Asst 03: DUMB Testplan

1 Introduction

Not only do we want successful communication between a DUMB client and a DUMB server, but we also want to ensure the safety of our server.

1.1 Definitions:

Client: The DUMB client.
Server: The DUMB server.
Dummy Client: A alien client written in an attempt to find faults within the server.

1.2 Procedure:

We determined that this project could essentially be broken down into sets of functionalities. I.E. Each command has an opposite command, and we would like to test both commands at the same time.

1. First and foremost, we would like to have functional connection/disconnection. (2.1)
2. Next, creation and deletion of boxes. (2.2)
3. Followed by opening and closing of boxes. (2.3)
4. And lastly, putting and retrieving messages. (2.4)
5. Extreme edge cases/error handling. (2.5)

2 Testsuites

2.1 Connections

Prerequisite: Establishing connections.

1. Establish a connection to **Server**. Accept a "HELLO" from the **Client**, and respond with "HELLO DUMBv0 ready!". Make sure that both parties are content with these responses, and have acted accordingly.
2. Establish a connection to **Server** using **Dummy Client**. Since there is no "handshake" using "HELLO", the **Server** will reject the **Dummy Client**. This results in a special error that we have classified as *ER: INVCL*.
3. Quit. Close the connection with the **Client** sending "GDBYE", and determining the connection has closed. Then exit the **Client**.

2.2 Create/Delete

First, we want to make sure that creation and deletion of boxes is functional. For this, we go through the following test:

1. Create a box, then delete it. Attempt to open that same box. This should result in an error.
2. Attempt to delete a non-existant box. This should result in an error.
3. Create a box, then attempt to create another box under the same name. This should result in an error.
4. Give the create and delete commands a box name that is incompatible. This should result in an error.

Upon completion of this test, we can safely presume that our creation and deletion functions are working, disregarding cases where boxes contain messages. Next, we want to ensure the boxes can be opened and closed.

2.3 Open/Close

Assume that we already have multiple boxes created without error.

1. Open a box, then simply close it. Attempt to open the box again. This should not result in any errors.
2. Close a box without opening a box. This should result in an error.
3. Open a box, then have another instance of **Client** attempt to open the same box. This should result in an error.
4. Open a box, then have another instance of **Client** attempt to close the box. This should result in an error.
5. Open a box, then attempt to open a box without closing the previous box. This should result in an error.

Completion of this test finally allows us to test message creation and deletion.

2.4 Put/Next

Assume that we already have a box created without error.

1. Put a message in a box, and retrieve the message using next. This should not result in an error.
2. Retrieve a message using next without putting a message in a box. This should result in an error.
3. Put 3 messages into a box, retrieve a message, and put more messages in the box. Retrieve all messages. All messages should have correct order, and this should not result in an error.

Finishing up this test, we have proven 3 sets of functionality. We now piece them together and test more complex cases.
Ex: Open a box, put a message, close the box, quit, reconnect, open the same box, and retrieve the message.

2.5 Extreme cases/error handling

In these tests, we want to cover what we could not through our testing of each set of commands.

1. Give poor arguments to each command. (I.E. Not providing the correct box name, putting an empty message)
2. Using put and next when there is not a box open.
3. Deleting a box that is already open.
4. Deleting a box with messages.

3 Conclusion

We believe these testsuites described will prove that our **Client** and **Server** are not only extremely powerful, but very safe and secure. Through these series of testsuites, we have shown that our commands are highly functional and intelligent in regards to error detection and reporting.