FINAL REPORT

SJSU SAN JOSÉ STATE UNIVERSITY

Distributed Airline Reservation System

Ifunanya Nnoka
With the guidance of Professor Rod Fatoohi

Contents

ABSTRACT

Airline reservation systems are basically used widely for managing flight schedules, seats, costs and maintaining the clients in an easy way. An airline's direct distribution works within their own reservation system, as well as pushing out information to the GDS. A second type of direct distribution channel are consumers who use the internet or mobile applications to make their own reservations.

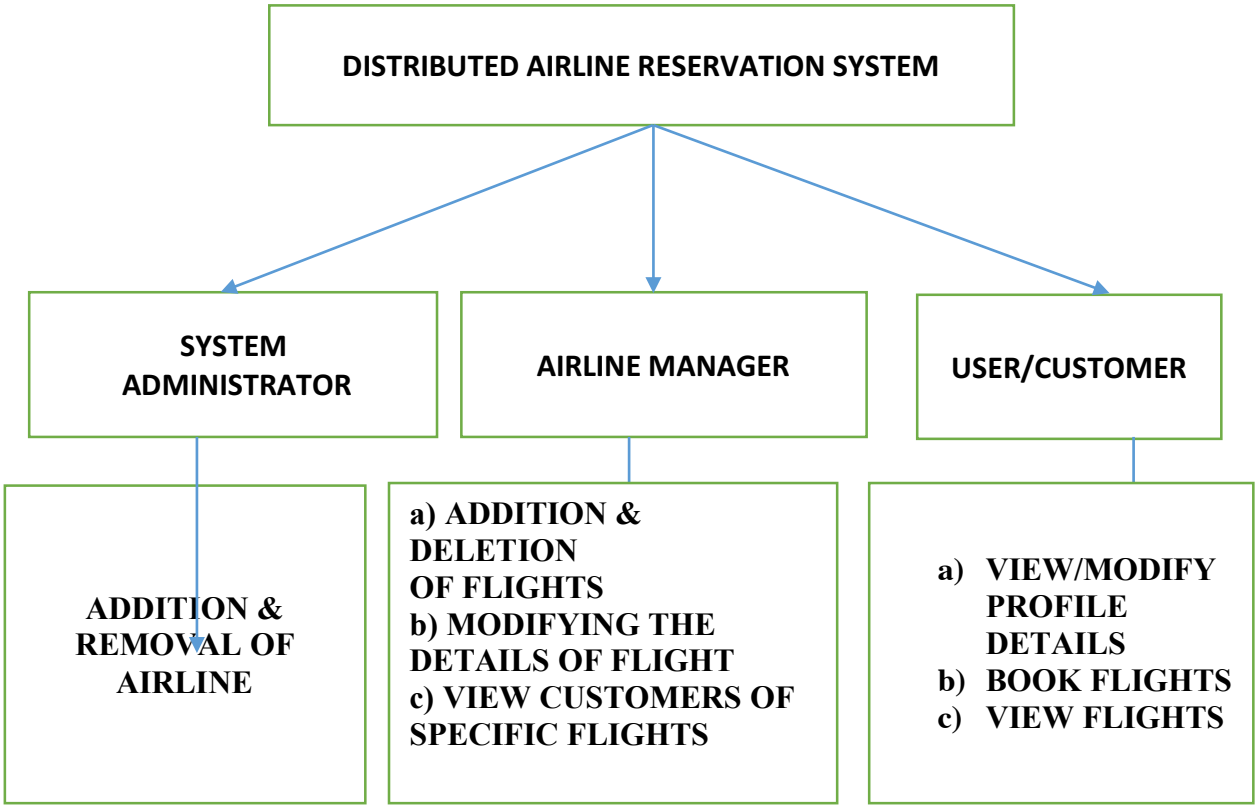
The modern airline reservation helps to simplify airline management tasks and service customer needs from the time of initial reservation through completion of the flight.

Customers who wish to travel in a flight have a wide variety of airlines and a range of timings to choose from. There are a lot of discounts and luxuries available for the customers in addition to the luxury and in-flight entertainment in various flights to attract potential customers.

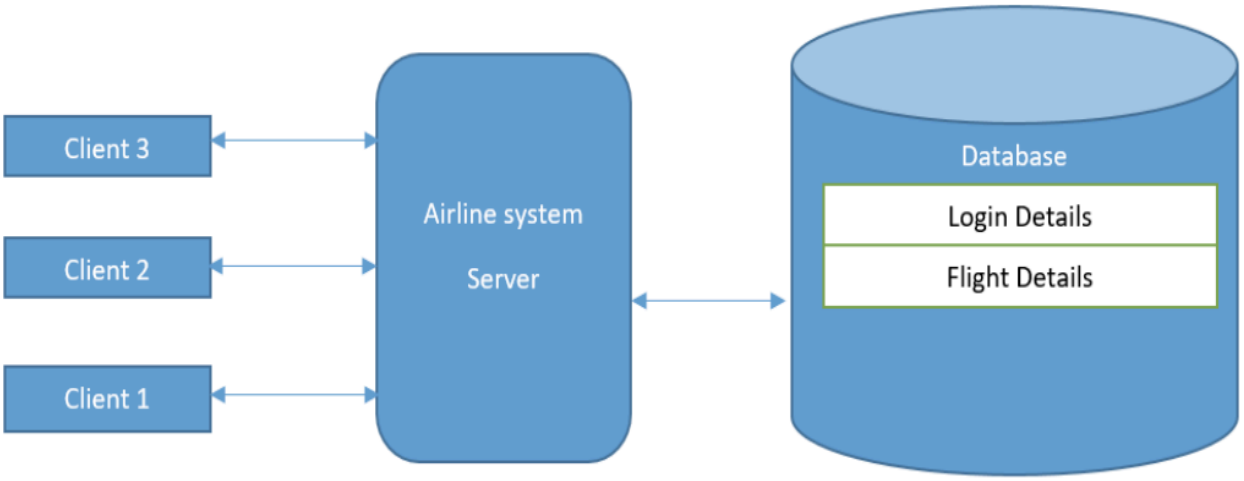
With the development of internet over several years, airline reservation is commonly done using websites of various airlines. This is the most hassle free and easiest way to make reservations and takes very less amount of time. Previously airlines owned their own reservation systems with travel agents subscribing to them. Today, the Global Distribution system are run by independent companies with airlines and travel agencies as major subscribers.

Our project mainly works with the data present in the database. It reads the data and presents it to the user, allows the user to make various choices required for reservation. The user gets all the data required to make a decision for the flight like time, number of flights, cost etc.

PROJECT DESIGN



PROJECT ARCHITECTURE



DATABASE IMPLEMENTATION

Sqlite3 library is the chosen database implementation. This library gives us the ease of creating and manipulation of databases in C programming language.
SQLite is a relational database management system contained in a C programming library. In contrast to many other database management systems, SQLite is not a client–server database engine. Rather, it is embedded into the end program.

SQLite engine has no standalone processes with which the application program communicates. Instead, the SQLite library is linked in and thus becomes an integral part of the application program. The library can also be called dynamically. The application program uses SQLite's functionality through simple function calls, which reduce latency in database access: function calls within a single process are more efficient than inter-process communication. SQLite stores the entire database (definitions, tables, indices, and the data itself) as a single cross-platform file on a host machine. It implements this simple design by locking the entire database file during writing. SQLite read operations can be multitasked, though writes can only be performed sequentially.

EXPLAINATION OF MODULES

MODULE 1- SYSTEM ADMINISTRATOR

The main function of the system administrator is the management of airline users. All airlines are required to be registered by the system admin before they can access or modify any flight information in the database using the server. The system admin has the sole authority of adding an airline representative, thereby giving them full access to the database, the system administrator also has the sole authority of deleting an airline user thereby denying them access to the database, the system admin can also modify the airline users, situations might arise whereby an airline user might need to change their user ID or might need to be granted a different kind of access to the database. All of these functionalities are responsibilities held only by the system administrator.

MODULE 2 - AIRLINE MANAGER

The airline manager implements various functionalities required as standard airline protocol. These include a server system that enables an airline representative (client) to securely access an airline database to perform various task necessary for an airline to function properly. Some of these functionalities include an ability to add and delete airlines, access to information on flight availability and booked seats, and an ability to modify flight details and delete flights. The airline manager is responsible for ensuring customers are able to retrieve the most up to date information on flights before booking. An airline manager can be looked at as the “vendor” server whereby the major task of the users of the application is to add all the flights that are open to customers for booking, and also to periodically be able to modify details of the flight when required, in order for the database/customer server module to be able to service a customer effectively.

MODULE 3 - CUSTOMER

The customer server module focuses on creating an interface between the costumer and airline manager. The subsequent goal of this module is to have all the necessary customer information stored in the database, and to be able to successfully help a customer book a desired flight using all of the flight information stored on the database by the airline manager. This module assists a costumer with making flight reservations by extracting flight information from the airline database which align with the customer’s request, this information is then buffered to the client. The client has the option of booking a flight, after which the client is given a transaction number which is used as a primary key to properly allocate all of the client’s information on the database. Client can login and/or use the transaction number to access booking details. Finally, the

customer server module then updates the airline manager with the information of the seat booked, this is done by updating the number of seats available on the flight in the database.

PROJECT REQUIREMENTS

This project aims to service customers who want to make flight reservations easily. The customer will not have the hassle of making reservations using travel agencies. All that is required of a user is an internet connection, and they can make reservations in a short period of time. Now-a-days with the booming of world wide web, almost everyone is connected using the internet. This makes it incredibly easy for most customers to have access to our flight reservation system.

Customers are presented with information on available flights routing through their desired source and destination, such information includes details of number of seats available, cost of flight, date of departure and date of arrival. When the customer decides the flight which they wish to travel with, an option for booking is made available. The customer is required to enter some basic information that will help identify them during travel as well as help them retrieve their reservation information at a later time. All of these information is stored in a database for later use. If the user decides to cancel the booking, the user can do so as well.

We also have an airline manager module which focuses on making sure clients (customers) get the most up to date information on flights available. The airline user/ manager can add the flights into the database. These flights can be viewed by the customer who wishes to make a booking. The airline manager can modify the flight details and will be have access to a periodic update on booked seats. This basically serves as a backend to the customer interface i.e., the modifications done by the airline manager is shown to customer who makes reservations.

The system administrator adds the airline manager who then adds the flight details into the database. The admin needs to login using his credentials in order to add airline user for security reasons.

TEST PLAN

SYSTEM ADMINISTRATOR

SCOPE:

The goal for this sub task was to implement a system administrator server algorithm that poses as an interface between the system admin (client) and database. System admin is the most privileged user of the system. He can add, delete and modify an airline professional to the system.

OBJECTIVES:

Test scenarios include:

- User should be able to add an Airline user to the system.
- User should not be able to add a duplicate user in the system.
- User should be able to modify a user in the system.
- User should be able to delete a user in the system.
- User should not be able to delete a user not there in the system.
- User should be able to view all the users in the system.

REQUIRED RESOURCES:

- Linux/POSIX Operating System
- IDE and integrated software development tools
- Sqlite3 Library installation
- Files: errexit.c, passivesock.c, passiveTCP.c and tcpserver.c

MODULE TESTING:

Test Cases:

Test Case #	Description	Input data	Steps	Expected result
1	User should be able to login as a system admin	Username: System_Admin, Password: iamadmin	1. Login to the system using system admin credentials	User should be able to login to the system as a system admin
2	User should not be able to login with wrong username	Username: Admin, Password: iamadmin	1. Login to the system using system admin credentials	An error message should be displayed and the login screen should be displayed again
3	User should not be able to login with a wrong password	Username: System_Admin, Password: admin	1. Login to the system using system admin credentials	An error message should be displayed and the login screen should be displayed again

4	User should be able to view 5 options: add, modify, view, delete user and to exit the system.	Username: System_Admin, Password: admin	1. Login to the system using the system admin credentials	Screen with the below options should be displayed: 1. Add a user 2. Modify user details 3. View a user 4. Delete a user 5. Exit
5	User should be able to Add a user.	User should be logged into the system	1. Input value '1' from the screen option.	User should obtain a screen with the following prompt: Please add user in the below format <username> <password>
			2. Input username: "Turkish" and password: "iamturkish"	The user details should be added to the database
			3. Input value '3'	All the users should be listed in the screen including the one recently added
6	User added by system admin should be able to login to the system	Username: 'Turkish', Password: 'iamturkish'	1. Login to the system using the credentials for 'Trukish' username	The user should be able to login to the system
7	User should be able to modify the user password	System_Admin should be logged into the system	1. Input value '2' from the screen option	User should obtain a screen with the following options: Please enter in the below format: <username> <password>
			2. Input username: Turkish password: turkish	The database should be updated with the details
8	User should be able to delete a user	User should be logged into the system	1. Input '4' as an option	"Please enter the <username> should be displayed.
			2. Input "Turkish"	Modified successfully should be displayed
9	User should be able to view the users in the system	User should be logged into the system	1. Input '3' as an option	All the users in the system should be displayed
10	User should be able to exit the system	User should be logged into the system	1. Input '5' as an option	"Have a nice day!!!" should be displayed

AIRLINE MANAGER

SCOPE:

The goal for this sub task was to implement an airline server algorithm that poses as an interface between the airline professional (client) and airline database. The ultimate purpose of the airline manager module is to periodically relay up-to-date flight information to the airline customer via a shared database. To meet the requirements of a well implemented airline manager, an airline server application should have the ability to access the airline database and relay information to the airline representative (client), and should also be able to retrieve information from the airline representative to modify flight information on the airline database as long as the modification is requested by a system-admin verified user.

OBJECTIVES:

Test scenarios include:

- An attempt to add a flight to the database using an authentic airline ID.
- An attempt to add a flight to the database using an incorrect airline ID.
- An attempt to view information on a flight that exists in server using flight number.
- An attempt to view information of a flight that does not exist in server using flight number.
- An attempt to modify flight information of a flight that exists in server using flight number and airline ID.
- An attempt to modify flight information of a flight that does not exist in server using flight number and airline ID.
- An attempt to delete a flight that exists in server using flight number and airline ID.
- An attempt to delete a flight that does not exist in server using flight number and airline ID.
- An attempt to view all flights existing and an ability to see all of the flight information in the database.
- An attempt to view all flights of a specific airline that exists in server using airline ID.
- An attempt to view all flights of a specific airline that does not exist in server using airline ID.

REQUIRED RESOURCES:

- UNIX/POSIX Operating System
- IDE and integrated software development tools
- Sqlite3 Library installation
- File: tcpserver.c
- File: airlinemain.c
- File: passivesock.c
- File: passiveTCP.c
- File: errexit.c
- File: sqlite3.c
- File: sqlite3.h
- File: sqlite3ext.h
- File: tcpclient.c

MODULE TESTING:

Test Criteria:

FUNCTION	INPUT	PASS	FAIL
Add a Flight	flight name matching airline ID	• Server requests for all flight details • Final message: "Flight {airline name} added to server!" • Server returns the flight Summary below	If final message: "Flight {airline name} added to server!" Is not received then test fails
	flight name and not matching airline ID	Incorrect name for user airline	If server requests for flight number and subsequent flight information, then the test fails.
View a Flight	Flight name that exists in server	Server returns message below: Search Result: NAME: {flight number} UID: {airline ID} AIRLINE: {Airline name} SOURCE: {departure location} DESTINATION: {arrival} PRICE: {flight costs} TOTAL_SEATS: {total seats} OPEN_SEATS: {number of available seats} DATE_DEPATURE: {date/time of departure} DATE_ARRIVAL: {date/time of arrival}	Server returns: "We're sorry that flight does not exist on our server. Enter y to search more fields, n to exit."
	Flight name that does not exist in the database	Server returns: "We're sorry that flight does not exist on our server. Enter y to search more fields, n to exit."	If pass message isn't received, then test fails.

Modify a Flight	Flight name that exists And authorized users	Server returns options: Enter to modify: <<:: 1. All ::>> <<:: 2. Source ::>> <<:: 3.Destination ::>> <<:: 4. Price ::>> <<:: 5. No of Seats Available ::>> <<:: 6. Date of Departure ::>> <<:: 7. Date of Arrival ::>> After new fields have been modified server sends final message with modified flight information: "Updated Flight Details for {Flight name}:"	Server returns: SQL error: {message}Enter y to modify more fields, n to exit."
	Flight name that does not exist on server And unauthorized users	"We're sorry that flight does not exist on our server"	Server returns: SQL error: {message}Enter y to modify more fields, n to exit
Delete a Flight	Flight name that exists in server and authorized users	"Flight {flight number} deleted successfully! Enter y to delete more flights, n to exit"	error: {message} "Enter y to delete another field, n to exit."
	Flight name that does not exist in server And unauthorized users	You are not authorized to delete this flight. <:: 1. Go Back ::>> <<:: 2. Exit 'Delete Flight' ::>>	Server displays: "ERROR reading from socket"
View All Flights	Case: Flights exists on server	Server sends: "Search Results: All Flights" (followed by a list of all flights in the database)	"No flights available at the moment. Please try again later!"
	Case: No flights exists on server	"No flights available at the moment. Please try again later!"	Server displays: "ERROR reading from socket"
View all flights of a Specific Airline	Case: flights from airline exist on server	Server sends: "Search Results: All Flights from {airline name}" (followed by a list of all flights in the database)	"We're sorry there are no records of flights from {airline name} on our server. Enter y to search a different airline, n to exit."
	Case: No flights from airline exists on server	"We're sorry there are no records of flights from {airline name} on our server. Enter y to search a different airline, n to exit."	Server displays: "ERROR reading from socket"

INTEGRATION TESTING

The airline manager features must also be tested as an integrated system with the system administrator module and customer services module. As an entity in the integrated system, the airline manager is required to comply with the restrictions put on the system by the system administrator. Server must not give access to users without being authenticated via user name and password. Airline server must also be able to retrieve updated flight seat information after a customer user has booked a seat.

Test Criteria:

FUNCTION	INPUT	PASS	FAIL
Flight Manager Authentication	Correct <Username><Password>	"Logged in Successfully"	Otherwise failed
	Incorrect <Username><Password>	"Incorrect Username or Password"	Otherwise failed

The conditions above must be met before Functions in the module testing can be held as true.

FLIGHT MODULE LIMITATIONS AND RISKS:

One of the major limitations and risks I came across during the implementation of this module was the issue of server buffer size/memory allocation, more specifically the buffer size that was required when sending data to the TCP client, and size limitations enforced by my computer system. In test cases when a flight manager would like to view all flights recorded on the server or view all flights of a specific airline, there becomes a crucial need for a large enough buffer size. While designing the algorithm this was a major hitch. Initially we had added up to 20 flights to the database, but when it came to testing the "view all flights" and "view all flights from a specific airline" implementation, we found that the server would hang. Being that this is my first socket programming class, I was a bit inexperienced with servers so it took me a while to figure out why the application stopped working. Nonetheless, after a tremendous period of debugging I came to realize that the '1001' byte server buffer was not large enough to handle all of the data that the database was throwing at it, so ultimately it froze. I increased the size of the buffer of both the client and the server to '2001' bytes and yet again it wasn't large enough, so I went big and tried using a '4096' sized buffer and somehow that was 'too big' because the process rejected it (I still intend on looking into the reason for that). Finally, I settled for a '2700' sized byte which can currently handle about 14 flight details comfortably, any amount beyond that might cause some issues. In short, there is a lot of emphasis on the importance of buffer/memory allocations, so while running the program testers should be aware that there is a limitation on the amount of flight one can retrieve at a time from the TCP server.

CUSTOMER USER FEATURES

SCOPE:

The function of the customer server module is to pose as an interface or "middle-man" between the customer and the airline manager. The airline manager in a sense "advertises" its flights to the customer (client) by storing all of the flight information on a shared database which is accessible by the customer server application, the customer then chooses whether or not to book a seat on a flight. After a customer (client) has booked a seat, the customer server is then left with 2 major tasks: (1) To relay the information of booked seats to the airline manager, by updating the record on the database of the quantity of seats booked. (2) The customer server is also required to record the customer (client) information on the database so that the customer can easily pull up their flight reservation detail, this is achieved by ensuring that the customer registers with the server before booking a flight, and also ensuring that each reservation made has a transaction number associated with it. This way for a customer to retrieve their booking information they need only to have their "username and password" and/or the "transaction number" associated with the flight reservation.

With all that being said, to meet the requirements of a well implemented customer user application, the server application should have the ability to register a user, securely log in registered users, retrieve flight information from the database and return to user, give users the option of booking a flight, assign a user a transaction no. after a booking has been confirmed, users should be able to retrieve their booking information with a transaction number and {username, password}. The customer module must also be able to update flight information on the data base, particularly no. of available seats should be updated.

OBJECTIVES:

Test scenarios include:

- An attempt to register new users and upload their information to the database.
- An attempt to register with a username that has been already used.
- An attempt to login users with their correct username and password.
- An attempt to login users with their username and incorrect password.
- An attempt to login users with a username that doesn't exist.
- An attempt to search for flights through a source and destination that matches a flight in the server.
- An attempt to search for flights through a source and destination that does not match any flight in the server.
- An attempt to book a flight with seat number that hasn't been taken.
- An attempt to book a flight from a collection of multiple flights.
- An attempt to book a seat number that has been already booked.
- An attempt to retrieve reservation details with a transaction number that matches user account.
- An attempt to retrieve reservation details with a transaction number that is assigned to a different user account.
- An attempt to retrieve reservation details with a transaction number that does not exist in server.
- An attempt to retrieve reservation details by logging in with username and password.
- View number of available seats after a client has booked a seat.

REQUIRED RESOURCES:

- UNIX/POSIX Operating System
- IDE and integrated software development tools
- Sqlite3 Library installation
- File tcpserver.c
- File customer_services.c
- File passivesock.c
- File passiveTCP.c
- File errexit.c
- File sqlite3.c
- File sqlite3.h
- File sqlite3ext.h
- File tcpclient.c

MODULE TESTING:

Test Criteria:

FUNCTION	INPUT	PASS	FAIL
Customer Registration	New unassigned username and password	"{username} you have been successfully registered. Press Enter to login."	"Username {username} not available. Please choose a different username."
	Already assigned username and password	"Username {username} not available. Please choose a different username."	"{username} you have been successfully registered. Press Enter to login."
Customer Login	Correct username & password pair	"Welcome {username}! What would you like to do? <<:: 1. SearchFlights::>> <<::2.Viewbookings::>>"	"Incorrect password!"
	Correct username & wrong password pair	"Incorrect password!"	"Welcome {username}! What would you like to do? <<:: 1. SearchFlights::>> <<::2.Viewbookings::>>"
	Username that does not exist in server	username {username} does not exist in our server!! Enter '1' to use a different username '2' to exit	"Welcome {username}! What would you like to do? <<:: 1. SearchFlights::>> <<::2.Viewbookings::>>" OR "Incorrect password!"
Search flights	Flight that matches source and destination exists in server	<<<<++++FLIGHT{no}.++++>>>> NAME: {flight no.} AIRLINE: {airline} SOURCE: {source} DESTINATION: {destination} PRICE: {price} OPEN SEATS: {#of open seats} Date of Departure: {date/time} Date of Arrival: {date/time} Enter 1. :::>> Book Flight 1 Enter 2. :::>> Book Flight 2 . . Enter #. :::>> Search More Flights	"No flights available through that route. Search more flights? Enter <<:: 1. Search Flights ::> <<:: 2. Exit ::>>"

	Flight that matches source and destination does not exist in server		"No flights available through that route. Search more flights? Enter <<:: 1. Search Flights ::> <<:: 2. Exit ::>>"	No message or returns flight that does not match
Book flights	Select flight	Enters number in the search list corresponding to desired flight	"Seats already booked: #, #, #, #, Choose a seat number to book btw seat 1 - seat {total seats}. Exclude booked seats."	If sever returns booking flight details for a different flight
	Seat number that hasn't been booked		Enter <<:: 1. Complete Transaction ::>> <<:: 2. Cancel ::>>	"Seats already booked: #, #, #, #, Choose a seat number to book btw seat 1 - seat {total seats}. Exclude booked seats."
	Seat number that has been booked		"Seats already booked: #, #, #, #, Choose a seat number to book btw seat 1 - seat {total seats}. Exclude booked seats."	Enter <<:: 1. Complete Transaction ::>> <<:: 2. Cancel ::>>
	Enters 1 to complete transaction		"Congratulations! Your flight reservations have been made. Your Order Number is #. Press 1 to view your flight reservations."	"Enter 1 to search more flights or any character to exit"
	Enters 2 to cancel		"Enter 1 to search more flights or any character to exit"	"Congratulations! Your flight reservations have been made. Your Order Number is #. Press 1 to view your flight reservations."
View reservation with a transaction no.	View reservation with a transaction no. belonging to user		"Transaction No.: # Flight No.: {name} First Name: {user's} Last Name: {lastname} Airline: {airline} From: {source} To: {destination} Seat No.: # Date of Depature: {date/time} Date of Arrival: {date/time} Price: {price} <<:: 1. Exit ::> <<:: 2. Search Flights ::>>"	"Order number does not match your records" OR "Order number not in ours records. Please Enter a valid order number."
	View reservation with a transaction no. belonging to a different user		"Order number does not match your records"	"Transaction No.: # Flight No.: {name} First Name: {user's} Last Name: {lastname} Airline: {airline} From: {source} To: {destination} Seat No.: # Date of Depature: {date/time} Date of Arrival: {date/time} Price: {price}

			<<:: 1. Exit ::>> <<:: 2. Search Flights ::>>" For different user
	View reservation with a transaction no. that does not exist	"Order number not in ours records. Please Enter a valid order number."	Sql error message
View reservation with username and password	View reservation with a username and password for user who has made a reservation	"Transaction No.: # Flight No.: {name} First Name: {user's} Last Name: {lastname} Airline: {airline} From: {source} To: {destination} Seat No.: # Date of Depature: {date/time} Date of Arrival: {date/time} Price: {price} . . . "Transaction No.: # Flight No.: {name} First Name: {user's} Last Name: {lastname} Airline: {airline} From: {source} To: {destination} Seat No.: # Date of Depature: {date/time} Date of Arrival: {date/time} Price: {price} Shows all bookings if more than 1	"Looks like you have made no reservations with us"
	View reservation with a username and password for user who has made no reservation	"Looks like you have made no reservations with us"	"Transaction No.: # Flight No.: {name} First Name: {user's} Last Name: {lastname} Airline: {airline} From: {source} To: {destination} Seat No.: # Date of Depature: {date/time} Date of Arrival: {date/time} Price: {price} . .
Cancel Reservation	Transaction number that belongs to user		
	Transaction number that does not belong to user		

INTEGRATION TESTING

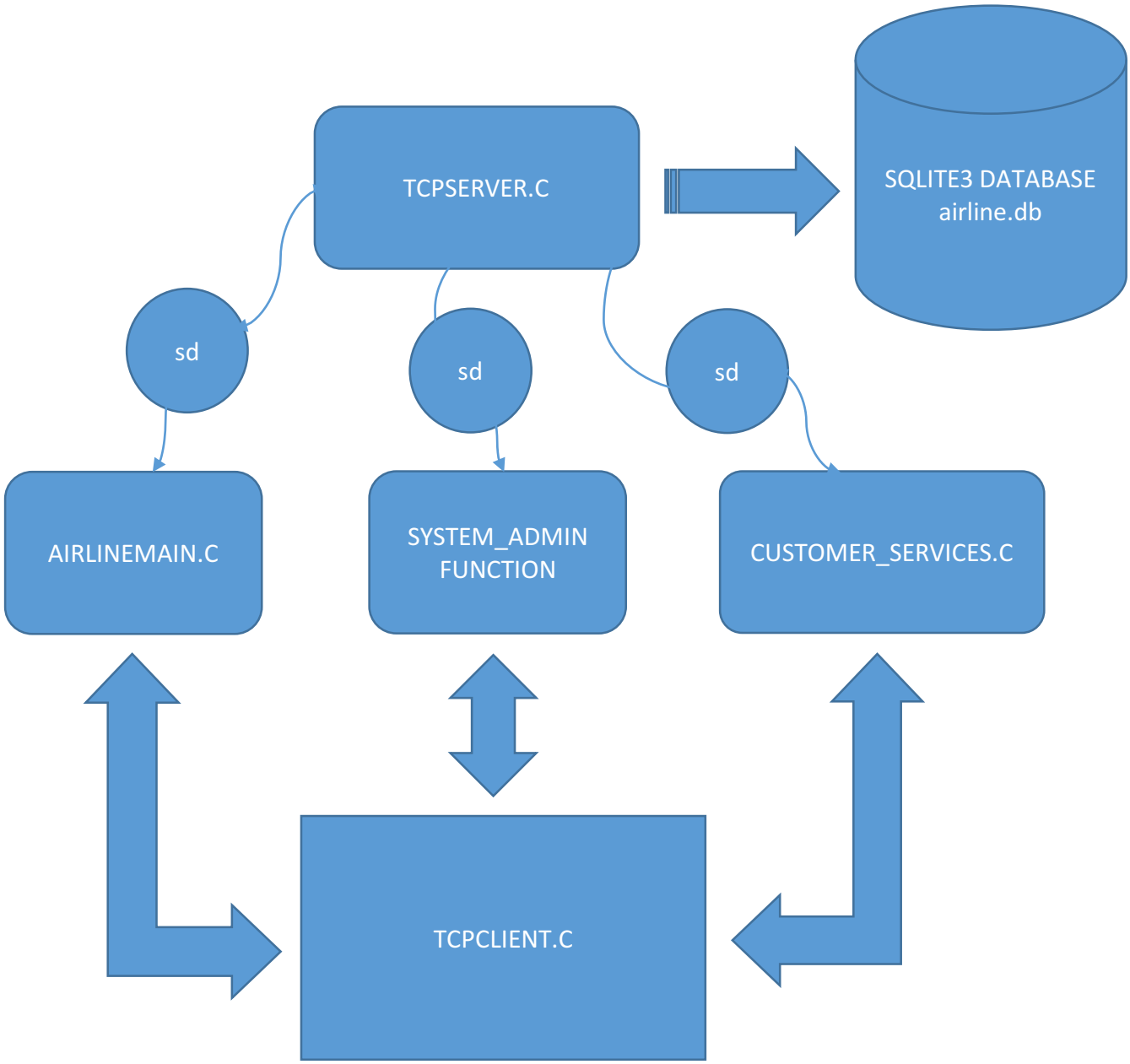
The customer server module as a standalone module functions as it would function in the integrated system as long as it has access to the airline database.

CUSTOMER MODULE LIMITATIONS AND CHALLENGES:

One of the major limitations and risks we came across during the implementation of this module was yet again the issue of memory and buffer size. In test cases when a customer would like to view transactions that they have made, there becomes a crucial need for a large enough buffer size. For example, during testing I made multiple transactions on my account, when it became time to view all of my transactions the server would freeze due to the large data size it had to send across. I had about 12 -15 transactions made, At "1001" bytes it was impossible for me to retrieve all of that information at the client side. Now, ideally most customers wouldn't have more than one or two reservations at a given time, I had about 12 which wasn't realistic. But it reminded me yet again about the importance of memory.

Another challenge we came across was how to implement a technique for the customer to be able to choose from multiple flights, this seems a bit straight forward, but the difficulty lies in the way in which sqlite3 calls back data from the database. While we figured that we could strcmp() the flight name with the arguments passed by the database and count how many arguments passed and hence number the flights that way, it still proved volatile. Ultimately, we had to assign a standard of only 6 bytes to a flight name so that when we strcmp() and extract the names into a buffer we could divide it by 6 to get the number of flights and thus number them, that way the client can correctly make a choice and the server can correctly detect what flight was selected.

PROJECT IMPLEMENTATION AND SCREENSHOTS



Server and Client Call Flow.

NETWORK PROTOCOLS AND STRATEGIES

Network protocols and strategies implemented in the distributed airline system is TCP. Why TCP? TCP is a connection oriented protocol. This protocol follows a three-way handshake to establish a connection between client and the server. Once the client server connection is established, the client and server can exchange messages. Since the system involves critical transaction we cannot use UDP protocol which is a connectionless protocol.

Server implementation follows the design of Pre-threaded multi server. Fifty threads for server are created and kept in a pool of listening servers. With each client request one server thread is assigned to service the server. Once the client is served and it exits, the thread returns to the pool and is in listening state. Hence our system is capable of serving fifty clients simultaneously.

Semaphore locks are implemented to avoid the access of critical section by different clients. This is done to prevent overwriting of data into the database. No two thread should be in the same critical section of the code. This prevents the concurrent access of database invalidation of the data due to overwrite. Semaphore locks are implemented using semaphore library provided by C. We use sem_t data type to declare a semaphore. Sem_init is used to initialize semaphore value. It also defines the limit of semaphore variable. Sem_wait() is used before entering the critical section. Sem_post() is used after exiting the critical section.

DATA ENCRYPTION AND USER INTEGRITY:

Data Encryption is a very important aspect of our civilization. There is always a need for data integrity and confidentiality. For our Module we found it best to implement an encryption algorithm to ensure data confidentiality when communicating between client and server, and for this reason we have implemented a simple 25bit Encryption algorithm for sending and receiving data between server and client. We also need to ensure data integrity and user authentication, and for this reason we have also implemented a message digest hashing algorithm, MD5 to authenticate users and store sensitive login credentials.

User Authentication and Data Integrity:

To verify users and protect their account, we have implemented MD5 cryptographic functions into our program. MD5 is a powerful hashing algorithm that maps an arbitrary long piece of plaintext to a fixed-length string of 128 bits, this string is known as its message digest (analogous to a fingerprint). Due to its one-way function characteristics, it is computationally impossible to compute the plaintext given the message digest, i.e given MD(P) it is computationally impossible to compute P. This characteristics of hashing algorithms is possibly the most important one. A string of text will always be mapped to a specific message digest when hashed. Based on these facts, during for example, customer registration we can safely retrieve a username and password and store them as a message digest, thereby preventing those with access to the database from retrieving a user's login information.

```
char *hash_gen_md5(const char *field, int len) {
    int i;
    MD5_CTX p;
    unsigned char digest[MD5_DIGEST_LENGTH];
    char *hash = (char*)malloc(33);

    MD5_Init(&p);
    while (len > 0) {
        if (len > 512) {
            MD5_Update(&p, field, 512);
        } else {
            MD5_Update(&p, field, len);
        }
        len -= 512;
        field += 512;
    }
    MD5_Final(digest, &p);
    for (i = 0; i < sizeof(digest); ++i) {
        snprintf(&(hash[i*2]), MD5_DIGEST_LENGTH*2,
                 "%02x", (unsigned int)digest[i]);
    }
    return hash;
}
#endif /* customer_h */
```

The function above generates a message digest for a given string of text with arbitrary lengths and returns its hash value, a 32-byte hexadecimal value which suffice as a fingerprint for the given text. For our project we use compute the hash values of user credentials during registration and store them in the database. Upon login, the computed hash values of user credentials entered at login should always match the hash recorded during registration.

Upon, registration the “gen_hash_md5” function is called to compute the message digest of the flight user’s password before the account is finally created.

```
    paswdd_hash = gen_hash_md5(password,strlen(password));
    asprintf(&query, "insert into Flight_Users(name,password,airline,uid)"\
    "values('%s','%s','%s',%s);",name,paswdd_hash,airline,uid);
    rc = sqlite3_exec(db, query, 0,0, &zErrMsg);
```

```
sqlite> select * from Flight_Users;
name                password                           airline      UID
------------------  ---------------------------------  -----------  ----
System_Admin        a41acc7effe601de1dc2099a4e2fdd7c   N/A          0
klm_user1           2b774a335ea09a3c6bb7bd38216926cf   KLM          100
delta_user1         025f20f41a336fae7b66f2fe95b27770   Delta        200
emirates_user1      d09fe287bac7a3bff254549ff90d887d   Emirates     300
united_user1        aeac9dbe2af798d5c6390bd86d782bfa   United       400
arik_user1          e85218012f314a3beaffffb6eb130eff   Arik         500
Lufthanza_user1     b40d9399785c77c41cd5784319291a69   Lufthanza    600
jetair_user1        6296293fc17fb34c22176928594a712c   JetAirways   800
britishair_user1    bda51853df0e170885d321d0760aab29   BritishAirw  700
klm_user2           2b774a335ea09a3c6bb7bd38216926cf   KLM          100
delta_user2         025f20f41a336fae7b66f2fe95b27770   Delta        200
```

Shown above, the slots password in the flight_users account table is filled with the corresponding 32-bit hexadecimal hash values of the user’s password.

So why MD5? As opposed to storing an encrypted login information and then decrypting before use? Well because MD5 focuses more on data integrity and not confidentiality. Since the password does not need to be known by anyone but the user, there is no need to go through the computation of encrypting and decrypting every time we need to access the information. Since, a message digest of a text will always compute to the same hex value, it is more efficient to store sensitive data as a hash value and then compare with the hash of the user during login. This way password stored in the database will never be able to be decrypted. MD5 is also efficient in the sense that, no matter what size of text you input to the algorithm it will always produce a message digest of 32-Hex bytes.

Data Encryption:

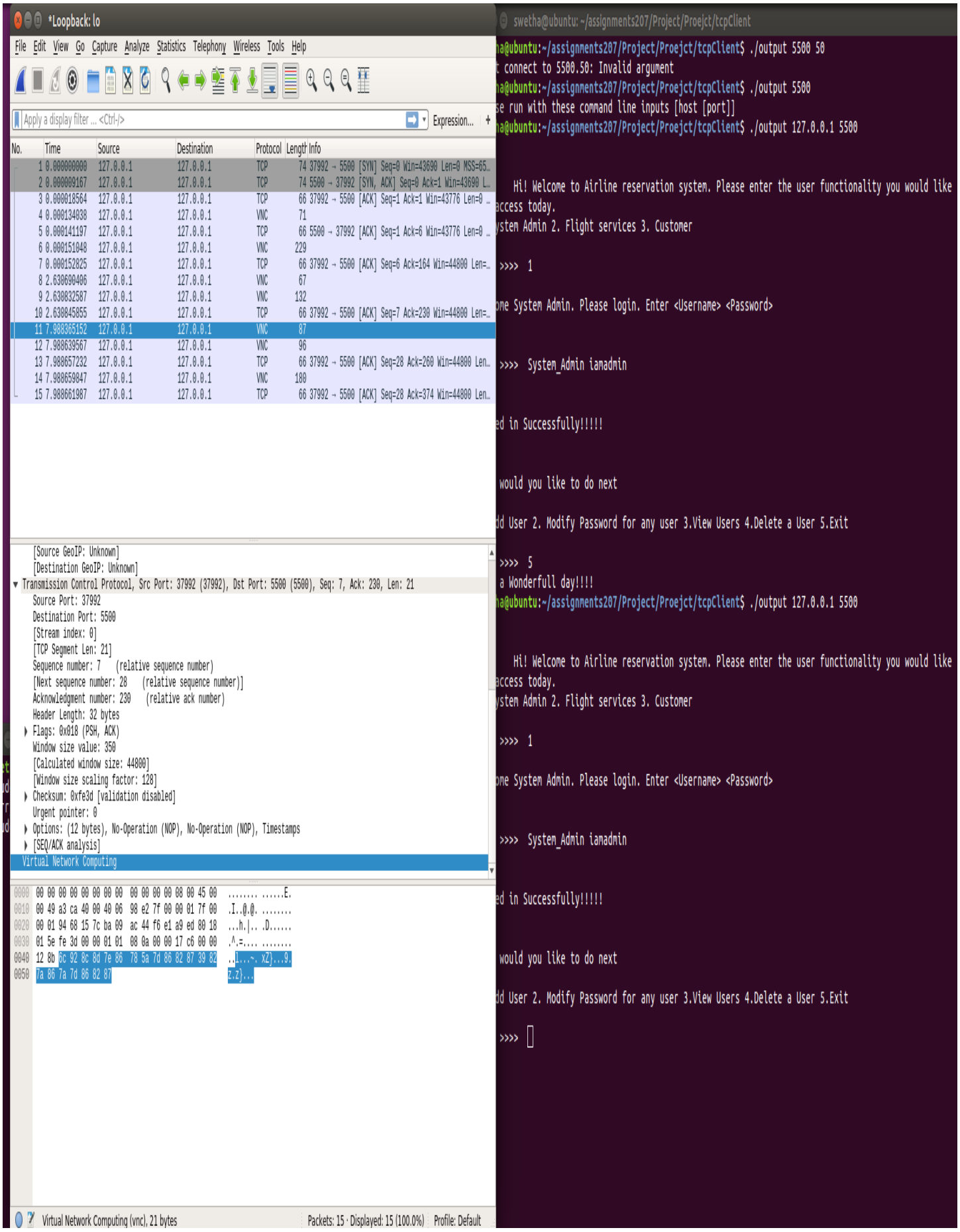
For data transfer across the network, we found it essential to include an encryption algorithm that protects messages between users. Seeing, as MD5 computations cannot be decrypted, an implementation of a 25-bit Encryption/Decryption algorithm suffice.

```
char * Encrypt(char *string)
{
    char *linee =  (char *)malloc(sizeof(char) * 2701);

    int n=strlen(string);
    int i=0;
    for(i=0;i<n;i++){
        string[i] = string[i]+25;
    }
    strcpy(linee, string);
    return linee;

}

char * Decrypt (char *string)
{
    char *linee =  (char *)malloc(sizeof(char) * 50);
    int n=strlen(string);
    int i=0;
    for(i=0;i<n;i++){
        string[i] = string[i]-25;
    }
    strcpy(linee, string);
    return linee;
}
```

Above is are the functions for encryption/Decryption of messages between the client and server.
The basic concept of the algorithm is to use a shifting mechanism to distort a string. Client sends encrypted data across the network after passing it through the Encrypt() function, and then Decrypt() messages arriving from the server, and vice-versa for the server, this way data travels securely. On arrival to the server, if a password needs to be authenticated or stored then MD5 is implemented.

```
  strcpy(username,fromclient);
  /*Request for password*/
  printf("%s",wclientMsg);
  bzero(toclient,2701);
  strcpy(toclient,pswQueryMsgg);
  encrypt = Encryptt(toclient);
  n1=write(ssock,encrypt,strlen(encrypt));
  if (n1 <= 0){
      errexit("ERROR writing to socket\n");
  }
  printf("%s",rclientMsg);
  bzero(fromclient,sizeof(fromclient));
  n1 = read(ssock,fromclient,sizeof(fromclient));
  decrypt=Decryptt(fromclient);
  bzero(fromclient,sizeof(fromclient));
  strcpy(fromclient,decrypt);
pthread_mutex_unlock(&getUserLock);
hash_psw = hash_genn_md5(password, strlen(password));
pthread_mutex_lock(&userloginLock);
int a = login_flight_prof(username, hash_psw);
pthread_mutex_unlock(&userloginLock);
```

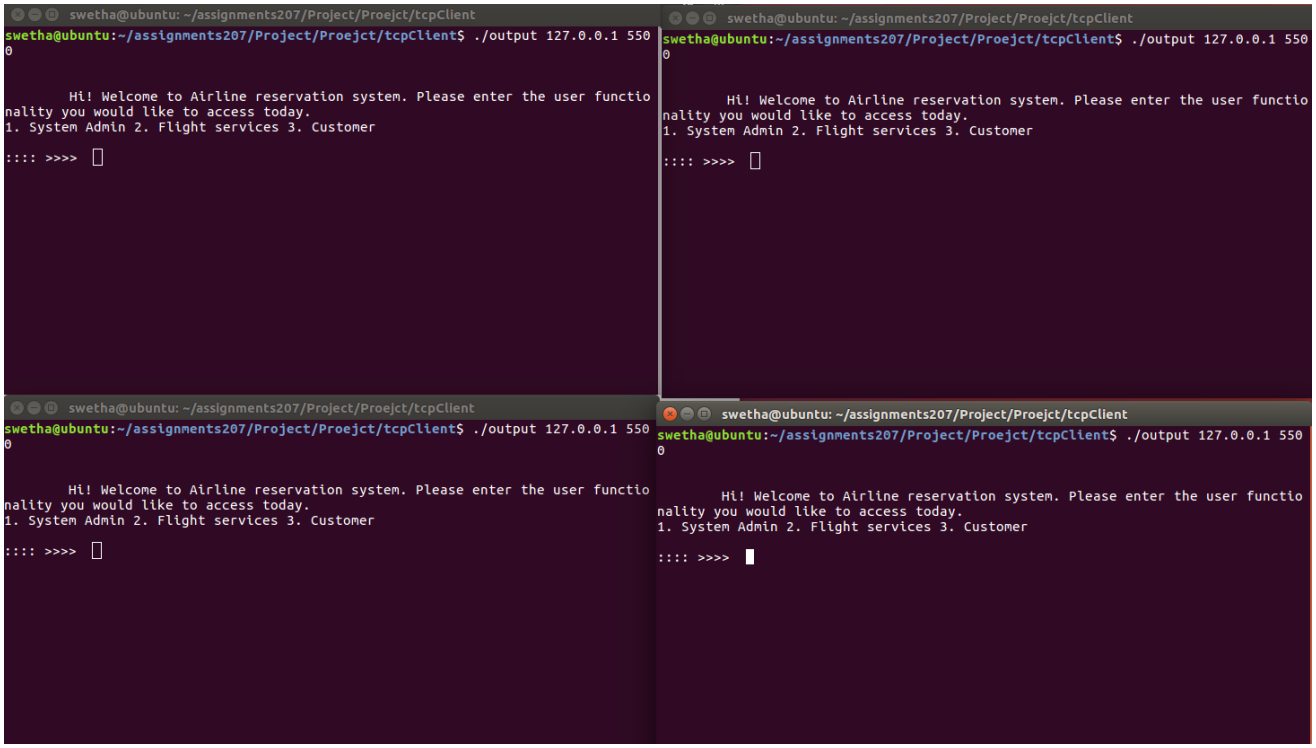
Above is the basic Idea, messages from client are decrypted using a 25-bit encryption, and the Hash value of the password is then computed using MD5.



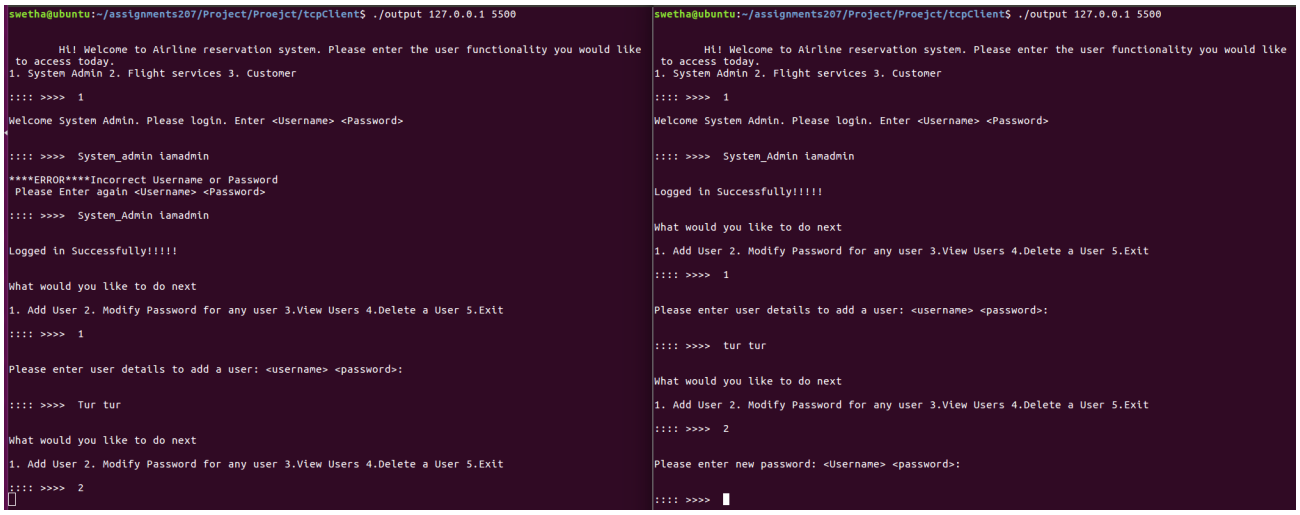
WireShark Analysis of Encryption

SERVER AND CLIENT IMPLEMENTATION

Multithreading in client



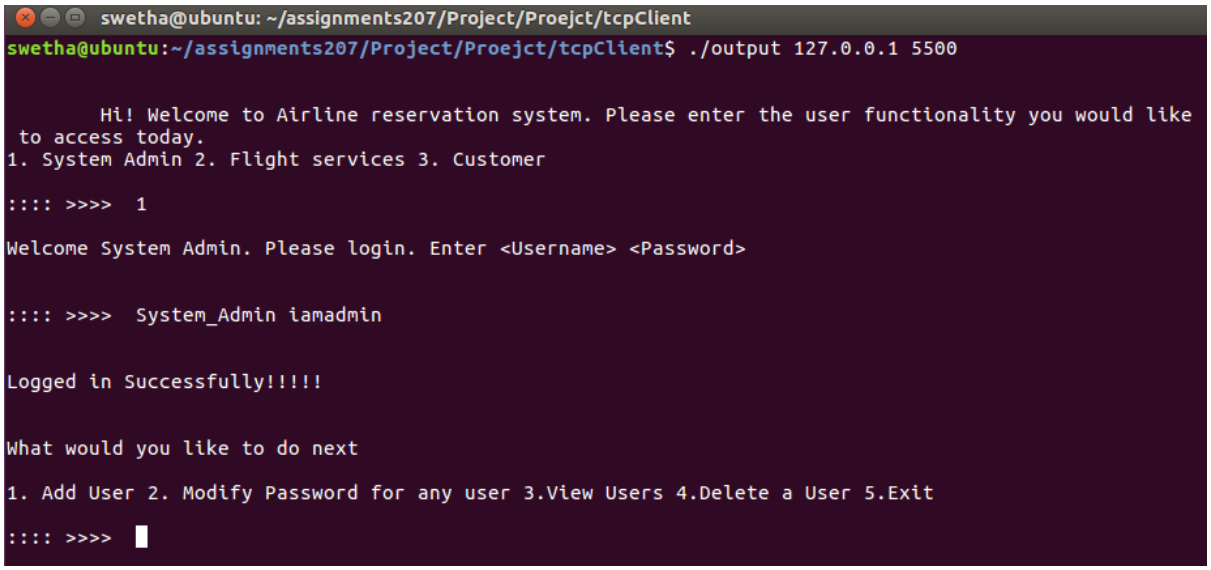
Critical section access in client



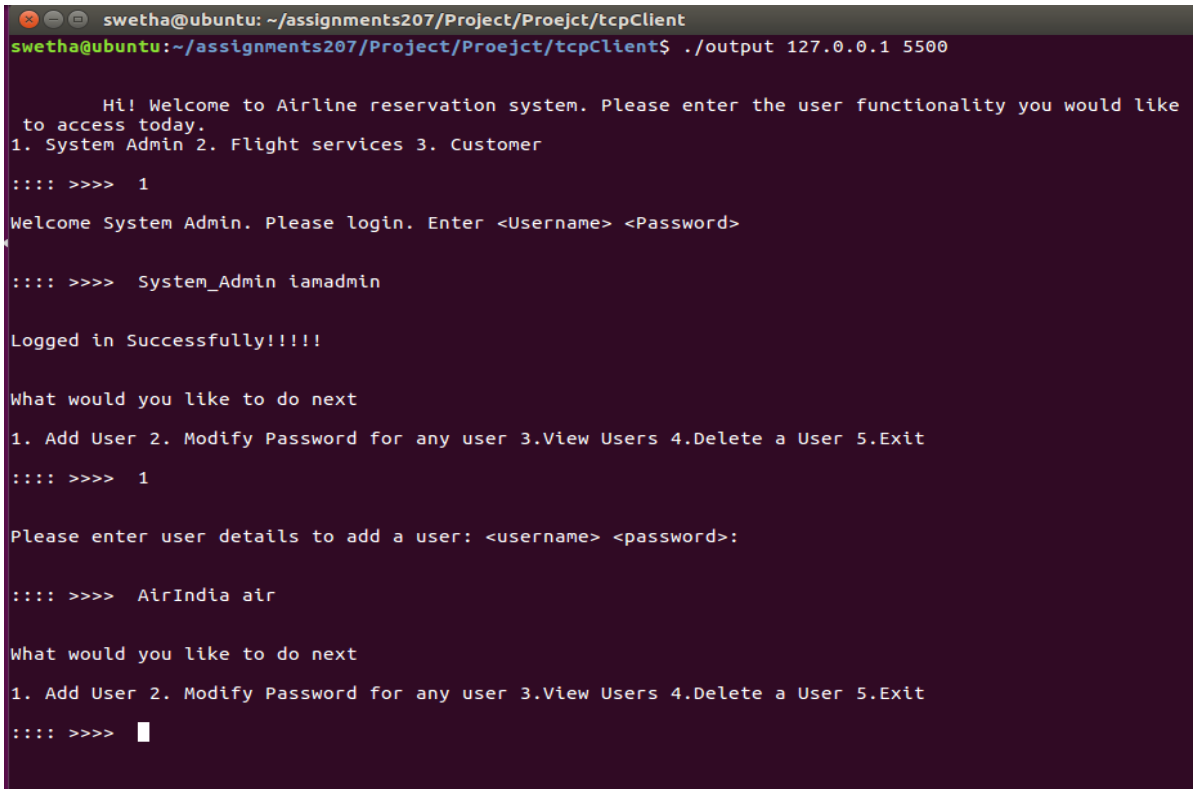
User1 is not able to access Modify password for a user module since user2 is in that section.

SYSTEM ADMINISTRATOR MODULE IMPLEMENTATION

1. Login:



2. Add a user



3. Modify a user

```
What would you like to do next

1. Add User 2. Modify Password for any user 3.View Users 4.Delete a User 5.Exit

:::: >>>>  2

Please enter new password: <Username> <password>:

:::: >>>>  AirIndia india

What would you like to do next

1. Add User 2. Modify Password for any user 3.View Users 4.Delete a User 5.Exit

:::: >>>>
```

4. View a user

```
What would you like to do next

1. Add User 2. Modify Password for any user 3.View Users 4.Delete a User 5.Exit

:::: >>>>  3


Users of this system are:
AirIndia
System_Admin


What would you like to do next

1. Add User 2. Modify Password for any user 3.View Users 4.Delete a User 5.Exit

:::: >>>>
```

5. Exit

```
What would you like to do next

1. Add User 2. Modify Password for any user 3.View Users 4.Delete a User 5.Exit

:::: >>>>  5
Have a Wonderfull day!!!!
swetha@ubuntu:~/assignments207/Project/Proejct/tcpClient$
```

TEST CASE EXECUTION

Test Case #	Description	Expected result	Actual result	Status
1	User should be able to login as a system admin	User should be able to login to the system as a system admin	User was able to login to the system as a system admin	Pass
2	User should not be able to login with wrong username	An error message should be displayed and the login screen should be displayed again	An error message was displayed and the login screen should be displayed again	Pass
3	User should not be able to login with a wrong password	An error message should be displayed and the login screen should be displayed again	An error message was displayed and the login screen should be displayed again	Pass
4	User should be able to view 5 options: add, modify, view, delete user and to exit the system.	Screen with the below options should be displayed: 1. Add a user 2. Modify user details 3. View a user 4. Delete a user 5. Exit	Screen with the below options was displayed: 1. Add a user 2. Modify user details 3. View a user 4. Delete a user 5. Exit	Pass
5	User should be able to Add a user.	User should obtain a screen with the following prompt: Please add user in the below format <username> <password>	User obtained a screen with the following prompt: Please add user in the below format <username> <password>	Pass
		The user details should be added to the database	The user details were added to the database	Pass
		All the users should be listed in the screen including the one recently added	All the users were listed in the screen including the one recently added	Pass
6	User added by system admin should be able to login to the system	The user should be able to login to the system	The user was able to login to the system	Pass
7	User should be able to modify the user password	User should obtain a screen with the following options: Please enter in the below format: <username> <password>	User obtained a screen with the following options: Please enter in the below format: <username> <password>	Pass
		The database should be updated with the details	The database was updated with the details	Pass
8	User should be able to delete a user	“Please enter the <username> should be displayed.	“Please enter the <username> was displayed.	Pass
		Modified successfully should be displayed	Modified successfully was displayed	Pass

9	User should be able to view the users in the system	All the users in the system should be displayed	All the users in the system was displayed	Pass
10	User should be able to exit the system	"Have a nice day!!!" should be displayed	"Have a nice day!!!" was displayed	Pass

AIRLINE MANAGER IMPLEMENTATION

- ✓ Resource Management/Thread Synchronization
  - Sharing of Global Resources within Multiple Threads.
    - Avoid data corruption/distortion when modifying or accessing flight information.
    - Mutual Exclusive access to shared items (Pthread Mutex).
- ✓ Data Security/Function Restrictions
  - Use of Airline ID to restrict users to sub-groups of data
  - Airlines are protected from unauthorized data modification by other airlines
  - Only specific users are allowed specific tasks
    - Deletion of an airline ability granted to only a system admin.
    - Addition, deletion and modification of flights allowed by only users of the airline associated with the flight.
- ✓ User Authentication/Password Integrity
  - Hash function Implementation
    - MD5 Hashing Algorithm
  - Record Message Digest of Login Credentials to database
  - Guaranteed user authentication before access to account information.
  - Guaranteed confidentiality of login details. Plaintext of login credentials are never recorded to the database.
  - Airline data is protected. Only users authorized by the system administrator have access to airline data.
- ✓ Multithreaded Database Support
  - Simultaneous access and manipulation of database by multiple threads.
  - Concurrent and periodic update of database with every transaction.

Resource Management/Thread Synchronization

Since all threads within a process share a single set of global resources, there is a crucial need to ensure thread synchronization. More specifically, we use the pthread Mutex API functions to synchronize and coordinate sensitive data items that need protection.

An instance of mutex utilization:

In the airline manager module, whenever there is a need to retrieve a single field of data from a table in our database we call the function ***get_field(const char *field1, const char *table, const char *field2, const char *known).*** This function copies the data from the database and stores it in a global callback buffer. A sensitive data item of that sort must be protected to avoid unwarranted access of the information. For example, in a case where the UID of a user must be retrieved in order to properly grant or deny user permission from performing certain actions like deletion of a flight. We must be careful to ensure that the user is granted the right privileges to avoid unauthorized data modification.

Therefore, to address the proposed case a simple algorithm is implemented:
First the mutex for the item is dynamically initialized.

```
pthread_mutex_init(&getfieldLock, NULL);
```

Once initialized, the thread calls pthread_mutex_lock before making a call to get_field, and calls pthread_mutex_unlock after using the item.

```
/*Get uid from flight_users account to restrict user's
  ability to modify flights of other airlines*/
pthread_mutex_lock(&getfieldLock);
get_field("UID", "Flight_Users", "name",username);
strcpy(uidrestrict, call_buff2);
pthread_mutex_unlock(&getfieldLock);
if(uidrestrict[strlen(uidrestrict) - 1] == '\n')
    uidrestrict[strlen(uidrestrict) - 1] = '\0';
```

This procedure ensures that only one thread accesses the data item at a time. The first thread that calls pthread_mutex_lock() is given access, all succeeding threads that call the function are blocked by the system until the mutex has been released. This way not more than one thread accesses “call_buff2” at any instance.

Case: Two Threads Registering the Same Username Simultaneously

Authentication of Flight Professionals and Data Security:

Before a user is given access to 'flight professional' privileges, they must first verify that they have been authorized by the system administrator. This is achieved by a login procedure described below.
User credentials (username, password) are retrieved from clients. The 32 hex-byte hash value (hex derivative of the message digest) of the client's password (string) is computed, this value is then compared with the computed value recorded for the given username during registration (assuming the username exists in the database).

```
login:   getUserInfo((void*)ssock);
        strcpy(username,globalusr);
        strcpy(password,globalpassw);
        pthread_mutex_unlock(&getUserLock);
        hash_psw = hash_genn_md5(password, strlen(password));
        pthread_mutex_lock(&userloginLock);
        int a = login_flight_prof(username, hash_psw);
        pthread_mutex_unlock(&userloginLock);
        if (a == 2) {
        //user does not exist
```

If both hash values match the user is then given access to the database, but access is given with some restrictions in order to protect all users of the system.

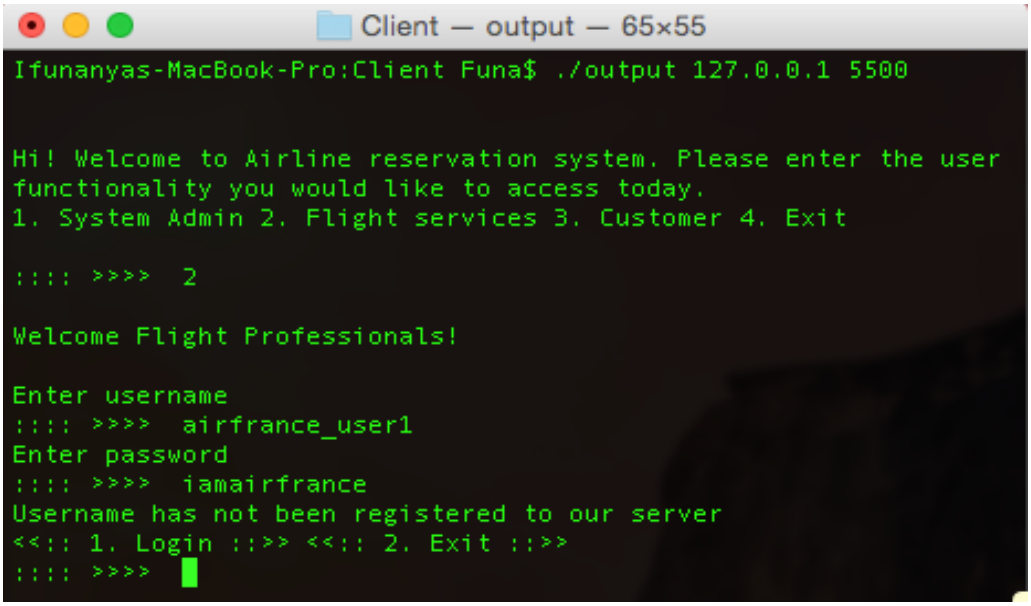
Shown below are users of the airline system, and all of the necessary information recorded for a user.

```
sqlite> select * from Flight_Users;
name                password                          airline      UID
------------------  --------------------------------  -----------  ----
System_Admin        a41acc7effe601de1dc2099a4e2fdd7c  N/A          0
klm_user1           2b774a335ea09a3c6bb7bd38216926cf  KLM          100
delta_user1         025f20f41a336fae7b66f2fe95b27770  Delta        200
emirates_user1      d09fe287bac7a3bff254549ff90d887d  Emirates     300
united_user1        aeac9dbe2af798d5c6390bd86d782bfa  United       400
arik_user1          e85218012f314a3beaffffb6eb130eff  Arik         500
Lufthanza_user1     b40d9399785c77c41cd5784319291a69  Lufthanza    600
jetair_user1        6296293fc17fb34c22176928594a712c  JetAirways   800
britishair_user1    bda51853df0e170885d321d0760aab29  BritishAirw  700
klm_user2           2b774a335ea09a3c6bb7bd38216926cf  KLM          100
delta_user2         025f20f41a336fae7b66f2fe95b27770  Delta        200
```

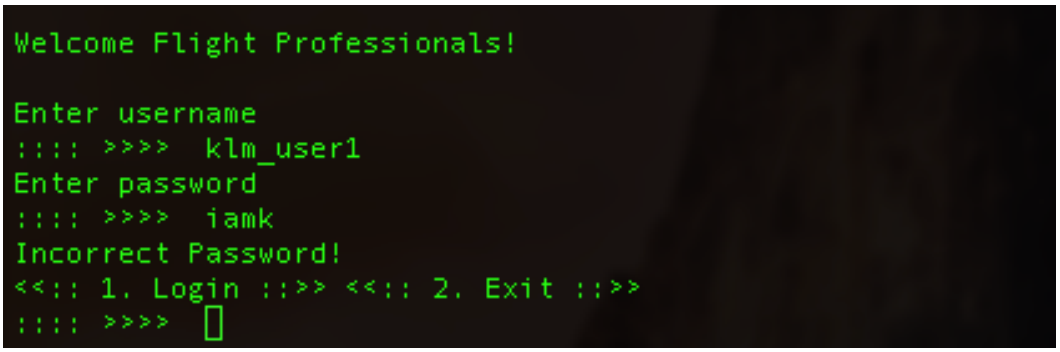
Database: Airline Professionals account information

We note that each user is assigned a UID equivalent to the ID of the user's airline. This UID will serve as a tool to identify users by their airlines and restrict users from carrying out specific actions. After users have logged in successfully, their UID is then extracted from database, and used periodically while in the airline manager procedure, to deny or grant users access to a requested functionality.

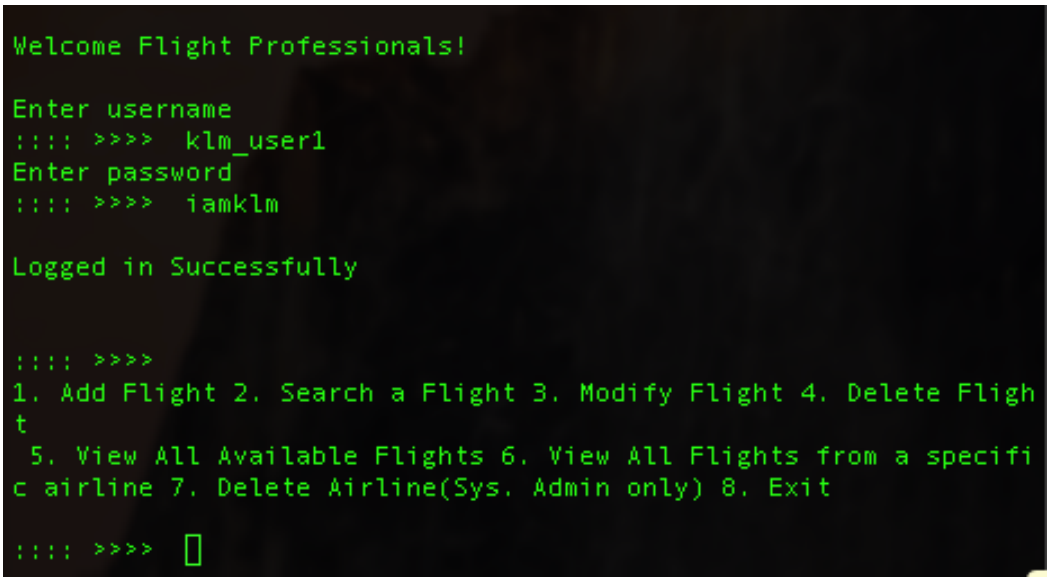
Login:



CASE: Username not registered



CASE: Incorrect Password



CASE: Login Successful

After users have been successfully logged in. The UID assigned to their airline is then copied and used as a restrictive key.

```
/*Get uid from flight_users account to restrict user's
 ability to modify flights of other airlines*/
pthread_mutex_lock(&getfieldLock);
get_field("UID", "Flight_Users", "name",username);
strcpy uidrestrict, call_buff2);
pthread_mutex_unlock(&getfieldLock);
if(uidrestrict[strlen(uidrestrict) - 1] == '\n')
    uidrestrict[strlen(uidrestrict) - 1] = '\0';
```

If users would like to add, modify or delete a flight. The “uidrestrict” is used to extract the user’s associated airline from the UID_STORE, this is then used to admit or restrict users to a given functionality.

```
sqlite> select * from UID_STORE;
UID                 AIRLINE
------------------  ----------------------------
100                 KLM
200                 Delta
300                 Emirates
400                 United
500                 Arik
600                 Lufthanza
800                 JetAirways
700                 BritishAirways
sqlite>
```

TABLE: UID_STORE

```
    printf("Request from client to add flight\n");
    /*Get airline name from UID_STORE account to restrict
     user's ability to modify flights of other airlines*/
    pthread_mutex_lock(&getfieldLock);
    get_field("airline", "UID_STORE", "UID", uidrestrict);
    strcpy(airlineRestrict, call_buff2);
    pthread_mutex_unlock(&getfieldLock);
    if(airlineRestrict[strlen(airlineRestrict) - 1] == '\n')
        airlineRestrict[strlen(airlineRestrict) - 1] = '\0';
add:  /*Request for flight name*/
```

Above is an example whereby a modification request from a client such as a request to “add a flight”, would first require that the airline of the flight the client intends to add matches the airline the client belongs to. This verification routine is limited to “add flight”, “delete flight” and “modify flight”. All users are allowed to view flight information of other airlines but are not permitted to utilize modification functions on other airlines.

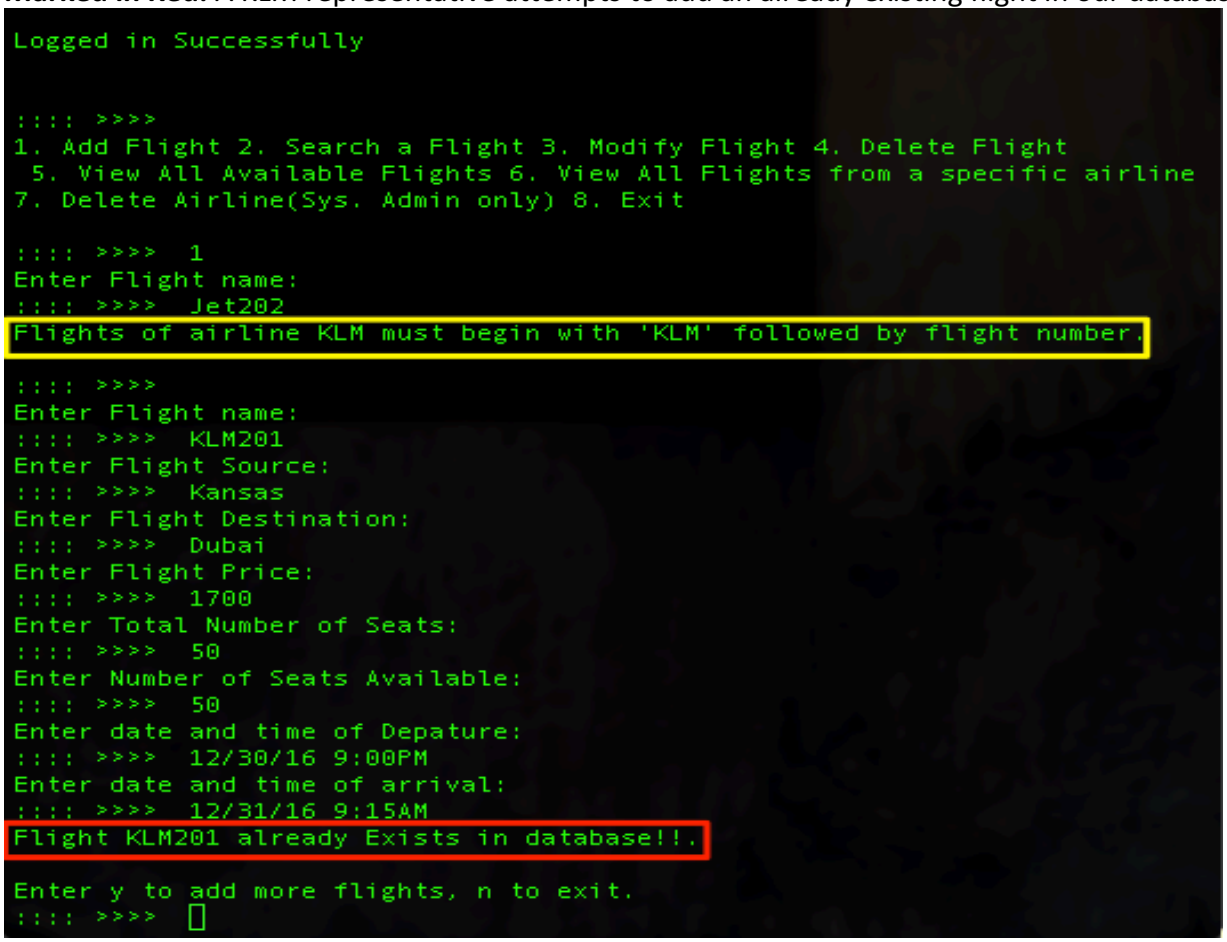
Add A Flight:

Before a flight can be added to the database the server must first verify that the airline ID matches the airline enquired by the user. After the airline name that the client is restricted to has been deduced, it is used to evaluate the name of the flight that the client would like to add. If it does not meet the naming specification, client is prompted.

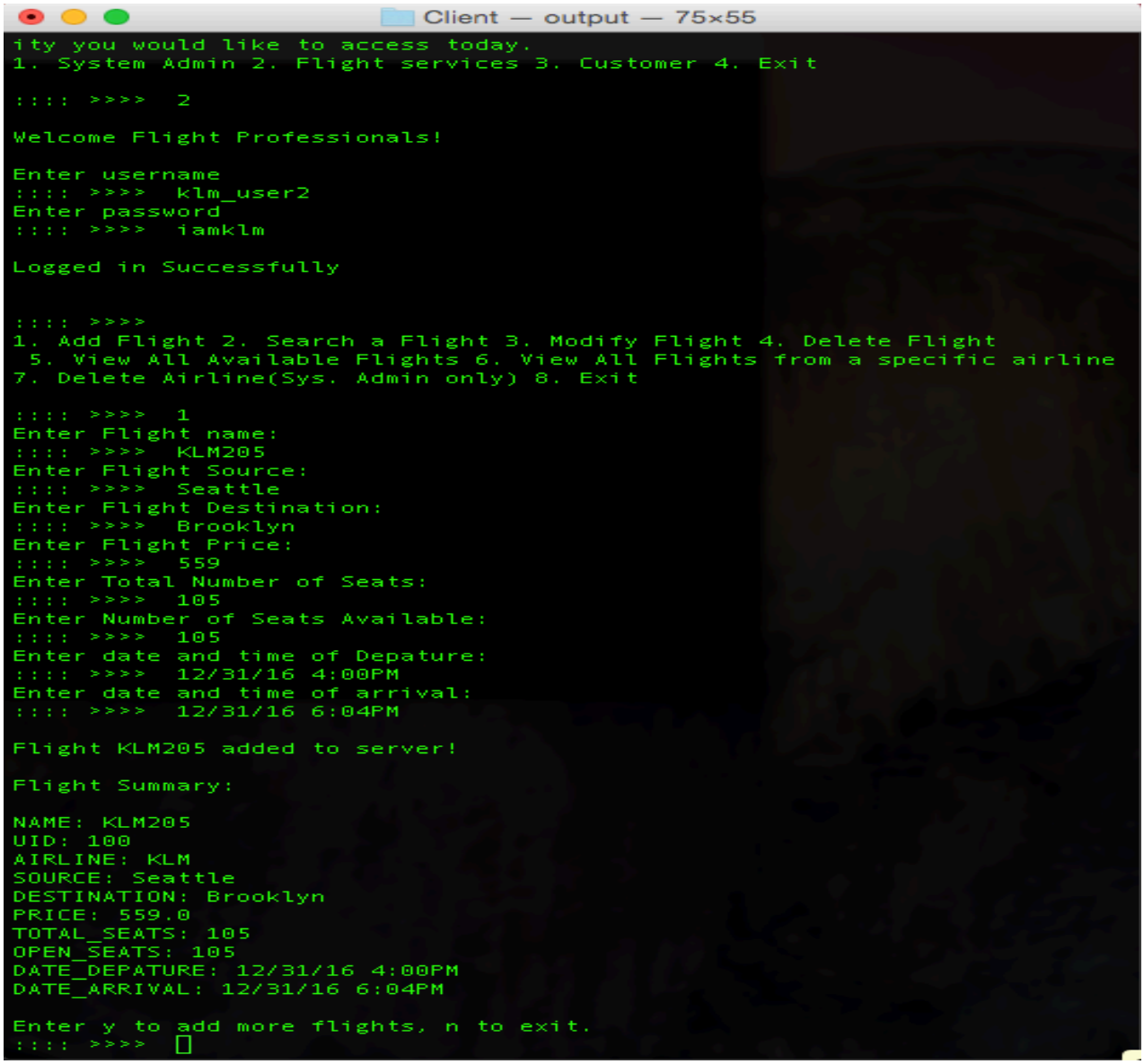
```
strcpy(name,fromclient);
if ((strncmp(name, airlineRestrict,3)) != 0){
    //incorrect name type for airline
    char name_tag[3];
    strncpy(name_tag, airlineRestrict,3);
    printf("%s",wclientMsg);
    bzero(toclient,sizeof(toclient));
    sprintf(toclient,"Flights of airline %s must begin with "\
    "'%s' followed by flight number.\n",airlineRestrict,name_tag);
    encrypt = Encryptt(toclient);
    n1=write(ssock,encrypt,strlen(encrypt));
    if (n1 <= 0){
        errexit("ERROR writing to socket\n");
    }
    goto add;
}
```

Code Snippet: Restricting an unauthorized user from adding a flight of an airline different from theirs.

Marked in yellow, a client attempts to add a flight for Jet Airways. Client is a KLM representative and is therefore restricted from completing the tasks.
Marked in Red: A KLM representative attempts to add an already existing flight in our database.



CASE: ADD Flight Failure



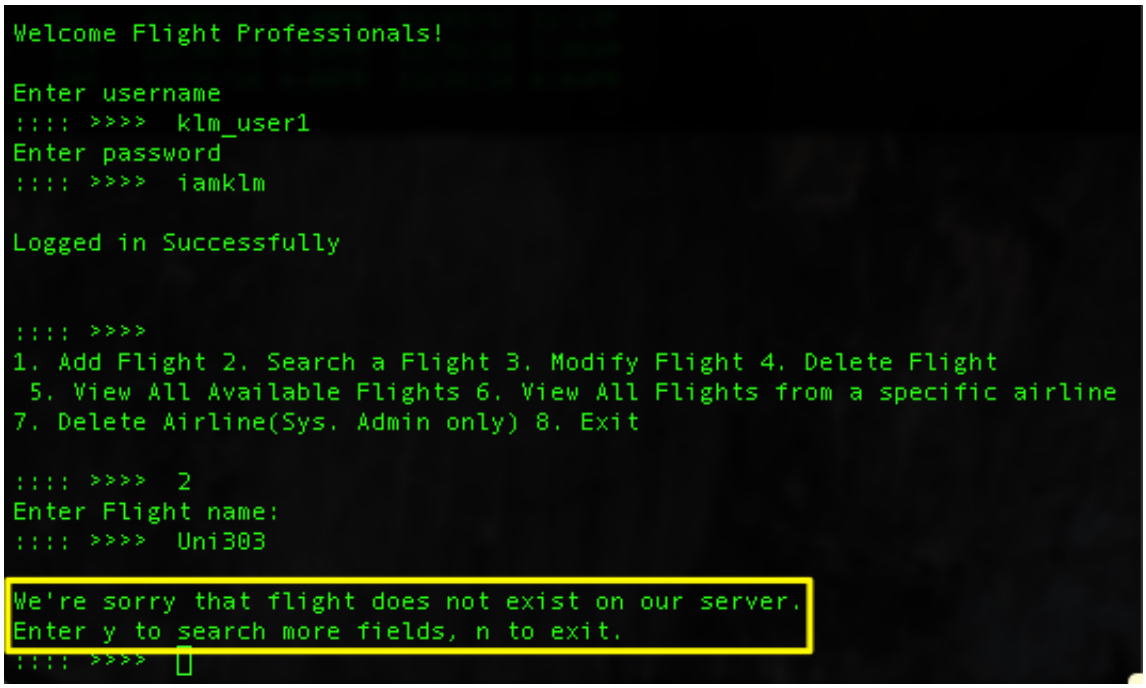
CASE: Successful addition of a flight.

Marked in Red: The new flight klm205 recorded in database.

```
sqlite> select * from DATA;
NAME     UID   AIRLINE          SOURCE           DESTINATION      PRICE    TOTA  OPEN  DATE_DEPATURE    DATE_ARRIVAL
-------  ----  ---------------  ---------------  ---------------  -------  ----  ----  ---------------  ---------------
KLM201   100   KLM              San Jose         Morocco          2225.0   95    95    12/02/17 9:15PM  12/03/17 5:05AM
KLM203   100   KLM              London           Chicago          1350.0   79    79    03/02/17 6:05PM  03/03/17 1:15AM
KLM204   100   KLM              Amsterdam        Lagos            1950.0   105   105   01/01/17 5:05AM  01/02/17 3:15AM
Del201   200   Delta            Tokyo            Accra            2315.5   94    94    03/05/17 9:00PM  03/06/17 4:17PM
Del202   200   Delta            Poland           Lisbon           2216.99  99    99    02/19/17 4:19PM  02/19/17 11:46P
Del203   200   Delta            San Diego        Boston           559.95   85    85    12/22/16 7:10AM  12/22/16 9:15AM
Emi201   300   Emirates         Dubai            Pretoria         2050.0   100   100   12/23/16 7:15PM  12/24/16 1:04PM
Emi202   300   Emirates         Los Angeles      Dubai            2305.0   115   115   01/05/17 9:00PM  01/06/17 3:25PM
Uni102   400   United           New York         Singapore        1205.0   79    79    01/13/17 5:15PM  01/04/17 2:34PM
Uni103   400   United           Dallas           Sacramento       345.0    105   105   12/19/17 9:37PM  12/19/17 11:15P
Jet302   800   JetAirways       China            Norway           2015.99  75    75    04/19/17 8:00PM  04/20/17 12:45P
Luf111   600   Lufthanza        London           Chicago          1950.0   117   117   07/25/17 3:09PM  07/26/17 5:15AM
Ari202   500   Arik             Enugu            Abuja            345.95   60    60    03/03/17 9:00PM  03/03/17 11:15P
Bri420   700   BritishAirways   Oxford           San Francisco    1699.0   125   125   12/30/16 5:45PM  12/31/16 7:06AM
KLM205   100   KLM              Seattle          Brooklyn         559.0    105   105   12/31/16 4:00PM  12/31/16 6:04PM
sqlite>
```

View A Flight:

Because viewing a flight isn't a modification service, all users are permitted to view any flight.



CASE: Flight not recorded in server

Notice below that a KLM flight professional is able to view flights of a different airline. However, as we will see, users cannot modify flight information of a different airline.

CASE: Flight exists in server

Modify Flights:

The modify flight functionality, puts a restriction on users, ensuring that only users of a specific airline have the ability to modify a flight. Airlines associated with the users are compared with the airline of the flight being accessed before users are able to modify a flight.

```
/*Check to make sure user is authorized to modify flight*/
pthread_mutex_lock(&getfieldLock);
get_field("airline","data","name",name);
strcpy(airline_, call_buff2);
pthread_mutex_unlock(&getfieldLock);
if(airline_[strlen(airline_) - 1] == '\n')
    airline_[strlen(airline_) - 1] = '\0';
if ((strcmp(airlineRestrict, airline_)) != 0){
//not authorized to modify flight
    printf("%s",wclientMsg);
```

Below a KLM Flight professional attempts to modify a flight of British Airways.

```
1. Add Flight 2. Search a Flight 3. Modify Flight 4. Delete Flight
 5. View All Available Flights 6. View All Flights from a specific airline
7. Delete Airline(Sys. Admin only) 8. Exit

:::: >>>>  3
Enter Flight name:
:::: >>>>  Bri420

You are not authorized to modify this flight.
<:: 1. Go Back ::>> <<:: 2. Exit 'Modify Flight' ::>>

:::: >>>>
```

CASE: Unauthorized user

```
You are not authorized to modify this flight.
<:: 1. Go Back ::>> <<:: 2. Exit 'Modify Flight' ::>>

:::: >>>>  1
Enter Flight name:
:::: >>>>  klm201

We're sorry that flight does not exist on our server.

:::: >>>>
```

CASE: Flight does not exist

```
:::: >>>>
Enter Flight name:
:::: >>>>  KLM203

Request to modify flight KLM203 with airline ID 100.

Enter to modify:
<<:: 1. All ::>> <<:: 2. Source ::>> <<:: 3.Destination ::>> <<:: 4. Price ::>>
<<:: 5. No of Seats Available ::>><<:: 6. Date of Departure ::>> <<:: 7. Date of Arrival ::>>

:::: >>>>  4
Enter Flight Price:
:::: >>>>  1295
Updated Flight Details for KLM203:

NAME: KLM203
UID: 100
AIRLINE: KLM
SOURCE: London
DESTINATION: Chicago
PRICE: 1295.0
TOTAL_SEATS: 79
OPEN_SEATS: 79
DATE_DEPATURE: 03/02/17 6:05PM
DATE_ARRIVAL: 03/03/17 1:15AM


Enter y to modify more fields, n to exit
:::: >>>>
```

CASE: Modification Successful

```
KLM203   100   KLM          London          Chicago          1350.0   79   79   03/02/17 6:05PM   03/03/17 1:15AM
```

Previous Flight Price in Database

Delete Flights:

Just like in the case of flight modification and addition. Users are restricted to deleting only flights of their respective airlines. Below we find a KLM representative attempt to Delete a Lufthansa Airline Flight.

```
1. Add Flight 2. Search a Flight 3. Modify Flight 4. Delete Flight
 5. View All Available Flights 6. View All Flights from a specific airline
7. Delete Airline(Sys. Admin only) 8. Exit

:::: >>>>  4
Enter Flight name:
:::: >>>>  Luf111

You are not authorized to delete this flight.
<:: 1. Go Back ::>> <<:: 2. Exit 'Delete Flight' ::>>

:::: >>>>
```

CASE: Not Authorized user

```
Enter Flight name:
:::: >>>>  KLM445

We're sorry that flight does not exist on our server.
Enter Flight name:
:::: >>>>
```

CASE: Flight Does Not Exist

Below we find that the KLM representative is able to delete a KLM flight.

```
Enter Flight name:
:::: >>>>  KLM203
Flight KLM203 deleted successfully!
Enter y to delete more flights, n to exit.
:::: >>>>
```

CASE: Flight Successfully Deleted

```
sqlite> select * from DATA;
NAME     UID   AIRLINE          SOURCE           DESTINATION      PRICE    TOTA  OPEN  DATE_DEPATURE    DATE_ARRIVAL
-------  ----  ---------------  ---------------  ---------------  -------  ----  ----  ---------------  ---------------
KLM201   100   KLM              San Jose         Morocco          2225.0   95    95    12/02/17 9:15PM  12/03/17 5:05AM
KLM203   100   KLM              London           Chicago          1350.0   79    79    03/02/17 6:05PM  03/03/17 1:15AM
KLM204   100   KLM              Amsterdam        Lagos            1950.0   105   105   01/01/17 5:05AM  01/02/17 3:15AM
Del201   200   Delta            Tokyo            Accra            2315.5   94    94    03/05/17 9:00PM  03/06/17 4:17PM
Del202   200   Delta            Poland           Lisbon           2216.99  99    99    02/19/17 4:19PM  02/19/17 11:46P
Del203   200   Delta            San Diego        Boston           559.95   85    85    12/22/16 7:10AM  12/22/16 9:15AM
Emi201   300   Emirates         Dubai            Pretoria         2050.0   100   100   12/23/16 7:15PM  12/24/16 1:04PM
Emi202   300   Emirates         Los Angeles      Dubai            2305.0   115   115   01/05/17 9:00PM  01/06/17 3:25PM
Uni102   400   United           New York         Singapore        1205.0   79    79    01/13/17 5:15PM  01/04/17 2:34PM
Uni103   400   United           Dallas           Sacramento       345.0    105   105   12/19/17 9:37PM  12/19/17 11:15P
Jet302   800   JetAirways       China            Norway           2015.99  75    75    04/19/17 8:00PM  04/20/17 12:45P
Luf111   600   Lufthanza        London           Chicago          1950.0   117   117   07/25/17 3:09PM  07/26/17 5:15AM
Ari202   500   Arik             Enugu            Abuja            345.95   60    60    03/03/17 9:00PM  03/03/17 11:15P
Bri420   700   BritishAirways   Oxford           San Francisco    1699.0   125   125   12/30/16 5:45PM  12/31/16 7:06AM
KLM205   100   KLM              Seattle          Brooklyn         559.0    105   105   12/31/16 4:00PM  12/31/16 6:04PM
sqlite>
```

DATABASE: Before KLM203 Deletion

Below we find that KLM203 no longer exists in the database.

```
sqlite> select * from DATA;
NAME     UID   AIRLINE          SOURCE           DESTINATION      PRICE    TOTA  OPEN  DATE_DEPATURE    DATE_ARRIVAL
-------  ----  ---------------  ---------------  ---------------  -------  ----  ----  ---------------  ---------------
KLM201   100   KLM              San Jose         Morocco          2225.0   95    95    12/02/17 9:15PM  12/03/17 5:05AM
KLM204   100   KLM              Amsterdam        Lagos            1950.0   105   105   01/01/17 5:05AM  01/02/17 3:15AM
Del201   200   Delta            Tokyo            Accra            2315.5   94    94    03/05/17 9:00PM  03/06/17 4:17PM
Del202   200   Delta            Poland           Lisbon           2216.99  99    99    02/19/17 4:19PM  02/19/17 11:46P
Del203   200   Delta            San Diego        Boston           559.95   85    85    12/22/16 7:10AM  12/22/16 9:15AM
Emi201   300   Emirates         Dubai            Pretoria         2050.0   100   100   12/23/16 7:15PM  12/24/16 1:04PM
Emi202   300   Emirates         Los Angeles      Dubai            2305.0   115   115   01/05/17 9:00PM  01/06/17 3:25PM
Uni102   400   United           New York         Singapore        1205.0   79    79    01/13/17 5:15PM  01/04/17 2:34PM
Uni103   400   United           Dallas           Sacramento       345.0    105   105   12/19/17 9:37PM  12/19/17 11:15P
Jet302   800   JetAirways       China            Norway           2015.99  75    75    04/19/17 8:00PM  04/20/17 12:45P
Luf111   600   Lufthanza        London           Chicago          1950.0   117   117   07/25/17 3:09PM  07/26/17 5:15AM
Ari202   500   Arik             Enugu            Abuja            345.95   60    60    03/03/17 9:00PM  03/03/17 11:15P
Bri420   700   BritishAirways   Oxford           San Francisco    1699.0   125   125   12/30/16 5:45PM  12/31/16 7:06AM
KLM205   100   KLM              Seattle          Brooklyn         559.0    105   105   12/31/16 4:00PM  12/31/16 6:04PM
sqlite>
```

DATABASE: After Deletion

View All Flights in the Server:

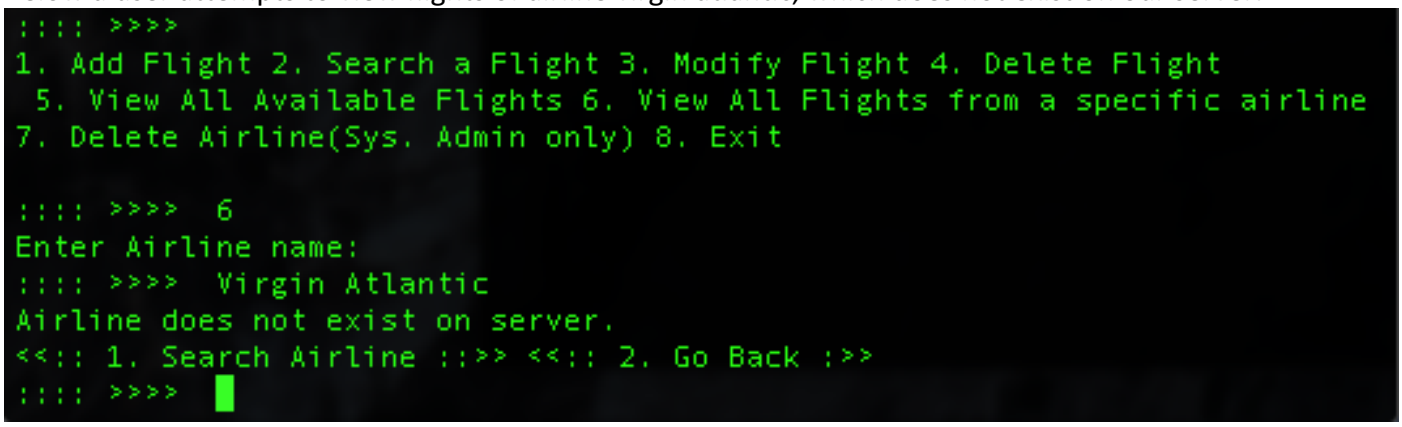
Because viewing flights isn't a modification service, all users are permitted to view all the flights in the database.

CASE: View All Flights

View Flights of A Specific Airline:

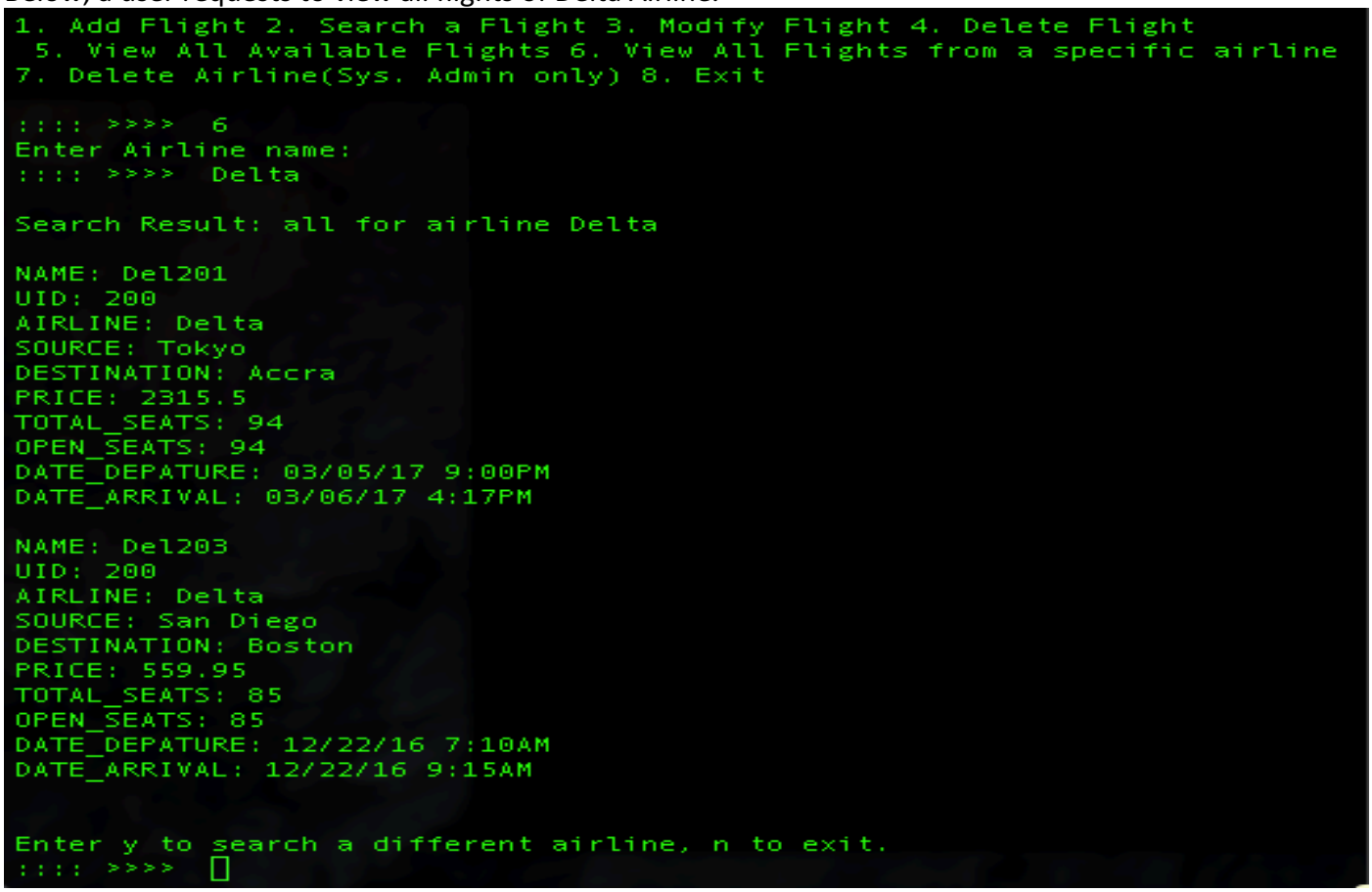
There are also no restrictions on the ability to view flights of a specific airline, as long as users have been authenticated via login.

Below a user attempts to view flights of airline virgin atlantic, which does not exist on our server.



CASE: Airline does not exist in server

Below, a user requests to view all flights of Delta Airline.



CASE: View Flight of an Airline Success

Delete Airline:

The delete airline service is only available to the system administrator. The system admin alone can delete an airline from the database, which will also subsequently delete the airline's UID from the UID_STORE.

```
'7':
   printf("Request from client to delete an airline\n");
   /*If sys admin, delete airline else break*/
   if ((strcmp(username, "System_Admin")) == 0) {
       /*Request for airline ID*/
       printf("%s",wclientMsg);
```

Below a flight professional requests to use the "delete airline" functionality, but is denied the request.

```
1. Add Flight 2. Search a Flight 3. Modify Flight 4. Delete Flight
 5. View All Available Flights 6. View All Flights from a specific airline
7. Delete Airline(Sys. Admin only) 8. Exit

:::: >>>>  7

You are not authorized to utilize this function.

:::: >>>>
```

CASE: User Unauthorized (Not a System Administrator)

Below, the system administrator logs in and successfully deletes British Airways and all of its data from the database.

```
Welcome Flight Professionals!

Enter username
:::: >>>>  System_Admin
Enter password
:::: >>>>  iamadmin

Logged in Successfully


:::: >>>>
1. Add Flight 2. Search a Flight 3. Modify Flight 4. Delete Flight
 5. View All Available Flights 6. View All Flights from a specific airline
7. Delete Airline(Sys. Admin only) 8. Exit

:::: >>>>  7
Enter Airline ID:
:::: >>>>  700

Delete all flights from airline BritishAirways?
<<:: 1. Confirm Delete ::>> <<:: 2. Cancel ::>>
:::: >>>>  1

Airline successfully deleted!

:::: >>>>
```

CASE: Delete Airline Successful (Authorized sys. Admin)

Our server has just one flight from British Airways.

```
sqlite> select * from DATA;
NAME     UID   AIRLINE          SOURCE           DESTINATION      PRICE    TOTA  OPEN  DATE_DEPATURE     DATE_ARRIVAL
-------  ----  ---------------  ---------------  ---------------  -------  ----  ----  ----------------  ----------------
KLM201   100   KLM              San Jose         Morocco          2225.0   95    95    12/02/17 9:15PM   12/03/17 5:05AM
KLM204   100   KLM              Amsterdam        Lagos            1950.0   105   105   01/01/17 5:05AM   01/02/17 3:15AM
Del201   200   Delta            Tokyo            Accra            2315.5   94    94    03/05/17 9:00PM   03/06/17 4:17PM
Del202   200   Delta            Poland           Lisbon           2216.99  99    99    02/19/17 4:19PM   02/19/17 11:46P
Del203   200   Delta            San Diego        Boston           559.95   85    85    12/22/16 7:10AM   12/22/16 9:15AM
Emi201   300   Emirates         Dubai            Pretoria         2050.0   100   100   12/23/16 7:15PM   12/24/16 1:04PM
Emi202   300   Emirates         Los Angeles      Dubai            2305.0   115   115   01/05/17 9:00PM   01/06/17 3:25PM
Uni102   400   United           New York         Singapore        1205.0   79    79    01/13/17 5:15PM   01/04/17 2:34PM
Uni103   400   United           Dallas           Sacramento       345.0    105   105   12/19/17 9:37PM   12/19/17 11:15P
Jet302   800   JetAirways       China            Norway           2015.99  75    75    04/19/17 8:00PM   04/20/17 12:45P
Luf111   600   Lufthanza        London           Chicago          1950.0   117   117   07/25/17 3:09PM   07/26/17 5:15AM
Ari202   500   Arik             Enugu            Abuja            345.95   60    60    03/03/17 9:00PM   03/03/17 11:15P
Bri420   700   BritishAirways   Oxford           San Francisco    1699.0   125   125   12/30/16 5:45PM   12/31/16 7:06AM
KLM205   100   KLM              Seattle          Brooklyn         559.0    105   105   12/31/16 4:00PM   12/31/16 6:04PM
sqlite>
```

DATABASE: Before airline is deleted

Notice below, no flights from British Airways exist.

```
sqlite> select * from DATA;
NAME     UID   AIRLINE          SOURCE           DESTINATION      PRICE    TOTA  OPEN  DATE_DEPATURE     DATE_ARRIVAL
-------  ----  ---------------  ---------------  ---------------  -------  ----  ----  ----------------  ----------------
KLM201   100   KLM              San Jose         Morocco          2225.0   95    95    12/02/17 9:15PM   12/03/17 5:05AM
KLM204   100   KLM              Amsterdam        Lagos            1950.0   105   105   01/01/17 5:05AM   01/02/17 3:15AM
Del201   200   Delta            Tokyo            Accra            2315.5   94    94    03/05/17 9:00PM   03/06/17 4:17PM
Del203   200   Delta            San Diego        Boston           559.95   85    85    12/22/16 7:10AM   12/22/16 9:15AM
Emi201   300   Emirates         Dubai            Pretoria         2050.0   100   100   12/23/16 7:15PM   12/24/16 1:04PM
Emi202   300   Emirates         Los Angeles      Dubai            2305.0   115   115   01/05/17 9:00PM   01/06/17 3:25PM
Uni102   400   United           New York         Singapore        1205.0   79    79    01/13/17 5:15PM   01/04/17 2:34PM
Uni103   400   United           Dallas           Sacramento       345.0    105   105   12/19/17 9:37PM   12/19/17 11:15P
Jet302   800   JetAirways       China            Norway           2015.99  75    75    04/19/17 8:00PM   04/20/17 12:45P
Luf111   600   Lufthanza        London           Chicago          1950.0   117   117   07/25/17 3:09PM   07/26/17 5:15AM
Ari202   500   Arik             Enugu            Abuja            345.95   60    60    03/03/17 9:00PM   03/03/17 11:15P
KLM205   100   KLM              Seattle          Brooklyn         559.0    105   105   12/31/16 4:00PM   12/31/16 6:04PM
sqlite>
```

DATABASE: After British Airways is deleted

```
sqlite> select * from UID_STORE;
UID                 AIRLINE
------------------  --------------------------------
100                 KLM
200                 Delta
300                 Emirates
400                 United
500                 Arik
600                 Lufthanza
800                 JetAirways
700                 BritishAirways
```

DATABASE: UID_STORE BEFORE DELETION

Notice Below that BritishAirways and its associated UID has been deleted

```
sqlite> select * from UID_STORE;
UID    AIRLINE
----   --------------------
100    KLM
200    Delta
300    Emirates
400    United
500    Arik
600    Lufthanza
800    JetAirways
sqlite>
```

DATABASE: UID_STORE AFTER DELETION

CUSTOMER USER IMPLEMENTATION

- ✓ Resource Management/Thread Synchronization
  - Sharing of Global Resources within Multiple Threads.
    - Mutual Exclusive access to shared items (Pthread Mutex).
  - Get most up-to-date seat availability information at every query.
  - Avoid Clients booking the same seat at the same time.
  - Avoid more than one thread updating available seats value at the same time during booking and cancellation of booked seat.
  - Avoid Clients registering the same username.
- ✓ User Authentication/Password Confidentiality
  - Hash function Implementation
    - MD5 Hashing Algorithm
  - Record Message Digest of Login Credentials to database
  - Guaranteed user authentication before access of account information.
  - Guaranteed confidentiality of login details. Plaintext of login credentials are never recorded to the database.
  - Booked seat protection. Only appropriate customer can cancel associated booked seat.
- ✓ Multithreaded Database Support
  - Simultaneous access and manipulation of database by multiple threads.
  - Concurrent and periodic update of database with every transaction.

Thread Synchronization (Mutex lock/unlock)

Since all threads within a process share a single set of global resources, there is a crucial need to ensure thread synchronization. More specifically, we use the pthread Mutex API functions to synchronize and coordinate sensitive data items that need protection.

An instance of mutex utilization:

For the server to authenticate a user, it makes a procedure call to a function login_user() which returns a global variable “id_k” that validates or denies a user, it is set to “1” in the callback function if the password entered by the user matches the password stored in the database for that username. Basically, for each thread to get an accurate return value no other thread should have access to login_user() and subsequently callback2(). Therefore, this sensitive data item must be managed properly to ensure that users accessing the resource at the same time are serviced properly, i.e. prevent authenticating an invalid user who tries to login at the same time as an authentic user, or prevent denying an authentic user.

```
int callback2(void *password,int argc,char **argv,char **azColName){
    int i;
    id_k = 0;
    for(i=0; i<argc; i++){
        if(strcmp((const char*) password,argv[i])==0){
            id_k = 1;
        }
    }
    return 0;
}
int login_user(char* user,char* password)
{
    char *zErrMsg = 0;
    char *sql;
    char buff[256];
    int rc;

    bzero(buff, 256);
    /*Creates SQL statement.*/
    sprintf(buff, "SELECT password from customer_account"\
    "where username = '%s';",user);
    sql = buff;
    /*Execute SQL statement*/
    rc = sqlite3_exec(db3,sql,callback2,(void*)password,&zErrMsg);
    if (rc != SQLITE_OK) {
        fprintf(stderr, "SQL error: %s\n", zErrMsg);
        sqlite3_free(zErrMsg);
    }
    return id_k;
}
```

Therefore, to address this a simple algorithm is implemented:
First the mutex for the item is dynamically initialized.

```
pthread_mutex_init(&loginLock, NULL);
```

Once initialized, the thread calls pthread_mutex_lock before calling login_user, and calls pthread_mutex_unlock after using the item.

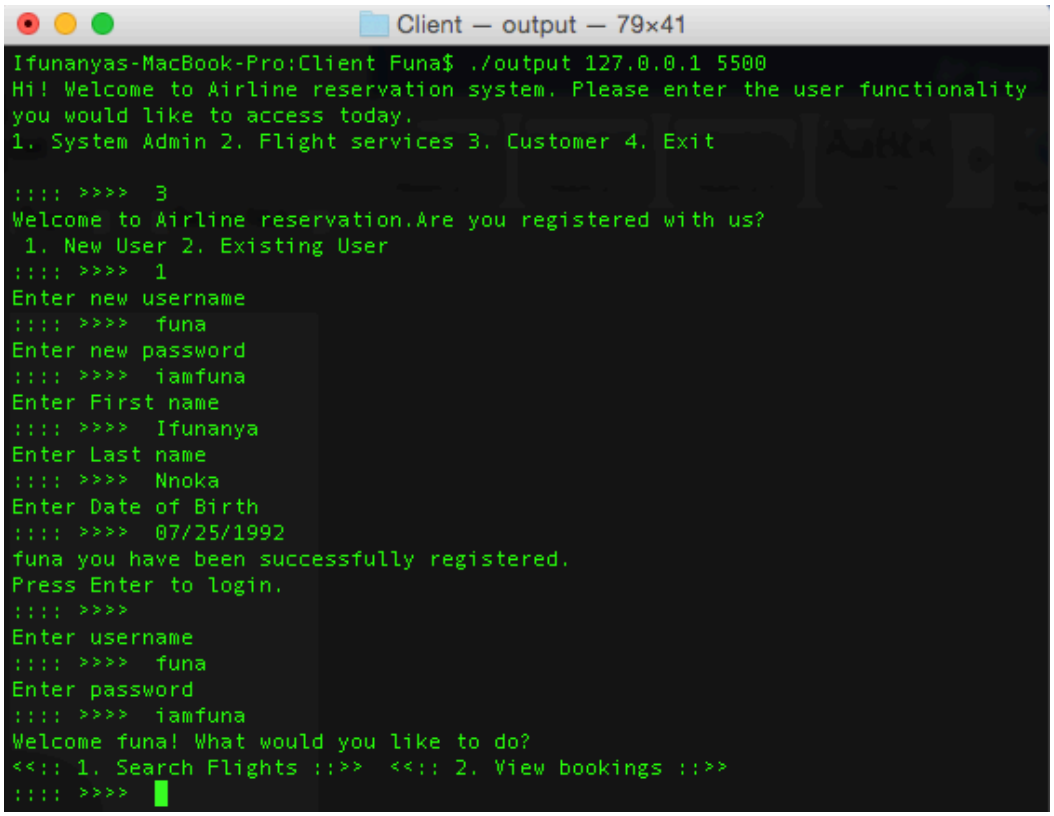
```
//authenticate login details
pswd_hash = hash_gen_md5(password, strlen(password));
pthread_mutex_lock(&loginLock);
j = login_user(user_hash,pswd_hash);
pthread_mutex_unlock(&loginLock);
    if (j == 1){
    printf("%s",wClientMsg);
    point = &toclient[0];
    bzero(temp, 256);
    bzero(temp2, 256);
    ptemp = &loginSuccessMsg[0];
    strcpy(temp, ptemp);
```

This procedure ensures that only one thread accesses the data item at a time. The first thread that calls pthread_mutex_lock is given access, all succeeding threads that call pthread_mutex_lock are blocked by the system until the mutex has been released (unlocked). Then the next thread is given access. This way not more than one thread accesses “id_k” and callback2() at any instance.

Case: Two Threads Registering the Same Username Simultaneously

Customer Registration and Authentication

First time users are required to create an account prior to accessing customer services. Necessary customer information is stored into the database for future use, these include: First name, Last name, Date of Birth, Hash value of username and Hash value of password.



Here we see the new customer account has been created.

	username	password	First_name	Last_name	Date_of_Birth
	Filter	Filter	Filter	Filter	Filter
1	b4cc344d25a2efe540adbf2678e2304c	1b5e491e4e4cf9448e272926359c7f3e	James	Brown	05/03/1933
2	ae074a5692dfb7c26aae5147e52ceb40	36d897b84869b87380039060432 6d6a1	Kelly	Anderson	12/12/1993
3	20c79ef8a4fcc250be4702fbd5045ffb	2bb723cca2051285ef9a222ea5306a1d	Amara	Agwuh	05/11/1990
4	332532dcfaa1cbf61e2a266bd723612c	c7499c207510b511e734520d8b7d843b	Sam	Cooke	01/22/1931
5	770cd84cc2e08d72502d5ed19dab8090	0dff91568b7a3459f44255aa811001a0	Jermaine	Cole	01/28/1985
6	bfe8c2cc0739b21db14efb4246ff30d8	2a9b3fca1d56c741b44d38e487cdf3ea	Ifunanya	Nnoka	07/25/1992

Encryption Algorithm and Data Integrity:

To verify users and protect their account, we have implemented MD5 cryptographic functions into our program. MD5 is a powerful hashing algorithm that maps an
arbitrary long piece of plaintext to a fixed-length string of 128 bits, this string is known as its message digest (analogous to a fingerprint). Due to its one-way function characteristics, it is computationally impossible to compute the plaintext given the message digest, and a string of text will always be mapped to a specific message digest when hashed. Based on these facts, during customer registration we can safely retrieve a username and password and store them as a message digest, thereby preventing those with access to the database from retrieving a user's login information.

```
char *hash_gen_md5(const char *field, int len) {
    int i;
    MD5_CTX p;
    unsigned char digest[MD5_DIGEST_LENGTH];
    char *hash = (char*)malloc(33);

    MD5_Init(&p);
    while (len > 0) {
        if (len > 512) {
            MD5_Update(&p, field, 512);
        } else {
            MD5_Update(&p, field, len);
        }
        len -= 512;
        field += 512;
    }
    MD5_Final(digest, &p);
    for (i = 0; i < sizeof(digest); ++i) {
        snprintf(&(hash[i*2]), MD5_DIGEST_LENGTH*2,
                 "%02x", (unsigned int)digest[i]);
    }
    return hash;
}
#endif /* customer_h */
```

The function above generates a message digest for a given username/password and returns its hash value, a 32-byte hexadecimal value which suffice as a fingerprint for the given username/password. The computed hash values of user credentials entered at login should always match the hash recorded during registration.

Upon, registration the "hash_gen_md5" function is called to compute the message digest of the customer's user name and password before the account is finally created.

```
pswd_hash = hash_gen_md5(password, strlen(password));
user_hash = hash_gen_md5(username, strlen(username));
p = createAccount(user_hash, pswd_hash, first_name, last_name,dateofBirth);
free(user_hash);
free(pswd_hash);
```

6	bfe8c2cc0739b21db14efb4246ff30d8	2a9b3fca1d56c741b44d38e487cdf3ea	Ifunanya	Nnoka	07/25/1992

Shown above, the slots for username and password in the customer account table are both filled with the corresponding 32-bit hexadecimal hash values of the user's login credentials.

User Authenticity:

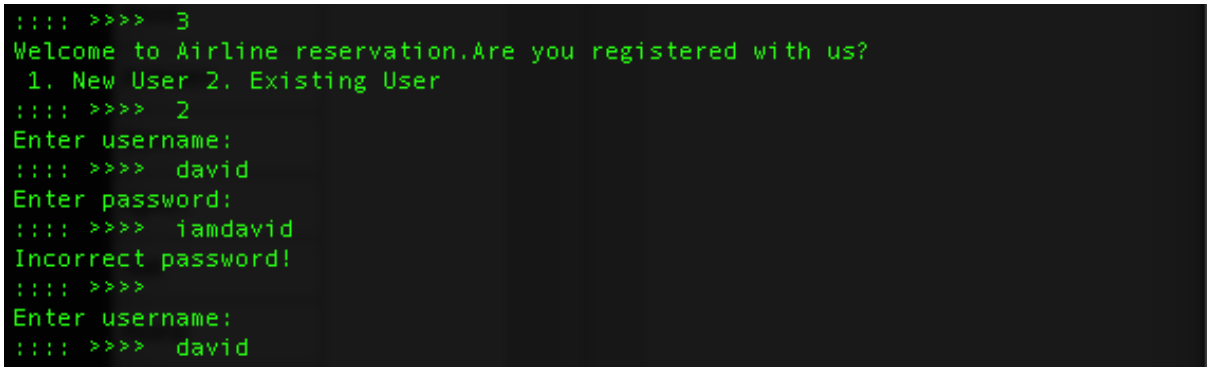
Upon registration customers may login to search for flights or view their booking information. After a customer enters their login credentials, the information is extracted and hashed to produce the hash value for the username and password entered by the customer. The next procedure is to ensure the customer account exists in the database, this is done by comparing the hash value of the username given by the customer with all of the username hash values stored in the database. If the program does not find a match the customer is prompted to enter a valid username. If the username exists in the database, the program then compares the hash value of the password entered by the user with the hash value of the password stored in the database for that account. If the hash values match, the customer is given access to the account and all of the services offered, else the customer is notified that the password entered is invalid and therefore prompted to enter their correct login credentials.

```
//check if username exists in server first
user_hash = hash_gen_md5(username, strlen(username));
bp = &buffa[0];
bp += sprintf(bp,"%s",slctUsrMsg);
bp += sprintf(bp,"'%s';",user_hash);
pthread_mutex_lock(&authLock);
j = authenticate(buffa);
pthread_mutex_unlock(&authLock);
if (j == 1) {
//user does not exist in server
```

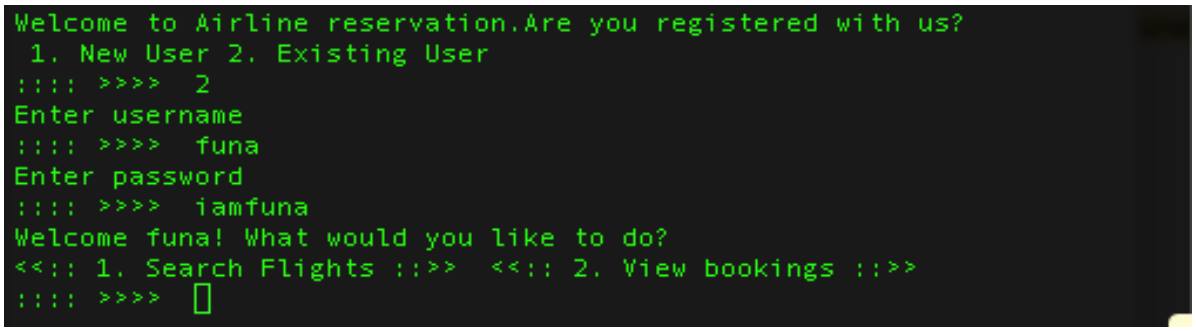
```
:::: >>>>  3
Welcome to Airline reservation.Are you registered with us?
 1. New User 2. Existing User
:::: >>>>  2
Enter username
:::: >>>>  judy
Enter password
:::: >>>>  iamjudy
Username judy does not exist in our server!!
Enter '1' to use a different username '2' to exit
:::: >>>>
```

Case: User Account Not Recorded in Server.

```
//authenticate login details
pswd_hash = hash_gen_md5(password, strlen(password));
pthread_mutex_lock(&loginLock);
j = login_user(user_hash,pswd_hash);
pthread_mutex_unlock(&loginLock);
if (j == 1){
    //successfully logged in
    printf("%s",wClientMsg);
    point = &toclient[0];
    bzero(temp, 256);
    bzero(temp2, 256);
    ptemp = &loginSuccessMsg[0];
    strcpy(temp, ptemp);
    memmove(temp2, ptemp+8, strlen(ptemp+8)+1);
    memcpy(&temp[8], username, strlen(username));
    memcpy(&temp[8+strlen(username)], temp2, strlen(temp2));
    point += sprintf(point,"%s",temp);
    n1=write(ssock,toclient,strlen(toclient));
    if (n1 <= 0){
        errexit("ERROR writing to socket\n");}
    pass = 1; j = 0;
}else{
    //login failed
```



Case: Incorrect Password Entered for User Account



Case: Correct Password Entered for User Account

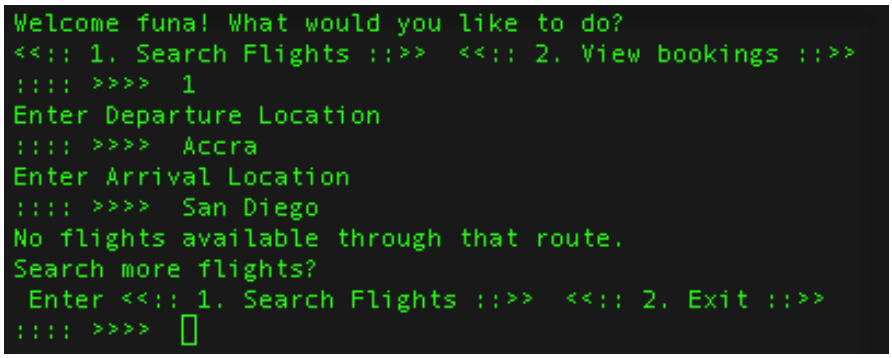
Customer Flight Search

When a customer chooses to search for a flight, two queries are returned to the client: (1) Departure Location Query, (2) Arrival Location Query. Once location information is retrieved from the client, the program calls function search_flight(depart, arrive) which returns a value indicating that a flight(s) exists or does not exist for those parameters. search_flight() executes its callback function "flight_identifier()". flight_identifier() does two major things (1) accesses the database and calls back the flight details which match the (arrive,depart) parameters and puts it in a global buffer "call_bus", (2) assigns a value to a global variable "f_ok" which indicates whether a flight(s) exists or does not exist for parameters (arrive,depart). Clearly there is a need to protect the search_flight() function and all subsequent global resources associated with it in our multi-threaded environment. This has been achieved by implementing pthread mutex synchronizing functions.

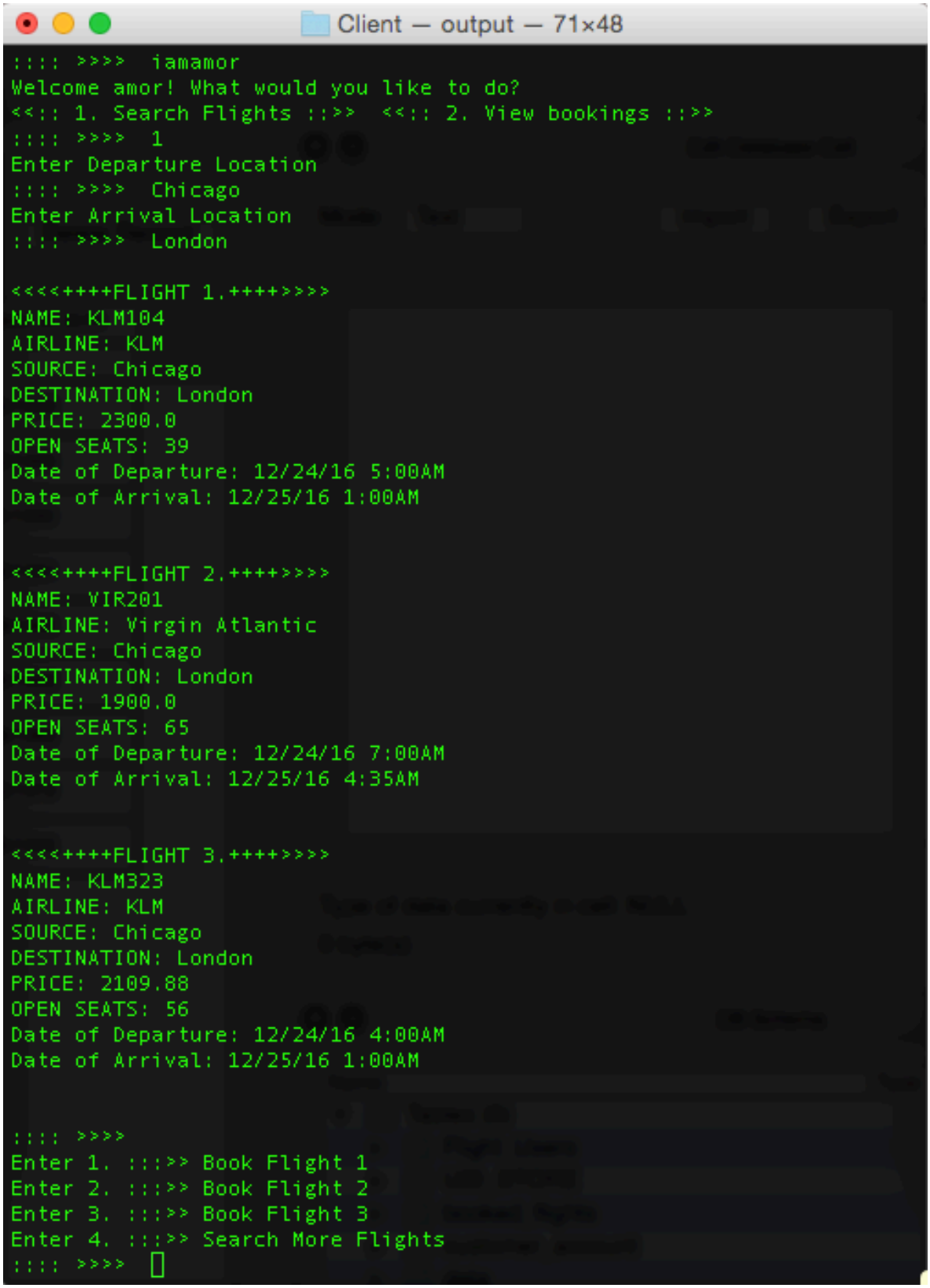
```
int flight_identifier(void *data, int argc, char **argv, char **azColName){
    int i;
    f_count++;
    //counts how many flight matched the client's search parameters
    cb3 += sprintf(cb3,"\n<<<<++++FLIGHT %d.++++>>>>\n",f_count);
    for(i=0; i<argc; i++){
        //extracts flight parameters if there is a match
        cb3 += sprintf(cb3,"%s: %s\n", azColName[i], argv[i] ? argv[i] : "NULL");
        //used to extract flight number/name parameter
        if (strncmp("NAME", azColName[i], 4)==0) {
            pp +=sprintf(pp,"%s", argv[i]);
        }
        //if null no flight matched client's search parameter
        if (argv[i] == NULL) {
            f_ok = 0;
        }else f_ok = 1;
    }
    cb3 += sprintf(cb3,"\n");
    return 0;
}
```

Since a call to search_flight uses flight_identifier as a callback function, to allow for thread synchronization, we lock its mutex item before the call and unlock after use.

```
bzero(local_cb, sizeof(local_cb));
pthread_mutex_lock(&searchLock);
p = search_flight(depart,arrive);
bzero(local_buf, sizeof(local_buf));
strcpy(local_buf, buf);
strcpy(local_cb, call_bus2);
pthread_mutex_unlock(&searchLock);
int _count = (strlen(local_buf)/6.0);
if (p != 1){
    //no flights matched
```



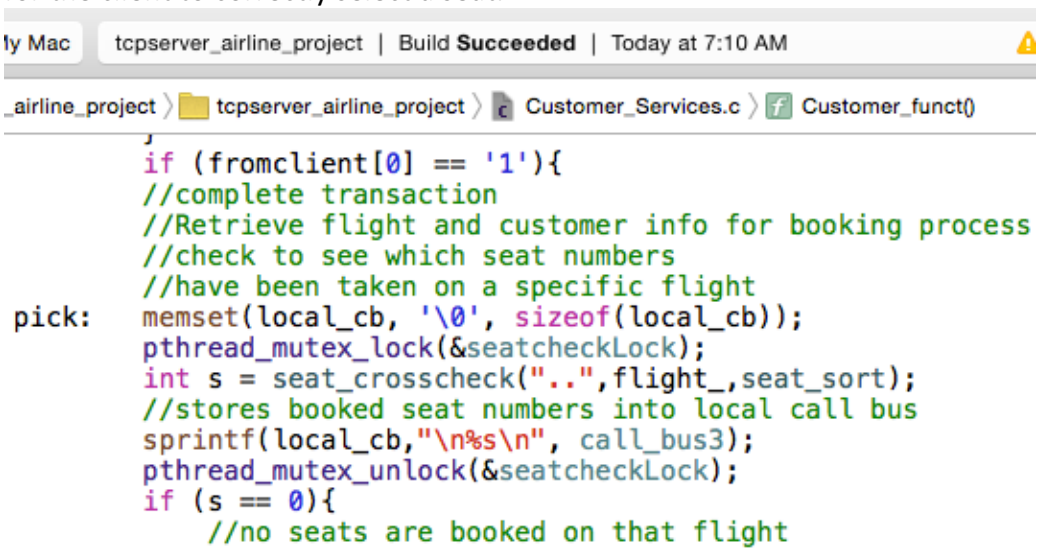
Case: No Flights Available



Case: Flights Available

Book Flight:

After a client picks a flight, the available seats information is extracted and returned to the client in order for the client to correctly select a seat.




```
        if (fromclient[0] == '1'){
        //complete transaction
        //Retrieve flight and customer info for booking process
        //check to see which seat numbers
        //have been taken on a specific flight
pick:   memset(local_cb, '\0', sizeof(local_cb));
        pthread_mutex_lock(&seatcheckLock);
        int s = seat_crosscheck("..",flight_,seat_sort);
        //stores booked seat numbers into local call bus
        sprintf(local_cb,"\n%s\n", call_bus3);
        pthread_mutex_unlock(&seatcheckLock);
        if (s == 0){
            //no seats are booked on that flight
```

Below we find that after a client has selected a seat, the selection is compared with already booked seats before it can be reserved for the client. In order to ensure thread synchronization, a mutex "seatcheckLock" is used to block subsequent clients from using seat_crosscheck. After reserving the seat or after deducing that the client picked an already booked seat the thread unlocks the mutex. Also during this process function retrieve_transaction is used to generate a transaction number for the client before recording the information to the database. The retrieve_transaction function also has to be executed before the mutex "seatcheckLock" is unlocked, this ensures that no 2 threads generate the same transaction number. The mutex item gettransLock is used with retrieve_transaction because it has more than one function and is called at other parts of the program. After these tasks have been accomplished, the flight is reserved and the resources become free for the next thread in the queue.

```
/* Compare client input with already reserved seats*/
pthread_mutex_lock(&seatcheckLock);
int x = seat_crosscheck(fromclient,flight_,seat_sort);
if (x == 1) {
    //client picked already booked seat
    pthread_mutex_unlock(&seatcheckLock);
    goto pick;
}else{
    //Books flight with valid seat num chosen
    //unlocks seatcheckLock only after flight is booked
    //gettransLock (a lock on the transaction number generator)
    //is unlocked only after flight is booked
    strcpy(seat_number,fromclient);
    order_no.i = getOrderInfo("0",gettransLock, retrieve_transaction);
    p =  BookFlightx(order_no.i, flight_, user_hash,
    first_name, last_name, airline, depart, arrive,
    seat_number, price, date_depart, date_arrive);
    pthread_mutex_unlock(&seatcheckLock);
}
    if (p == 1) {
        /*Successful booking*/
        //update available seats info in database
```

Below is a sample of a client attempting to book an already booked seat. Client is returned a list of all the already booked seats, and asked to choose from the available seats.

```
:::: >>>>
Enter 1. :::>> Book Flight 1
Enter 2. :::>> Book Flight 2
Enter 3. :::>> Book Flight 3
Enter 4. :::>> Search More Flights
:::: >>>>  1
Enter <<:: 1. Complete Transaction ::>>  <<:: 2. Cancel ::>>
:::: >>>>  1
Seats already booked:
 2, 5, 13, 4, 10, 22,
 Choose a Seat Number to Book Between Seat 1 - Seat 45.
:::: >>>>  13
Seats already booked:
 2, 5, 13, 4, 10, 22,
 Choose a Seat Number to Book Between Seat 1 - Seat 45.
:::: >>>>
```

Case: Seat Number Already Booked

```
Seats already booked:
 2, 5, 13, 4, 10, 22,
 Choose a Seat Number to Book Between Seat 1 - Seat 45.
:::: >>>>  9

Congratulations! Your flight reservations have been made.
Your Order Number is 11.
Enter 1 to view your flight reservations.
:::: >>>>  1

Transaction No.: 11
Flight No.: KLM104
First Name: Amor
Last Name: Chan
Airline: KLM
From: Chicago
To: London
Seat No.: 9
Date of Depature: 12/24/16 5:00AM
Date of Arrival: 12/25/16 1:00AM
Price: 2300.0


:::: >>>>
```

Case: Book flight Successful

Shown below is the new reservation information added to the database.

	order_no	Flight	user	First_name	Last_name	Airline	Source	Destination	seat_number	Price	Date_depart	Date_arrive
	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	1	KLM104	c0ec3b2d3be484b9bbfc0a2e040e87f6	Uju	Nwauwa	KLM	Chicago	London	2	2300.0	12/24/16 5:00AM	12/25/16 1:00AM
2	2	KLM104	b4cc344d25a2efe540adbf2678e2304c	James	Brown	KLM	Chicago	London	5	2300.0	12/24/16 5:00AM	12/25/16 1:00AM
3	3	KLM104	ae074a5692dfb7c26aae5147e52ceb40	Kelly	Anderson	KLM	Chicago	London	13	2300.0	12/24/16 5:00AM	12/25/16 1:00AM
4	4	LUF103	172522ec1028ab781d9dfd17eaca4427	David	Stone	Lufthanza	San Jose	Morocco	5	1294.0	12/29/16 11:00AM	12/30/16 1:00PM
5	5	KLM104	fc292bd7df071858c2d0f955545673c1	Arinze	Nnoka	KLM	Chicago	London	4	2300.0	12/24/16 5:00AM	12/25/16 1:00AM
6	6	KLM106	20c79ef8a4fcc250be4702fbd5045ffb	Amara	Agwuh	KLM	Poland	Germany	15	1100.0	11/25/16 12:00PM	11/25/16 11:00PM
7	7	DEL440	332532dcfaa1cbf61e2a266bd723612c	Sam	Cooke	Delta	San Diego	Sacramento	4	350.0	02/03/17 6:00PM	02/03/17 7:45PM
8	8	KLM104	770cd84cc2e08d72502d5ed19dab8090	Jermaine	Cole	KLM	Chicago	London	10	2300.0	12/24/16 5:00AM	12/25/16 1:00AM
9	9	DEL102	ae074a5692dfb7c26aae5147e52ceb40	Kelly	Anderson	Delta	Lagos	Atlanta	7	2200.0	12/24/16 12:45PM	12/25/16 10:00AM
10	10	KLM104	bfe8c2cc0739b21db14efb4246ff30d8	Ifunanya	Nnoka	KLM	Chicago	London	22	2300.0	12/24/16 5:00AM	12/25/16 1:00AM
11	11	KLM104	5da2297bad6924526e48e00dbfc3c27a	Amor	Chan	KLM	Chicago	London	9	2300.0	12/24/16 5:00AM	12/25/16 1:00AM

Below is an example of the server updating the available seats information right after booking the seat. An unlock of mutex retrLock is done after every select_() call because retrLock is used to lock a function retrieve() called in select_(). retrieve() is responsible for returning information from tables in the database, it puts all of this information in a global buffer which must be protected to avoid intrusion.

```
if (p == 1) {
    /*Successful booking*/
    //update available seats info in database
    pthread_mutex_lock(&updateLock);
    //get most updated number of open seats before update
    select_(0,"open_seats", "data", "name",(void*)flight_);
    strcpy(open_seats.str, call_bus4);
    pthread_mutex_unlock(&retrLock);
    open_seats.i = atoi(open_seats.str);
    open_seats.i--;
    update(5,"data","open_seats",(void*)open_seats.i,"name",flight_);
    pthread_mutex_unlock(&updateLock);
```

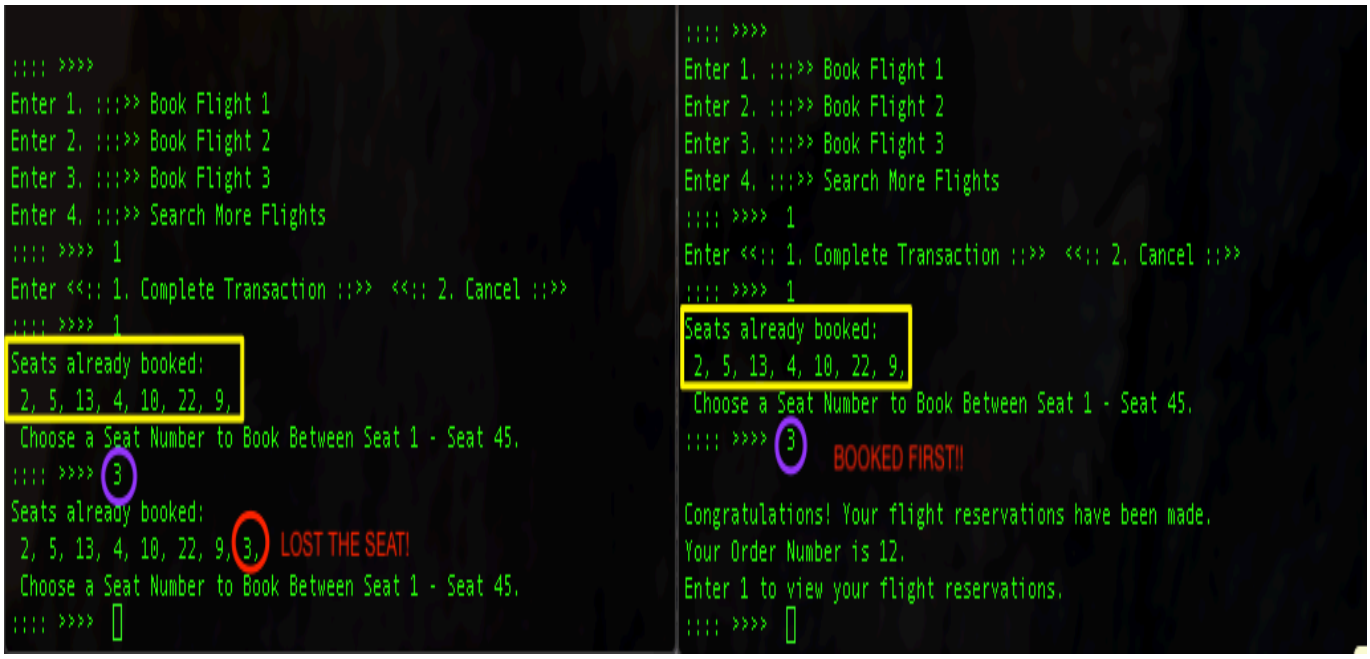
```
Client — output — 71×48
Enter Arrival Location
:::: >>>>  London

<<<<++++FLIGHT 1.++++>>>>
NAME: KLM104
AIRLINE: KLM
SOURCE: Chicago
DESTINATION: London
PRICE: 2300.0
OPEN SEATS: 39
Date of Departure: 12/24/16 5:00AM
Date of Arrival: 12/25/16 1:00AM
```

	NAME	UID	AIRLINE	SOURCE	DESTINATION	PRICE	TOTAL_SEATS	OPEN_SEATS	DATE_DEPATURE	DATE_ARRIVAL
	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	KLM104	100	KLM	Chicago	London	2300.0	45	38	12/24/16 5:00AM	12/25/16 1:00AM

Database Update: open_seats is updated after booking

Below is a case of two client's attempting to book the same seat number on the same flight simultaneously. Implementing the algorithm below and then unlocking "seatcheckLock" only after the seat has been reserved/booked ensures that no more than one client books a seat number.

```
/* Compare client input with already reserved seats*/
pthread_mutex_lock(&seatcheckLock);
int x = seat_crosscheck(fromclient,flight_,seat_sort);
if (x == 1) {
     //client picked already booked seat
     pthread_mutex_unlock(&seatcheckLock);
```



Case: 2 Clients attempt to book the same seat

View Booked Flight:

Clients are given the option to view all of their booking information or view by order number. If a client would like to view a booking detail by order number, the order number is first compared with data in the database to confirm that it exists and that it belongs to the client before the client can be given any information.

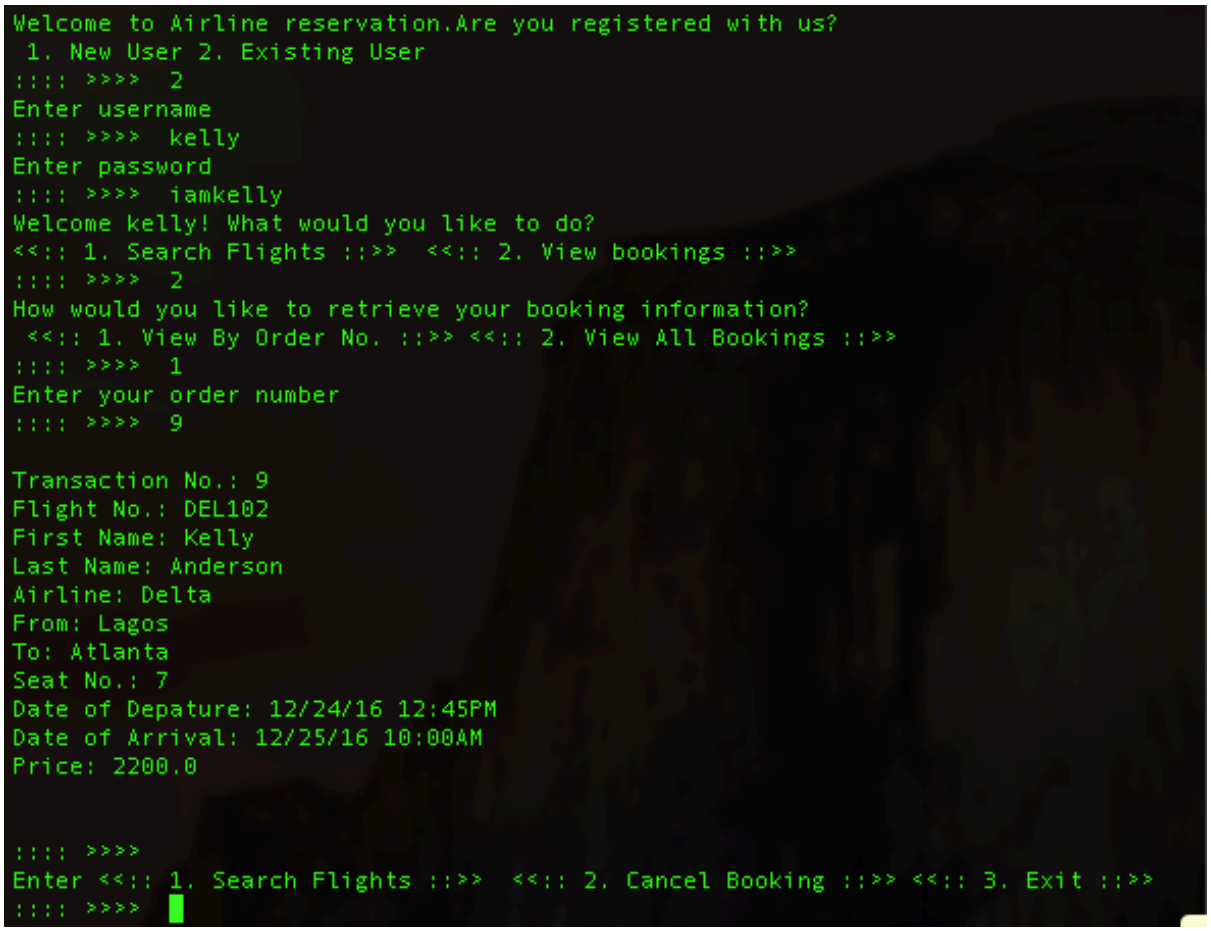
```
strcpy(order_no.str, fromclient);
//check if transaction number exists
//and also if it matches user
j = getOrderInfo(order_no.str,orderLock,confirm_order_);
if (j != 1) {
/*Transaction number does not match our records*/
}else{
    order_no.i = atoi(order_no.str);
    bzero(buffa, 1001);
    select_(2, "user", "booked_flights", "order_no", (void*)order_no.i);
    strcpy(user_temp, call_bus4);
    pthread_mutex_unlock(&retrLock);
    if (strcmp(user_temp, user_hash)!= 0){
    /*Transaction number assigned to a different client*/
```

```
Welcome to Airline reservation.Are you registered with us?
 1. New User 2. Existing User
:::: >>>> 2
Enter username
:::: >>>> kelly
Enter password
:::: >>>> iamkelly
Welcome kelly! What would you like to do?
<<:: 1. Search Flights ::>>  <<:: 2. View bookings ::>>
:::: >>>> 2
How would you like to retrieve your booking information?
 <<:: 1. View By Order No. ::>> <<:: 2. View All Bookings ::>>
:::: >>>> 1
Enter your order number
:::: >>>> 4
Order number does not match your records
:::: >>>>
How would you like to retrieve your booking information?
 <<:: 1. View By Order No. ::>> <<:: 2. View All Bookings ::>>
:::: >>>>
```

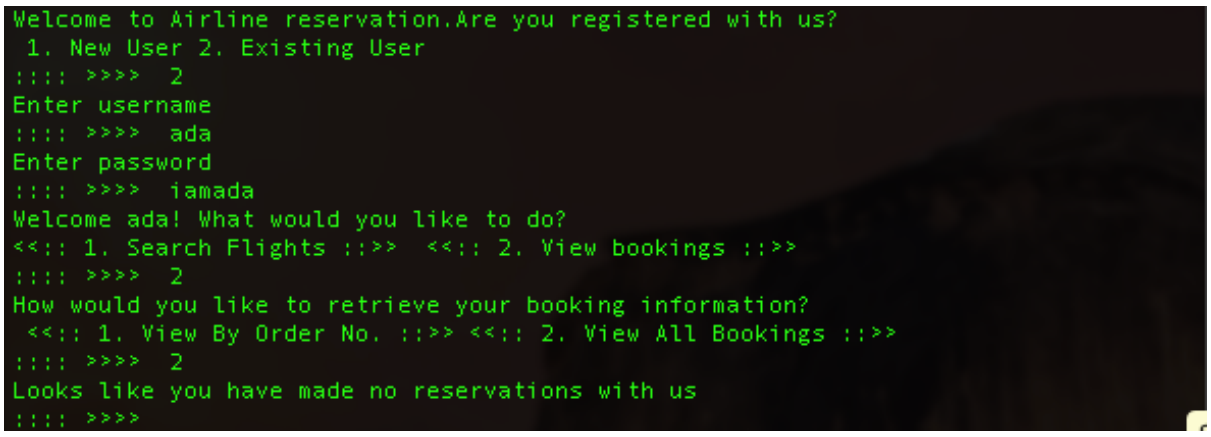
Case: Order number does not match client's record

```
:::: >>>>
How would you like to retrieve your booking information?
 <<:: 1. View By Order No. ::>> <<:: 2. View All Bookings ::>>
:::: >>>> 1
Enter your order number
:::: >>>> 15
Order number not in our records.
 Please Enter a valid order number.
:::: >>>>
```

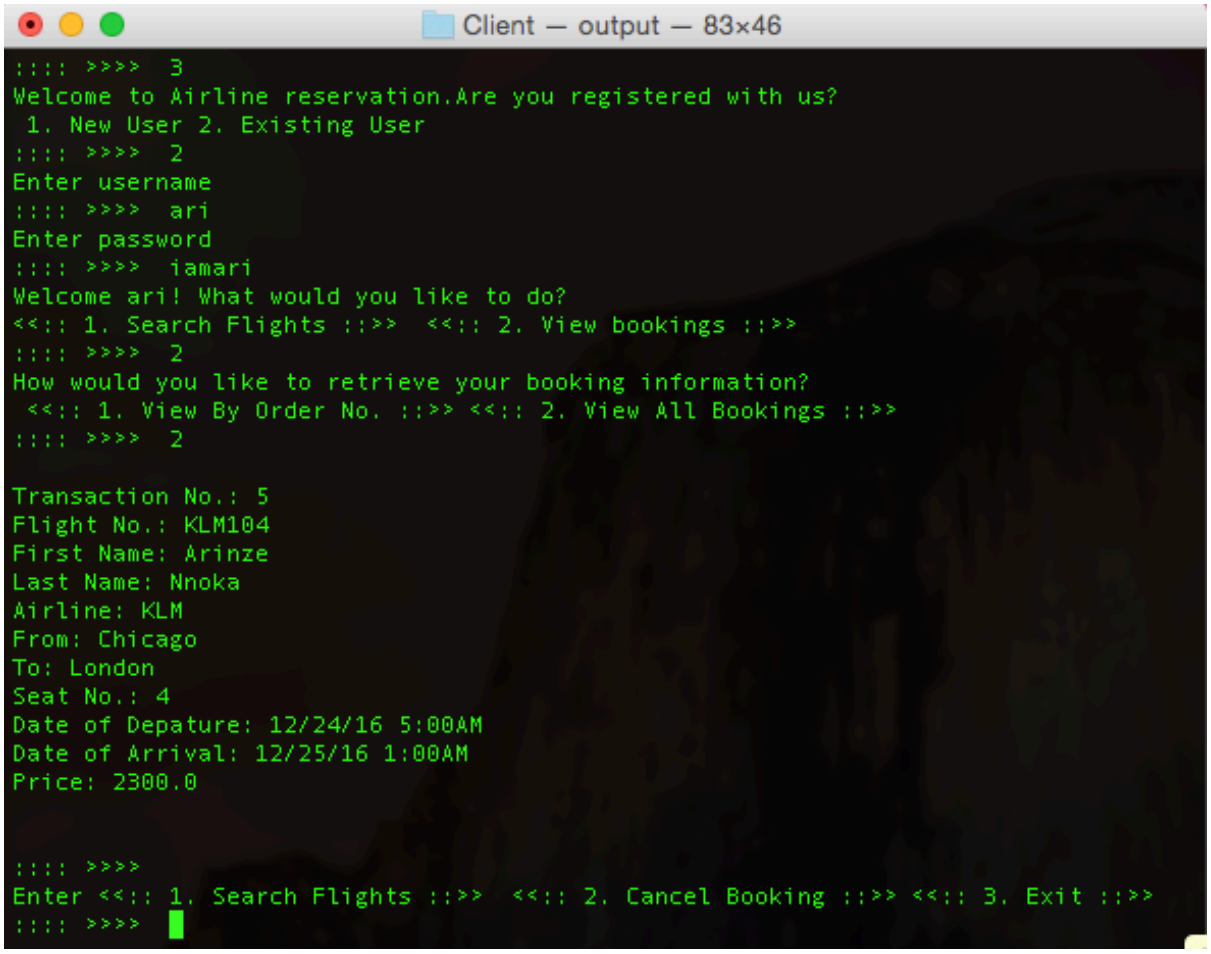
Case: Order number not recorded in server



Case: View by order no. with correct order no.



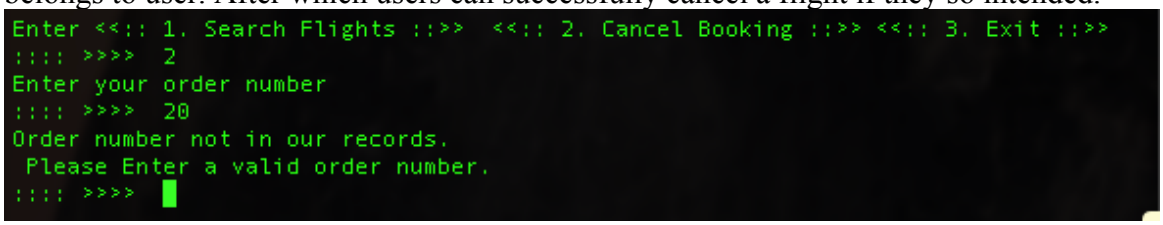
Case: Client has no reservation



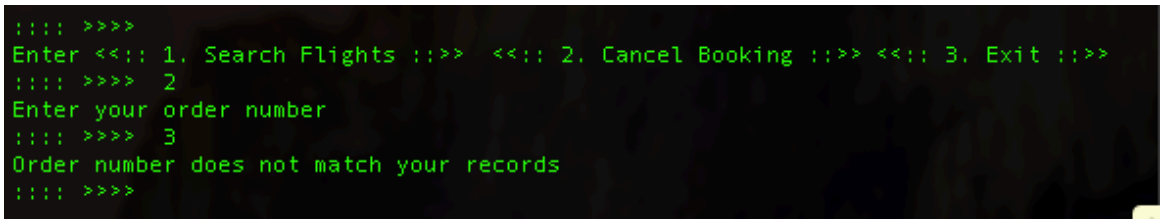
Case: View all bookings (via login only)

Cancel Flight Reservation:

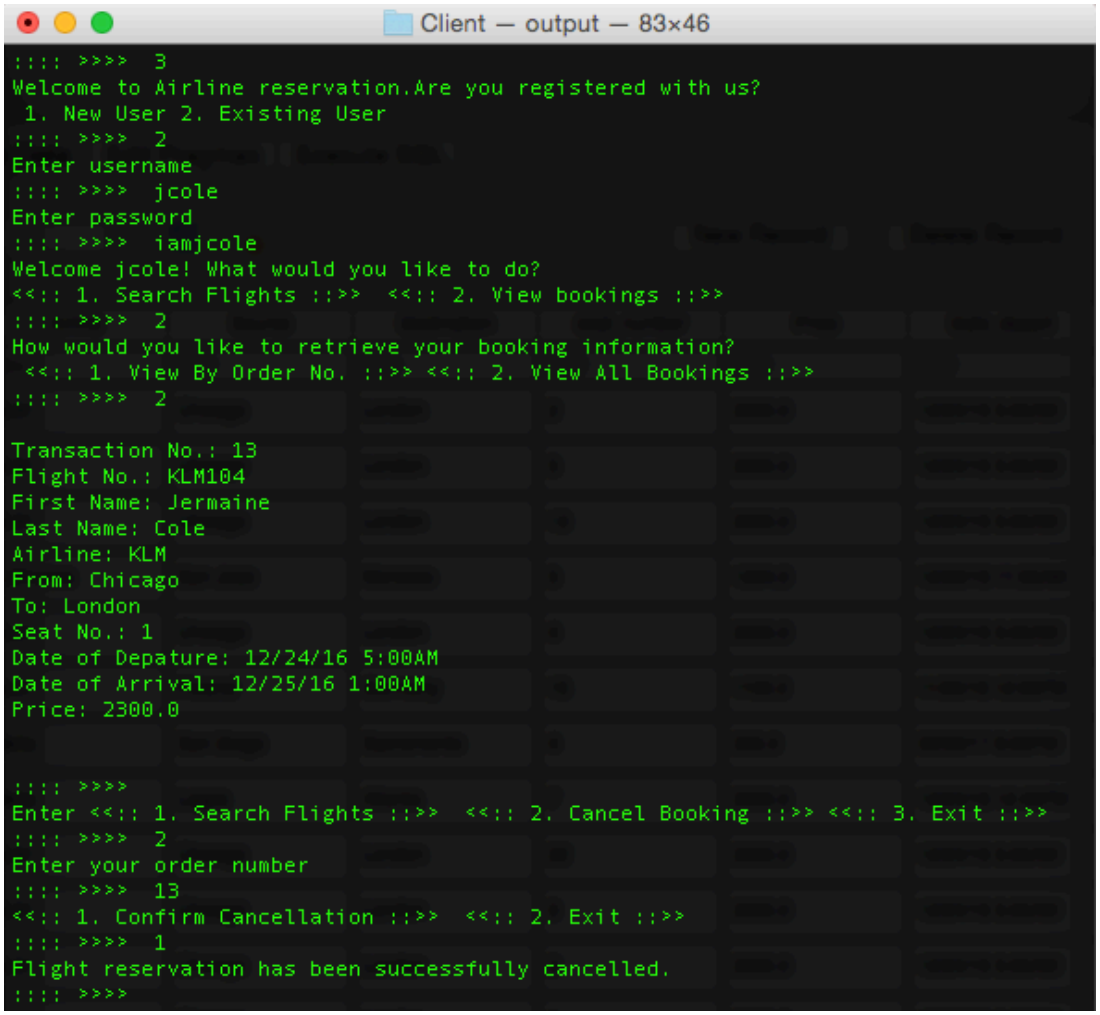
The first procedures for cancelling a flight reservation is similar to that of the “view booking”. An order number is retrieved from client and then compared with database information to make sure it exists and belongs to user. After which users can successfully cancel a flight if they so intended.



Case: Cancel booking with incorrect order no.



Case: Cancel booking with order no. belonging to another client



Case: Booking successfully cancelled

```
:::: >>>>
Welcome to Airline reservation.Are you registered with us?
 1. New User 2. Existing User
:::: >>>>  2
Enter username
:::: >>>>  jcole
Enter password
:::: >>>>  iamjcole
Welcome jcole! What would you like to do?
<<:: 1. Search Flights ::>>  <<:: 2. View bookings ::>>
:::: >>>>  2
How would you like to retrieve your booking information?
 <<:: 1. View By Order No. ::>> <<:: 2. View All Bookings ::>>
:::: >>>>  2
Looks like you have made no reservations with us
:::: >>>>
```

Case: Client finds no reservation on account after cancellation

Shown below, open seats information changed for flight KLM104 in airline database after customer cancels a flight.

	NAME	UID	AIRLINE	SOURCE	DESTINATION	PRICE	TOTAL_SEATS	OPEN_SEATS	DATE_DEPATURE	DATE_ARRIVAL
	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	KLM104	100	KLM	Chicago	London	2300.0	45	36	12/24/16 5:00AM	12/25/16 1:00AM
2	KLM106	100	KLM	Poland	Germany	1100.0	56	55	11/25/16 12:00PM	11/25/16 11:00PM

NAME	UID	AIRLINE	SOURCE	DESTINATION	PRICE	TOTAL_SEATS	OPEN_SEATS	DATE_DEPATURE	DATE_ARRIVAL
Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
KLM104	100	KLM	Chicago	London	2300.0	45	37	12/24/16 5:00AM	12/25/16 1:00AM
KLM106	100	KLM	Poland	Germany	1100.0	56	55	11/25/16 12:00PM	11/25/16 11:00PM

SERVER AND CLIENT CODE

https://drive.google.com/drive/folders/1Vk5NStMGXwGLvh9M1uwHryb2wcayMLSC?usp=sharing

PROJECT SCHEDULE & PLAN

a) Features planned

➢ ***System Administrator:*** Addition and deletion of Airlines, Car rental and Hotel services.	➢ ***Airline Professionals:*** 1. Addition and deletion of flights 2. Modifying the details of flights 3. View number of people booked for a particular flight
➢ ***Customers:*** 1. View/Modify Profile details. 2. Book flights. 3. View flights. 4. Cancel flight	➢ ***Login:*** 1. Airline professionals and system admin should have user login and password. 2. Login credentials should be encrypted using MD5

b) Progress

Tasks	Sub-tasks	Status	Planned date
Creation of Client	-	Complete	10/25/2016
Creation of server backbone	-	Complete	10/25/2016
Database management	-	Complete	11/03/2016
System admin features	Add User	Complete	11/04/2016
	View User	Complete	11/10/2016
	Delete User	Complete	11/11/2016
	Modify user details	Complete	11/12/2016

Airline user Functions	Add flights	Complete	11/01/2016
	View Flights	Complete	11/01/2016
	Modify Flight Details	Complete	11/01/2016
	Delete Flights	Complete	11/01/2016
	Delete Airline		
Customer user features	Customer Registration	Complete	11/13//2016
	Search Flights	Complete	11/13/2016
	Book flight		
	View Bookings using transaction number/ Login details	Complete	11/14/2016
	Cancel Booked Flight		
Encryption/Hashing Algorithm	25-bit Encryption Algorithm	Completed	11/16/2016
	MD5 Hashing Algorithm	Completed	11/30/2016
Project Management And Documentation	-	Complete	11/14/2016

FUTURE SCOPE

This project can be further modified to use in various applications like hotel reservation, car rental services etc. Since all these application uses the similar concept of distributed reservation, our project can be modified for various applications. We would have love to implement ipv6, perhaps its something we will still try out at our spare time. This project has thought us a lot about networking concepts, and we have improved on our skills due to this practice.