
----------------------------------------------------------------
Please choose a facility by typing [1-4]:
1: LT1
2: LT2
3: MR1
4: MR2
0: Exit and perform another query

Your choice of facility: 1
message sent to server
Status: 0
Response from server:
Message received from server:
For LT1:
---------------Booking 1---------------
Start Time: MONDAY/16:0
End Time: TUESDAY/4:0
```

Appendix A-3: ALO Simulation, sendDuplicateCancelsToServer

```
Choose your function (Enter '6' for menu): 2

Please enter the confirmation ID of the booking: 6b7ef13a-06b0-4f28-bf50-4efdc8a89d3b
Message count: 1
Status: 0
Response from server: UUID: 6b7ef13a-06b0-4f28-bf50-4efdc8a89d3b. LT1. Success! Booking removed.
Message count: 2
Status: 0
Response from server: UUID: 6b7ef13a-06b0-4f28-bf50-4efdc8a89d3b. LT1. Failure! Booking cannot be
found.
Message count: 3
Status: 0
Response from server: UUID: 6b7ef13a-06b0-4f28-bf50-4efdc8a89d3b. LT1. Failure! Booking cannot be
found.
Message count: 4
Status: 0
Response from server: UUID: 6b7ef13a-06b0-4f28-bf50-4efdc8a89d3b. LT1. Failure! Booking cannot be
found.
Message count: 5
Status: 0
Response from server: UUID: 6b7ef13a-06b0-4f28-bf50-4efdc8a89d3b. LT1. Failure! Booking cannot be
found.

Choose your function (Enter '6' for menu): 5

(MAIN MENU) Your choice of service ('9' for MANUAL): 1
----------------------------------------------------------------
Please choose a facility by typing [1-4]:
1: LT1
2: LT2
3: MR1
4: MR2
0: Exit and perform another query

Your choice of facility: 1
message sent to server
Status: 0
Response from server:
No active bookings for LT1
```

Appendix B-1: AMO Simulation, sendDuplicateChangesToServer



```
Booking confirmed! UUID of booking: 16b8034e-dc06-4c04-832f-d35d6c7ae2b9 | Facility: LT1 | Start:
MONDAY 13:30 | End: MONDAY 15:30

(MAIN MENU) Your choice of service ('9' for MANUAL): 7
---------------------------------------------------------------
Please choose a test function by typing [1-]:
1: Send repeated ping requests with duplicate message IDs
2: Send repeated booking cancellation requests with duplicate message IDs (idempotent)
3: Send repeated booking change (offset) requests with duplicate message IDs (non-idempotent)
4: Send repeated booking extension requests with duplicate message IDs (non-idempotent)
5: Back to main menu
6: Print menu

Choose your function (Enter '6' for menu): 3

Please enter the confirmation ID of the booking: 16b8034e-dc06-4c04-832f-d35d6c7ae2b9

Please enter the offset desired for this booking in minutes (negative for advancement, positive
for postponement)
(1 => 1min, 60 => 1hour, 3600 => 1 day): 30

Status: 0
Response from server: UUID: 16b8034e-dc06-4c04-832f-d35d6c7ae2b9. LT1. Success! Booking updated.
Status: 0
Response from server: UUID: 16b8034e-dc06-4c04-832f-d35d6c7ae2b9. LT1. Success! Booking updated.
Status: 0
Response from server: UUID: 16b8034e-dc06-4c04-832f-d35d6c7ae2b9. LT1. Success! Booking updated.
Status: 0
Response from server: UUID: 16b8034e-dc06-4c04-832f-d35d6c7ae2b9. LT1. Success! Booking updated.
Status: 0
Response from server: UUID: 16b8034e-dc06-4c04-832f-d35d6c7ae2b9. LT1. Success! Booking updated.

Choose your function (Enter '6' for menu): 5

(MAIN MENU) Your choice of service ('9' for MANUAL): 1
---------------------------------------------------------------
Please choose a facility by typing [1-4]:
1: LT1
2: LT2
3: MR1
4: MR2
0: Exit and perform another query

Your choice of facility: 1
message sent to server
Status: 0
Response from server:
For LT1:
---------------Booking 1---------------
Start Time: MONDAY/14:0
End Time: MONDAY/16:0
```

Appendix B-2: AMO Simulation, sendDuplicateExtendsToServer

```
(MAIN MENU) Your choice of service ('9' for MANUAL): 7
----------------------------------------------------------------
Please choose a test function by typing [1-]:
1: Send repeated ping requests with duplicate message IDs
2: Send repeated booking cancellation requests with duplicate message IDs (idempotent)
3: Send repeated booking change (offset) requests with duplicate message IDs (non-idempotent)
4: Send repeated booking extension requests with duplicate message IDs (non-idempotent)
5: Back to main menu
6: Print menu


Choose your function (Enter '6' for menu): 4

Please enter the confirmation ID of the booking: 16b8034e-dc06-4c04-832f-d35d6c7ae2b9

Please enter the extension desired for this booking in hours (30-minute block)
(i.e. 30-minute => 0.5, 2-hours => 2): 2
Status: 0
Response from server: UUID: 16b8034e-dc06-4c04-832f-d35d6c7ae2b9. LT1. Success! Booking updated.
Status: 0
Response from server: UUID: 16b8034e-dc06-4c04-832f-d35d6c7ae2b9. LT1. Success! Booking updated.
Status: 0
Response from server: UUID: 16b8034e-dc06-4c04-832f-d35d6c7ae2b9. LT1. Success! Booking updated.
Status: 0
Response from server: UUID: 16b8034e-dc06-4c04-832f-d35d6c7ae2b9. LT1. Success! Booking updated.
Status: 0
Response from server: UUID: 16b8034e-dc06-4c04-832f-d35d6c7ae2b9. LT1. Success! Booking updated.

Choose your function (Enter '6' for menu): 5

(MAIN MENU) Your choice of service ('9' for MANUAL): 1
----------------------------------------------------------------
Please choose a facility by typing [1-4]:
1: LT1
2: LT2
3: MR1
4: MR2
0: Exit and perform another query

Your choice of facility: 1
message sent to server
Status: 0
Response from server:
For LT1:
---------------Booking 1---------------
Start Time: MONDAY/14:0
End Time: MONDAY/18:0
```

Appendix B-3: AMO Simulation, sendDuplicateCancelsToServer

```
(MAIN MENU) Your choice of service ('9' for MANUAL): 7
----------------------------------------------------------------
Please choose a test function by typing [1-]:
1: Send repeated ping requests with duplicate message IDs
2: Send repeated booking cancellation requests with duplicate message IDs (idempotent)
3: Send repeated booking change (offset) requests with duplicate message IDs (non-idempotent)
4: Send repeated booking extension requests with duplicate message IDs (non-idempotent)
5: Back to main menu
6: Print menu


Choose your function (Enter '6' for menu): 2

Please enter the confirmation ID of the booking: 16b8034e-dc06-4c04-832f-d35d6c7ae2b9
Message count: 1
Status: 0
Response from server: UUID: 16b8034e-dc06-4c04-832f-d35d6c7ae2b9. LT1. Success! Booking removed.
Message count: 2
Status: 0
Response from server: UUID: 16b8034e-dc06-4c04-832f-d35d6c7ae2b9. LT1. Failure! Booking cannot be
found.
Message count: 3
Status: 0
Response from server: UUID: 16b8034e-dc06-4c04-832f-d35d6c7ae2b9. LT1. Failure! Booking cannot be
found.
Message count: 4
Status: 0
Response from server: UUID: 16b8034e-dc06-4c04-832f-d35d6c7ae2b9. LT1. Failure! Booking cannot be
found.
Message count: 5
Status: 0
Response from server: UUID: 16b8034e-dc06-4c04-832f-d35d6c7ae2b9. LT1. Failure! Booking cannot be
found.

Choose your function (Enter '6' for menu): 5

(MAIN MENU) Your choice of service ('9' for MANUAL): 1
----------------------------------------------------------------
Please choose a facility by typing [1-4]:
1: LT1
2: LT2
3: MR1
4: MR2
0: Exit and perform another query

Your choice of facility: 1
message sent to server
Status: 0
Response from server:
No active bookings for LT1
```