COMP 4900A - Group 5

Team Project: Restaurant Management System Application

Sean Chung (101156263),
Johnny Lee-De Medeiros (101141885),
Richard Donghoon Kim (101138475),
Mohammad Khan (101155812),

December 9, 2022.

Index

1. Short Title, Brief Description and Link to the Demo Video

The Restaurant Management System Application

For our final project of designing a program that employs a real-time operating system, we decided to implement a Restaurant Management System Application (RMSA) that is built on the QNX Neutrino platform. Our application simulates the interaction between the client (customer) and server (restaurant management system) by properly handling customer requests that are likely to be used in a real production environment. The Restaurant Management System Application handles in-house requests including getting a table in the restaurant, creating a table order, and printing their receipt as well as online requests, namely creating a reservation for a certain date and creating an online order for takeout or delivery. In addition, our program utilizes the QDB database framework for storing crucial information for the restaurant such as restaurant profile, tables, reservations, table meta-tags, menu items, online orders, and table orders.

Link to Demo Video:

Recording:
https://mediaspace.carleton.ca/media/Restaurant%20Management%20System%20Application%20Demonstration%20Video/1_wf46bk34

2. Authors and Individual Responsibilities

Sean Chung

- Responsible for restaurant server handling requests from onlineClient and inHouseClient
    - Handling “get table”, ‘create table order” and “get print receipt” message requests from the inHouseClient.
    - Handling “online reservations” and “create online order” message requests from the onlineClient
    - Creating and handling periodic timer pulse events for checking if reservations have started/ended. Also for reading/writing information to our database using functions in “dbfuncs.h”

Johnny Lee-De Medeiros

- Responsible for the QNX database including its:
    - Table Design and Entity & Relational documentation diagrams
    - Creation of the table schemas within the “schema.sql” DDL file in the “database_files” directory of the project repo.
    - Instructions for installing QDB as well as general knowledge for loading the database with its config file and with its associated schema file. Included in the “database_files” directory of the source code.
    - Intermediate functions for interacting with the QNX database from within source code. Functions are included within the “dbfuncs.h” file in the project repo. Function signatures include code blocks to explain their use, parameters, and return values as well as any side effects on the parameters.

Mohammad Khan

- Responsible for restaurant profile creation in the server (server.c)
    - Implemented a prompt asking for the manager to create a profile if no profile exists in the database.
    - Once logged into the profile, table tags associated with tables (used to represent the different types of tables) and the tables themselves are added into the database to represent that the restaurant has been initialized.
    - Once the tables have been added, a prompt was implemented that asks the restaurant manager to login to the profile that was just created. Proper error handling was added to check if the manager inputs an incorrect login username or password.
    - Once logged in, a new prompt asks if new menu items are to be added to the restaurant. A while loop was implemented to constantly ask the restaurant profile user to add items to the restaurant (this is added to the database) until they no

longer want any other items to be added. The server then enters the main while loop.

Richard Donghoon Kim

- Was responsible for all functionalities included in both the clients (inHouseClient.c, onlineClient.c) and the kitchen (kitchenServer.c).
    - Taking user inputs and building the message structure for “get table”, “create table order” and “get print receipt” requests in the inHouseClient, and communicating the message with the server..
    - Taking user inputs and building the message structure for “online reservations” and “create online order” requests in the onlineClient, and communicating the message with the server.
    - Building message structures for “create table order” and “create online order” and communicating the message with the kitchen server in both clients.
    - Handling “create table order” and “create online order” message requests from the clients in the kitchen server.

Each member wrote the explanation of their section in this report and drew the diagrams necessary.

3. Description of the Domain and the Problems Intended to Solve

The restaurant industry is one of the biggest sectors in the hospitality industry that focuses on food and beverage services to customers. It is one of the most common types of small businesses we can find in our daily lives. While there are several types of restaurants, such as fast food restaurants, cafes, fine dining or buffets, we intend to focus on casual dining restaurants. Our main domain will be restaurants that have tables for customers to dine on. Servers can take orders and deliver food to the table, but will also allow online orders, and pick-up service. The Restaurant Management System Application will seek to aid a business in managing incoming online and table orders, and reservations as well as let the restaurant owner configure a profile for the restaurant along with its associated menu and tables to be stored in the database.

The problem the RMSA seeks to solve is that of communication, planning, and connection involved in managing a restaurant as a commercial business. The program will allow owners to set up a digital profile of their menu and tables to then start taking orders and reservations through client devices which are sent to a server which will then process the requests. The clients will allow the user to input the required data for either an online or table order, or a reservation and then confirm it to be sent to the RMSA's server process. The server itself starts with authentication to log into the restaurant's digital profile and then retrieve the needed data stored in the QNX database to allow for the processing of incoming messages by the clients. When the clients send the orders to the server, it will also send the order information to the kitchen. Thereby allowing the kitchen to receive orders in real time, and organize their cooking arrangements accordingly to go through their busy nights.

4. Outline of Development Technology Used (Frameworks, Languages, and Libraries)

QNX Library functions used:

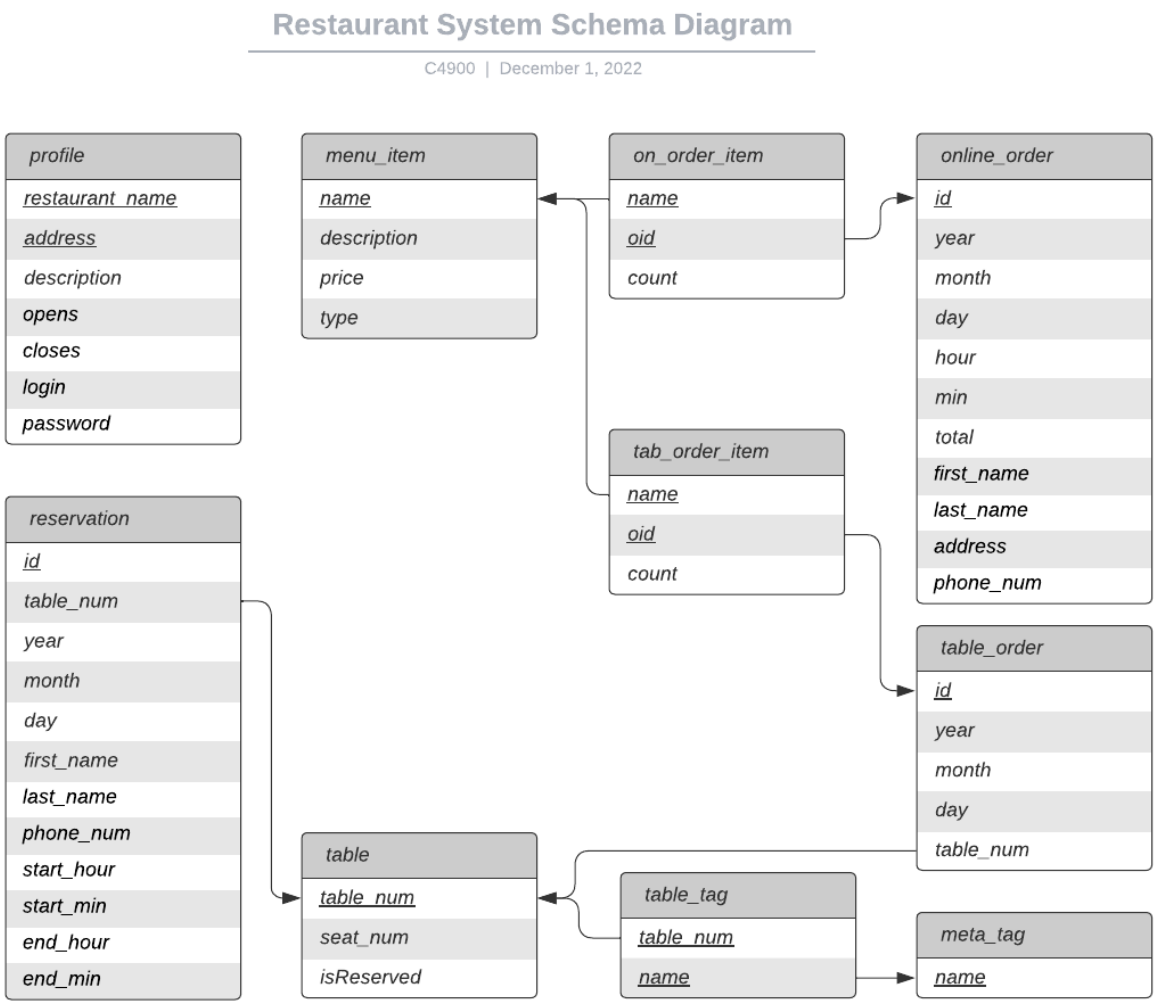
- sys/neutrino.h has useful message-sending functions and channel creation and connections.
- sys/types.h has various useful datatypes like pthread mutexes and conditional variables as well as timers.
- sys/iofuncs.h has non-portable low-level IO definitions
- sys/dispatch.h has functions for opening a connection to a server

QNX Database Specifically:

As described by QNX's user documentation, "The QNX Database (QDB) server is a small-footprint, embeddable SQL database server. It is designed as an easy-to-configure QNX Neutrino resource manager." Using the qdb.h library and its associated executable qdb allows us to utilize an SQLite database as our underlying storage of persistent profile data of the restaurant. The framework acts as means of communication between source code files in C and statements in Search Query Language to be directed to an opened connection to a database running with qdb. This was immensely important to allow for efficient storage and representation of data for the main client-server architecture to use. The QDB client was also useful for quickly looking at existing tables or performing SQL while developing to access the database quickly from within the QNX target virtual machine's command line.

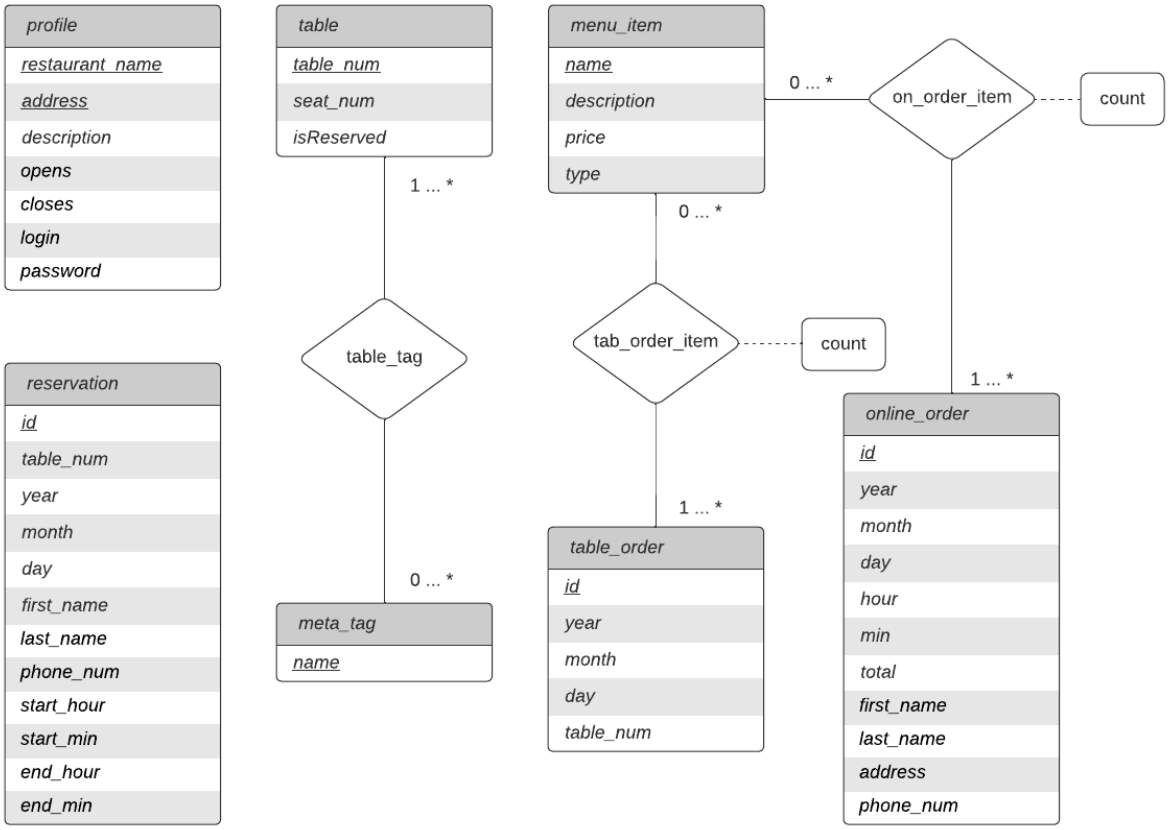
5. System Design and Methodology

Database Design Documentation



Restaurant System ER Diagram

C4900 | December 1, 2022



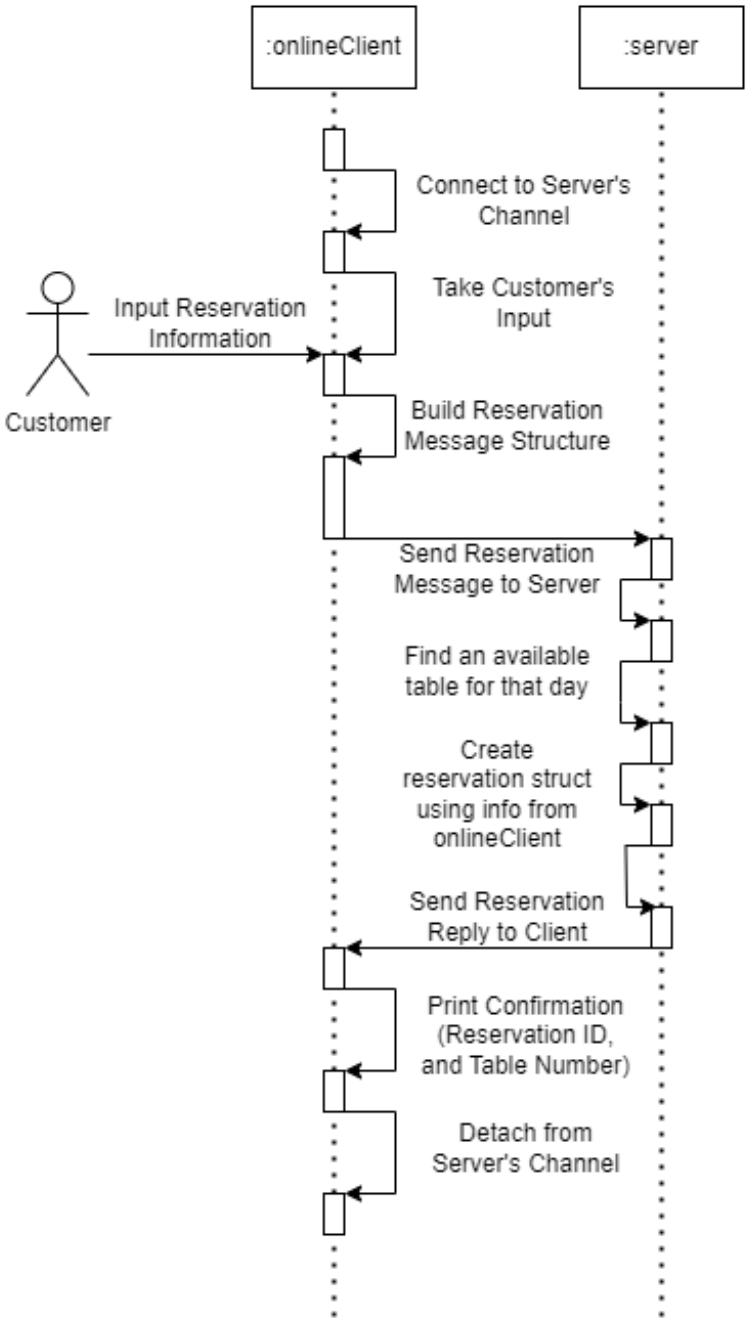
Functionalities

A. Restaurant initialization

Before the server and client can interact with each other, the server needs to initialize a restaurant in order for customers to make reservations and orders. When the server is ran for the first time, it will prompt the restaurant manager to create a profile for their restaurant. Profile creation is initially prompted when no profiles exist in the database (when the server is ran for the first time). Once the restaurant profile has been created a new prompt will ask for the manager to log in. The login information includes the username and password. If the username or password is incorrectly passed, it will output an error message and will continuously ask for the username and password until both values are correct. Once the manager has been logged in to their profile, the table tags representing the different tables a restaurant could have along with the tables themselves are initialized in the restaurant and added to the database.

Once the tables have been initialized in the restaurant, a new prompt will ask the manager if they would like to add new menu items to the restaurant. Initially, when a restaurant profile is created, there are no items currently in the restaurant. The manager has the option of either selecting yes or no to adding new menu items. If they select yes (y) they will be prompted to add the new menu item by filling in the required fields for an item. These fields include the name of the item, the description of the item, the price of the item and the type of the item (beverage, side dish etc.). Once these fields are correctly passed, the menu item will be added to the restaurant (which is added to the database). A message will indicate to the manager that the item has been successfully added and will prompt the manager if they would like to add another menu item. The manager can add a few more items and once they are done they can select the no (n) option and the prompt for adding new items will be closed. A new message will be displayed to the manager that the restaurant has been successfully initialized (including the profile, tables, and menu items being added to the database) and the server can now interact with the client to accept reservations and orders.

B. Reservation



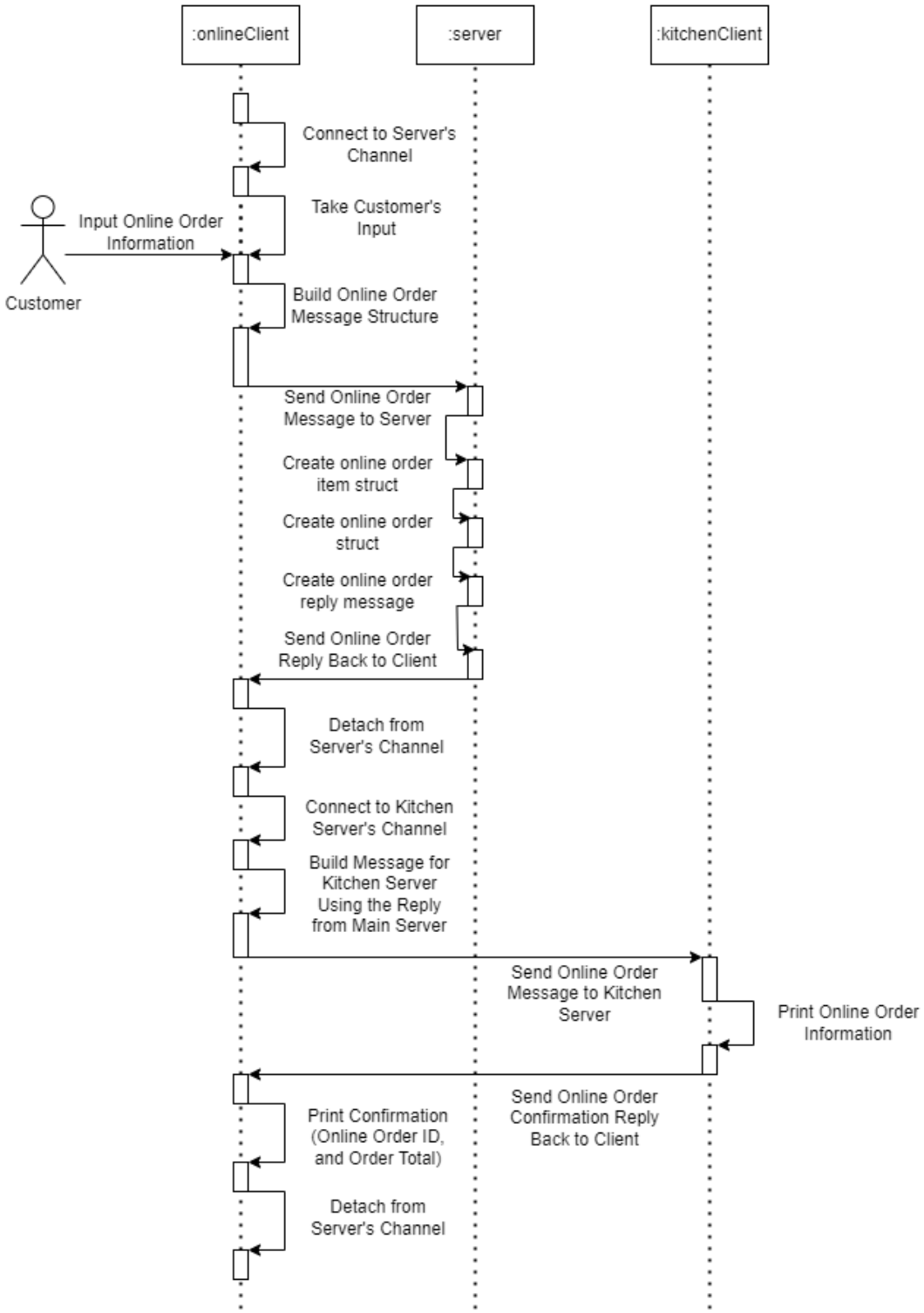
Customers are able to make a reservation via the online client. The online client will allow users to input the customer's information including: their name, phone number, date and time of the reservation, and their preferred type of table (outdoor seats, bar seats, patio, etc.). The online client will then build the message structure using the information, and send the message to the server.

Then the server will search for the most suitable table using the number of seats requested, preferred table type, and the date of the reservation. If an available table is found for the specified time and date, a reservation struct is created using the information that is passed to the server by the online client.

Finally, the server will reply back to the client with the reservation response structure that will contain the reservation ID, and the number of the table reserved.

We also make use of a real-time periodic timer that expires every 60 seconds. Each time the timer expires, the server will search through the reservations that are scheduled for the current day. By comparing the current time (hour and minutes) with the reservation's start and end times (hours and minutes), the server checks if any reservations have started or ended, in order to update the tables that are reserved for reservations to is reserved (1) or is not reserved (0) accordingly.

C. Online order

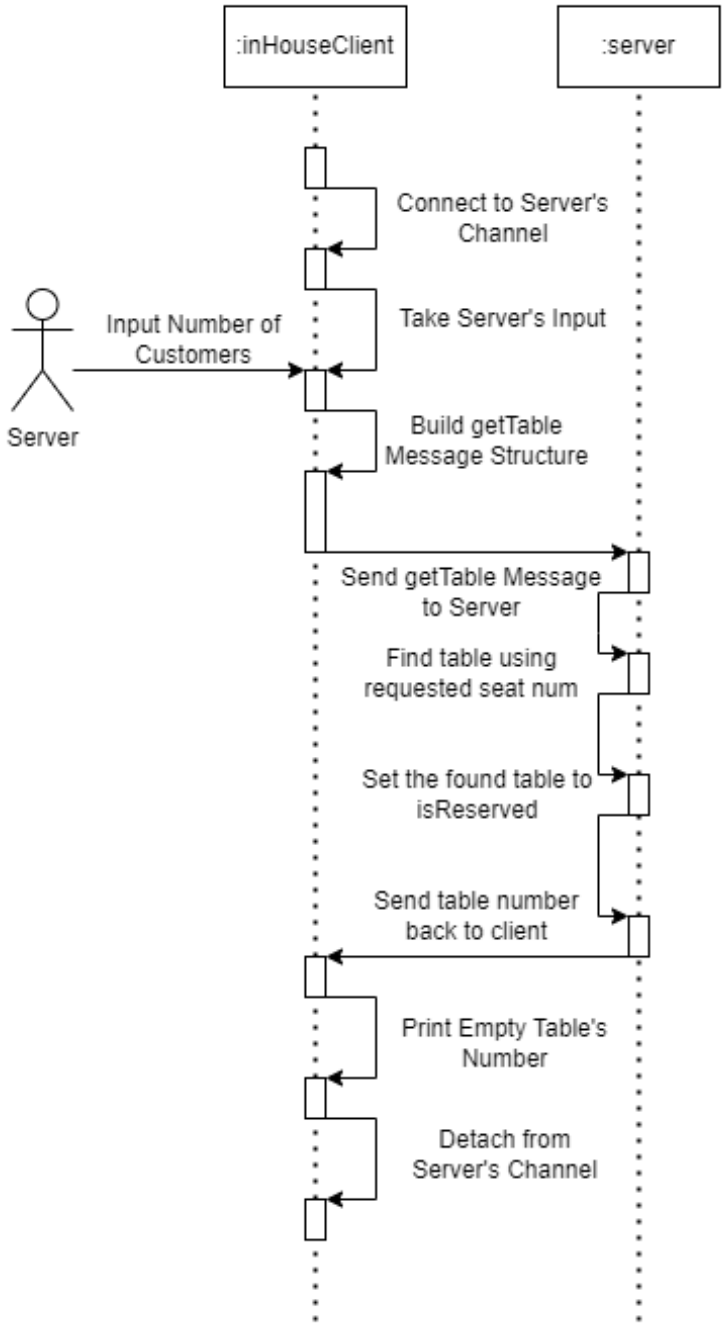


Customers are able to make an online order via the online client. The online client will allow users to input the customer's information including: their name, phone number, address and their actual order. The online client will get the current date and time automatically. The online client will then build the message structure using the information, and send the message to the server.

Then the server will make an online order item, using a struct that consists of the menu item name, a 'count' variable to store how many of the same menu items are ordered, and the order ID, for each menu item that the client has requested. On top of that, the server will create an online order struct, using the online order items, and will use the information passed by the client to create the online order for a requested date and time. Assuming the online order is successful, the server will create a message that contains the order ID and total as a message back to the client.

Next, the server will reply back to the client with the online order response structure that will contain the online order ID and the total price of the online order. After that, the online client will detach from the server's channel, and attach to the kitchen server's channel to communicate with it instead. While connected, it will send the order to the kitchen server, so the kitchen will have access to the new order, and can start cooking. The kitchen server will send a response just to make sure the message communication is complete. Finally, the online client will print the online order ID, and the online order total in proper format before it closes.

D. Getting a Table

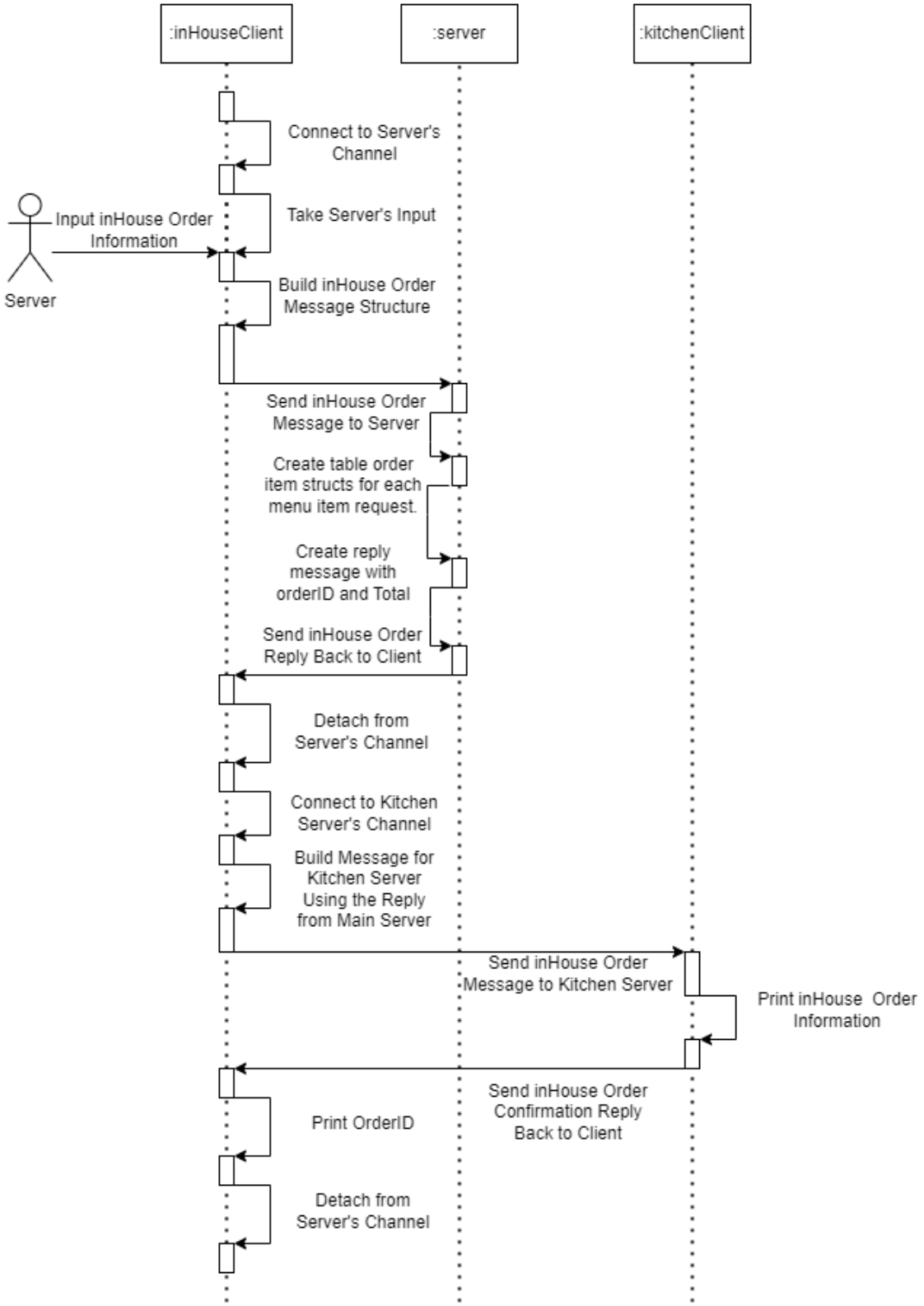


Customers that come in person will be greeted by a server, who will ask for the number of people in their party. Then, the server will be able to input the information on the inHouseClient (which would be installed on a device in real life). The inHouseClient will send the message to the server.

Once the server receives the message from the client indicating how many seats they would like, the server will search for a table with the requested amount of seats. If no tables with the requested number of seats are found, the server will continue searching for a table with the closest number of seats that have been requested by the client. If no tables were found, or the number of seats requested by the client exceeds the maximum number of seats a table can have, a value of -1 is sent as a message back to the client. If a table is found, the server will reply to the client with the corresponding table number.

Finally, the server will reply back to the client with the empty table's number so that the server will be able to guide the customers to their table.

E. In House Order

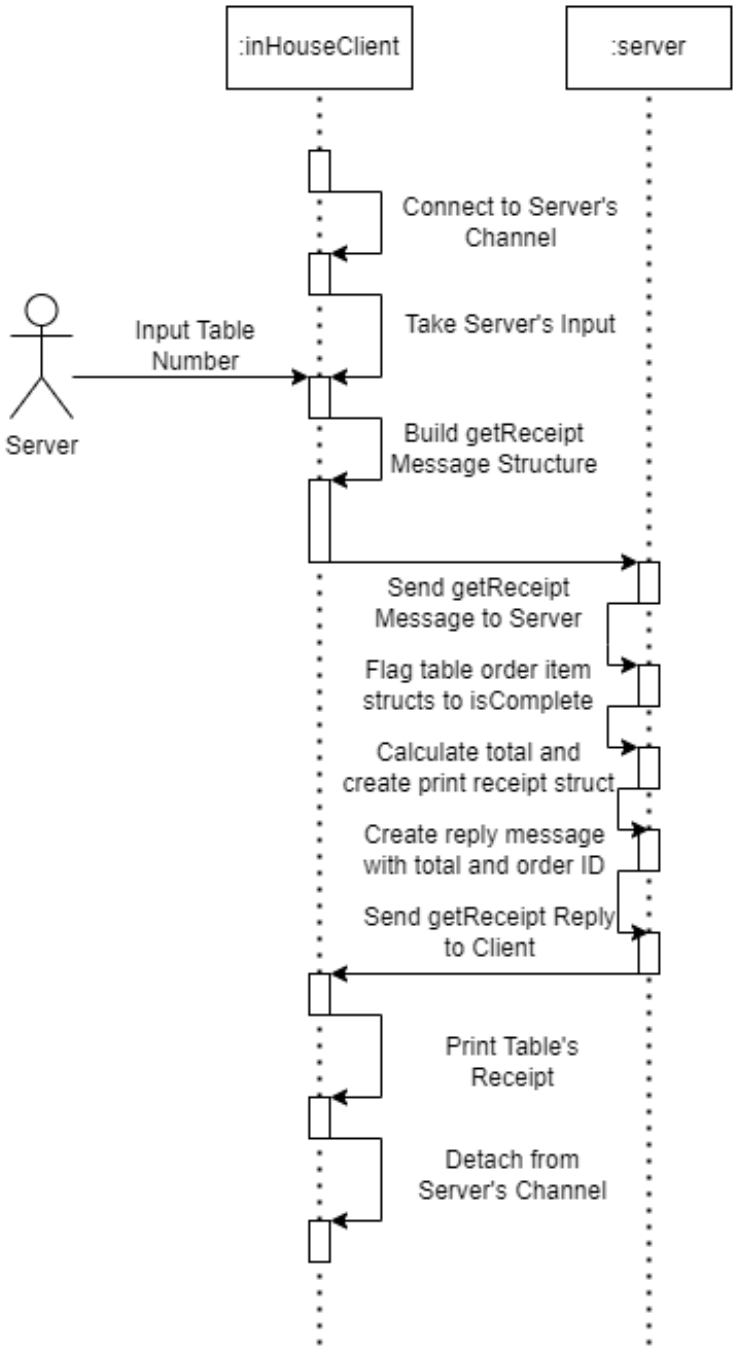


Customers are able to order in-house by talking to their server. The server will then input their order (menu items) along with their table number through the inHouseClient. The inHouseClient will build the message structure using the information that has been provided, and send the message to the server.

The server will check if the requested menu items exist on the restaurant's menu. Assuming all menu items are viable, the server creates a table order item struct that contains the menu item name, a ‘count’ variable for indicating duplicate menu items, the table number, and an ‘isComplete’ flag that is used to determine whether the customer has paid by printing their receipt. A unique order ID is then sent as a message back to the inHouseClient. If a menu item that is not available has been requested, the server sends -1 as the value back to the InHouseClient.

Next, the server will reply back to the client with the order ID. After that, the inHouseClient will detach from the server’s channel, and attach to the kitchen server’s channel to communicate with it instead. While connected, it will send the order to the kitchen server, so the kitchen will have access to the new order, and can start cooking. The kitchen server will send a response just to make sure the message communication is complete. Finally, the inHouseClient will print the order ID before it closes.

F. In-House Receipt



When the customers are done with their food, they can ask for their receipt. The server will be able to input the table number to inHouseClient, and it will send the message to the server.

Once the server receives the message, the server will search through the array of table order item structs that were created during the in-house order process, using the table number passed as a message by the InHouseClient. If no table orders that are incomplete exist, the server will reply to the client with a value of -1. If table order items for the specified table are found, the server will properly calculate the total of the in-house order, mark the 'isComplete' flag to true for each table order item struct and update the 'isReserved' flag for the particular table to false (since the customer is now leaving after paying).

Finally, the server will reply back to the client with the order number, and the order total so that the server will be able to transport the receipt to the customers.

Multi-Server Design

As one may easily notice, our application contains two servers: the main server and the kitchen server. The main server (just called "server") functions as a typical server that handles most of the main functionalities and communication with the clients. The kitchen server on the other hand, does not have any significant functionalities other than receiving orders from the clients. So, it needs to be recognized that the kitchen 'server' is just called a server because it receives messages from the client, but does not function as the typical server that we learned in this course.

6. Conclusion

The Restaurant Manage System Application seeks to create a solution for restaurant management for casual dining. In this mission, we wanted to create a pair of programs which would allow the business owner to create a digital profile of their restaurant, including their menu and tables, as well as allow clients or servers to make requests to a local server with table or online orders as well reservations. All the while, we wanted reliable message passing and speed which meant using QNX's suite of frameworks and the underlying real-time operating system of QNX Neutrino. Another key aspect was the use of the QNX database in order to store the digital representation of the restaurant and its past orders and reservations which would persist between running the client or server programs. While our team was foreign to the QNX database, and there was turmoil experienced learning to integrate it, the result means a much simpler solution to quickly accessing and storing data. In the end, our team of developers has created a good proof of concept which utilizes the client-server architecture and message passing to allow an easier and fast solution to handling incoming orders and reservations for a restaurant business.