**Test Plan**

**Client**

The client has many commands that it can send to the server:

1) quit → closes the connection between the server and the client

2) create → creates a new box

3) open → opens an existing box for the client

4) next → prints the next message in the box to the user

5) put → puts/stores a message in a message box

6) delete → delete a specific box

7) close → closes an open bos

The client tries to connect to the server for three times in case it fails the first two times. Once it connects, it receives messages from the server and based on these messages, it outputs non-protocol messages to the user. The client also has a help method that can help the user navigate all the commands. Most of the command functions were implemented using string manipulation methods that we have implemented ourselves, such as getSubString which return a subString if given the main string, and start and end indices. We also used few other functions from the string.h library such as strcmp and strstr to help us with processing the messages received from the server.

**Server**

The server has many tasks that have to be dealt with:

1) Listen for connections

2) Open a new thread per connection

3) Store all the shared data in a Mailbox System

4) Make sure that no threads are trying to read/write to shared memory simultaneously

5) Listen for the ctrl-c signal to properly free all memory and shut down the server connection

Listening for client connections was put in the main function because accept() will block until a connection is established. When a connection occurred, a thread will run with an initClient() function that had all the code to properly allow the client to read/write data to the Mailbox System.

**Data Structures used for Mailbox System**

The Mailbox System is a linked list of Message Box nodes. A Message Box node consists of 4 parts: The name of the message box, whether the message box is open or not, a pointer to the next message box, and also a pointer to a Queue. The queue is a linked list of nodes, where each node holds a character array to hold a message, and each node points to a next node. The queue system is used to store messages from the PUTMG and NXTMG commands in the proper order.

**Handling Synchronization and Freeing Memory**

Mutex’s are used in order to make sure that no two clients were reading or writing information to the Mailbox System at the same time. When the user presses ctrl-c to end the server, the signal is detected and calls an exitServer() function. All threads, mutexes, and the entire Mailbox System gets free()’d properly to ensure no memory leaks are present.

**Test Case 1**

In this test case we wanted to test for general functionality of creating, opening, putting, and getting a message from a message box.

**Client**

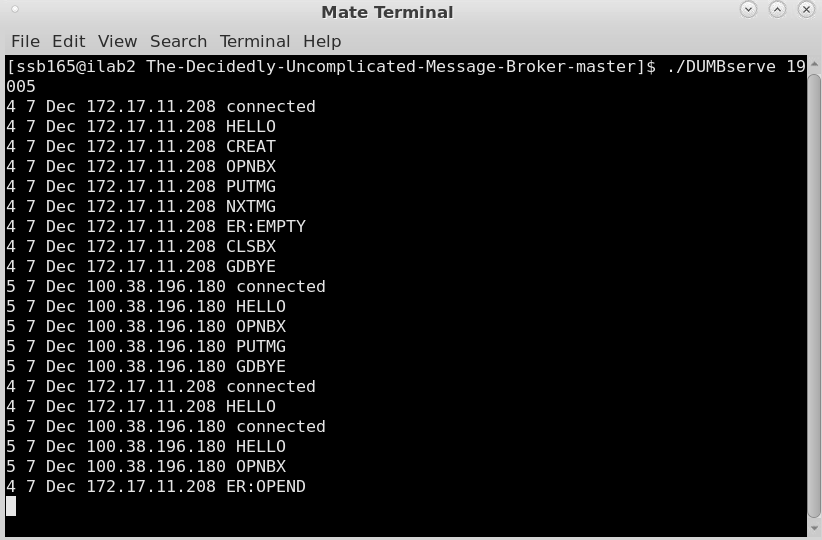
./DUMBclient [ilab2.cs.rutgers.edu](http://ilab2.cs.rutgers.edu/) 19005  
Success. You are connected.  
Please choose a command from the command list below:  
1. quit  
2. create  
3. delete  
4. open  
5. close  
6. next  
7. put  
create  
Okay, enter the name of the box  
create:> messagebox1  
Successfully created  
open  
Okay, open which message box?  
open:> messagebox1  
Successfully opened  
put  
Okay, enter the message you want to put in the box.  
put:> hello, there!  
Success. Message was put in the message box  
next  
hello, there!  
next  
Error. No messages left in this message box  
close  
Okay, close which message box?  
messagebox1  
Successfully closed  
quit  
Connection closed

**Server**

4 7 Dec 172.17.11.208 connected  
4 7 Dec 172.17.11.208 HELLO  
4 7 Dec 172.17.11.208 CREAT  
4 7 Dec 172.17.11.208 OPNBX  
4 7 Dec 172.17.11.208 PUTMG  
4 7 Dec 172.17.11.208 NXTMG  
4 7 Dec 172.17.11.208 ER:EMPTY  
4 7 Dec 172.17.11.208 CLSBX  
4 7 Dec 172.17.11.208 GDBYE

**Test Case 2**

In this test case, we wanted to test multi-connections within our server system. We repeated Test Case 1 on 2 clients, one run on linux, the other run on Windows. We creating a messagebox1 and opened messagebox1 on Linux. The Windows client then tried to open messagebox1, but got an ER:OPEND, because the Linux client already opened it first and did not close it. The screenshot of the server demonstrates these connections by looking at what the server outputted on the next page.



**Test Case 3**

With this test case we wanted to test the limit for putting a really really long message into a message box.

**Client**

open

Okay, open which message box?

open:> messagebox1

Successfully opened

put

Okay, enter the message you want to put in the box.

put:> kmplikhhpirbqnkxuiyjcqfpriddyertovsaxvlrfacmtvisrjjmphrvgwgnlyfwifjtuzhzcwperpcfrosfvobgrugdauxirlhdirbgxdcxvlvfhsqgzaaylmnqfcofbzaddyxohjiznujwyconvqsigjqcdvwvgqrtcppllmefqrkzwrfvstqjftatvltldxoagukmsrqrcvekoapipiihacsvbidtdlbjqojzihnnaxtxiejujkxinnzuvxcloxztxilcvregtmjnvastckjycuveeiugxnbstsdsbechlmzjojrprteedrrnnanzpltstdjxreuoqfjbbauwmxulfkpjgptkwpuxbkjdmmrymanrkjwqlvnrlvlbpgelgdhyuiyzscimupyvenclhqajchkzrpyosghostabvjmyppgrlxpnkmlasiedngkoopoodlknuzzvotwyrpkabsyuygjaxivmjulkuubihjvqgjuecdwlquunkvieahpaoquakczsopoocooqrneacvuubtcdsvjbogwbbzqnezjqjfihryzkeqvkbhxoaoorjxhwzahkasldskqgmhqjgvpaqpmfycmsajazizbbhcbidjlzpjmuiszmurqnowywpwacxclhigsjdvccdynuzyjncrhvqlyjhtedaiewgybkmiyjtonprtuvnafagwpbdsjqsstvqisyxjqrlyscvinnfohtavqdocijsovabvtgzturppygyplqojcfdywmbakxmgytpwdfoiydpctnnrwerksmnqfc

Your message was trimmed

Success. Message was put in the message box

next

hey, world

next

kmplikhhpirbqnkxuiyjcqfpriddyertovsaxvlrfacmtvisrjjmphrvgwgnlyfwifjtuzhzcwperpcfrosfvobgrugdauxirlhdirbgxdcxvlvfhsqgzaaylmnqfcofbzaddyxohjiznujwyconvqsigjqcdvwvgqrtcppllmefqrkzwrfvstqjftatvltldxoagukmsrqrcvekoapipiihacsvbidtdlbjqojzihnnaxtxiejujkxinnzuvxc

Here we see that the user inputted an 800 character long string, however the client notifies the user that their message has been trimmed and saved. When the user asks for the message back, they get 255 characters. This is because the limit of a message is 256 bytes including the null terminator, 255 bytes free to set by the user.

**Test Case 4**

This test case was created to make sure that 2 clients cannot try to open/delete/close each other’s message boxes. The client output is pasted below for client 1. Note that client 2 has messagebox2 open and messagebox3 does not exist.

**Client**

open  
Okay, open which message box?  
open:> messagebox1  
Successfully opened  
close  
Okay, close which message box?  
messagebox2  
Error. This box is not currently open  
close  
Okay, close which message box?  
messagebox3  
Error. This box is not currently open  
delete  
Okay, enter the name of the box  
messagebox1  
Error. This box is currently open  
delete  
Okay, enter the name of the box  
messagebox3  
Error. Box doesn't exist  
close  
Okay, close which message box?  
messagebox2  
Error. This box is not currently open  
close  
Okay, close which message box?  
messagebox1  
Successfully closed  
delete  
Okay, enter the name of the box  
messagebox1  
Successfully deleted  
delete  
Okay, enter the name of the box  
messagebox2  
Error. This box is currently open

**Test Case 5**

This test case was created to make sure that the program doesn’t allow for creating boxes with names less than 5 characters or more than 25 characters. The client output is below.

./DUMBclient 127.0.0.1 8989

Success. You are connected.

Please choose a command from the command list below:

1. quit

2. create

3. delete

4. open

5. close

6. next

7. put

create

Okay, enter the name of the box

create:> m

Error. The name must be 5 to 25 characters long and start with an alphabetic character

create

Okay, enter the name of the box

create:> myboxmyboxmyboxmyboxmyboxx

Error. The name must be 5 to 25 characters long and start with an alphabetic character

create

Okay, enter the name of the box

create:> mybox

Successfully created

**Test Case 6**

This test case was created to make sure that the user cannot open a new box before they close the current open box. The client output is below.

Okay, enter the name of the box

create:> mybox1

Successfully created

create

Okay, enter the name of the box

create:> mybox2

Successfully created

open

Okay, open which message box?

open:> mybox1

Successfully opened

open

Okay, open which message box?

open:> mybox2

Error. You have a box opened

close

Okay, close which message box?

mybox1

Successfully closed

open

Okay, open which message box?

open:> mybox2

Successfully opened

**Test Case 7**

This test case was created to make sure that the user cannot delete an opened box or a box that has messages(not empty). The client output is below.

create

Okay, enter the name of the box

create:> mybox

Successfully created

open

Okay, open which message box?

open:> mybox

Successfully opened

put

Okay, enter the message you want to put in the box.

put:> hi

Success. Message was put in the message box

put

Okay, enter the message you want to put in the box.

put:> hello!

Success. Message was put in the message box

delete

Okay, enter the name of the box

mybox

Error. This box is currently open

close

Okay, close which message box?

mybox

Successfully closed

delete

Okay, enter the name of the box

mybox

Error. This box is not empty

next

Error. Either the message box not open or doesn't exist

open

Okay, open which message box?

open:> mybox

Successfully opened

next

hi

next

hello!

next

Error. No messages left in this message box

close

Okay, close which message box?

mybox

Successfully closed

delete

Okay, enter the name of the box

mybox

Successfully deleted