Reservation Server

Design

digisoln.com

Table of Contents

Overview

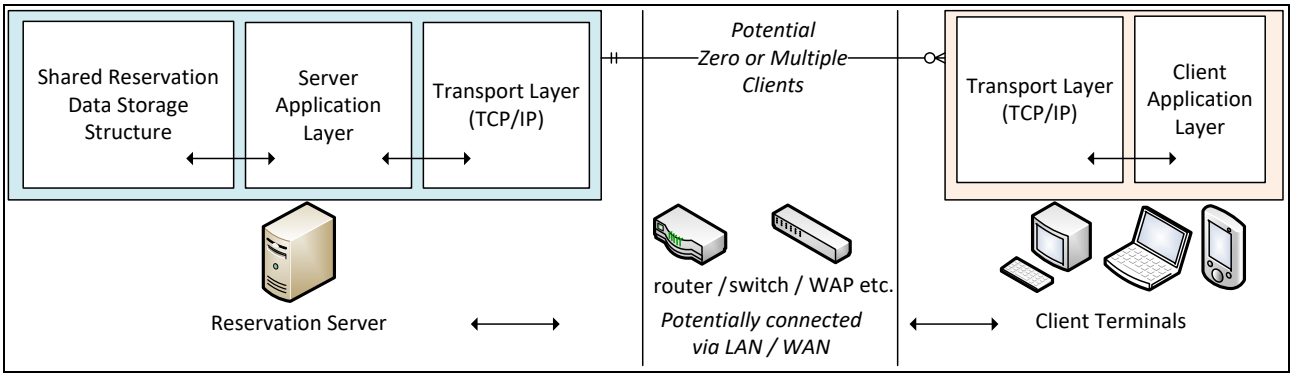
A reservation system is required for a restaurant for Christmas lunch that multiple customers can concurrently interact with. This interaction includes making a reservation, requesting seating availability information and cancelling a booking. The system itself must be implemented to ensure this concurrency is achieved, throughout which data integrity must be maintained by avoiding potential race conditions on shared data structures. This report will detail the development of this system through the design stage of development.

Design Documentation

To achieve a fully functioning, robust and efficient reservation system, preliminary investigations into the design of the product must be completed before coding can begin. Thus, the purpose of this design documentation is to develop the concept into a required set of specifications for implementation.

Architecture

To meet the required needs of the product, it is necessary to implement a solution that allows for multiple concurrent client connections over a local or wide area network. Using a top-down methodology, the basic architecture involved to best deliver this reservation system is shown in the following diagram:



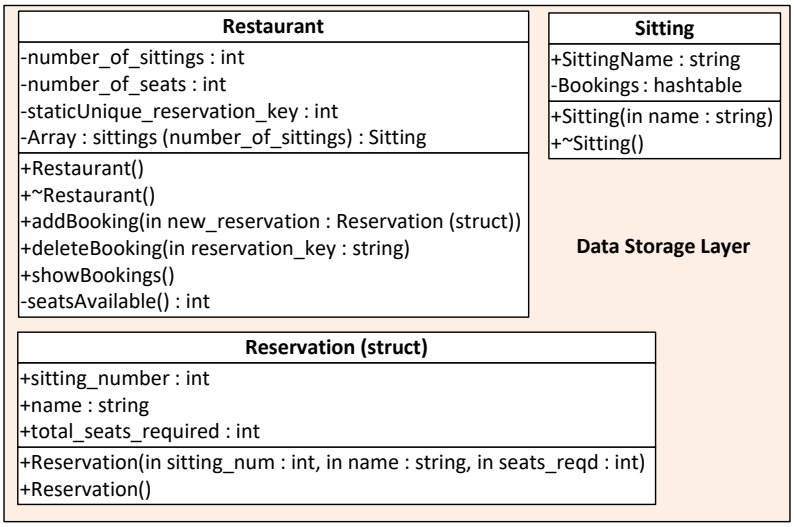
This architecture clearly shows the need for each of the following layers:

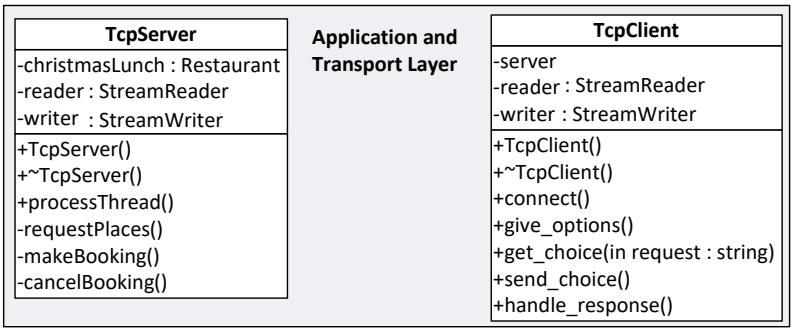
- Reservation Data Storage Structure – multiple transactions running on this structure must execute in *isolation* to ensure data is processed reliably (it is further recommended that these transactions must follow all of the ACID principles for ensuring data integrity);
- Application Layer:
    - Server – to provide and control the methods and processes operating upon the data storage area and within the transportation layer;
    - Client – to seek and respond to information gathered and sent within the transportation layer;
- Transport Layer – to enable connection and reliable communication between client devices and server.

Categorically, the architecture is distributed and multi-tiered, consisting of the presentation layer (user interface) as seen by the client, interacting with the server containing both the logical operations (application layer) and data storage structure. Use of the server application to communicate between client and data is best practise in this instance, as this will reduce the payload required to be sent over the network.

To most efficiently build this system to the requirements above, it is necessary to implement the solution within a C++/CLI environment, utilising the .NET 3.5 framework including the network, sockets and thread libraries this offers to handle the required concurrency and network connectivity.

With this in mind, the following diagram is the planned software architecture of the complete reservation system:



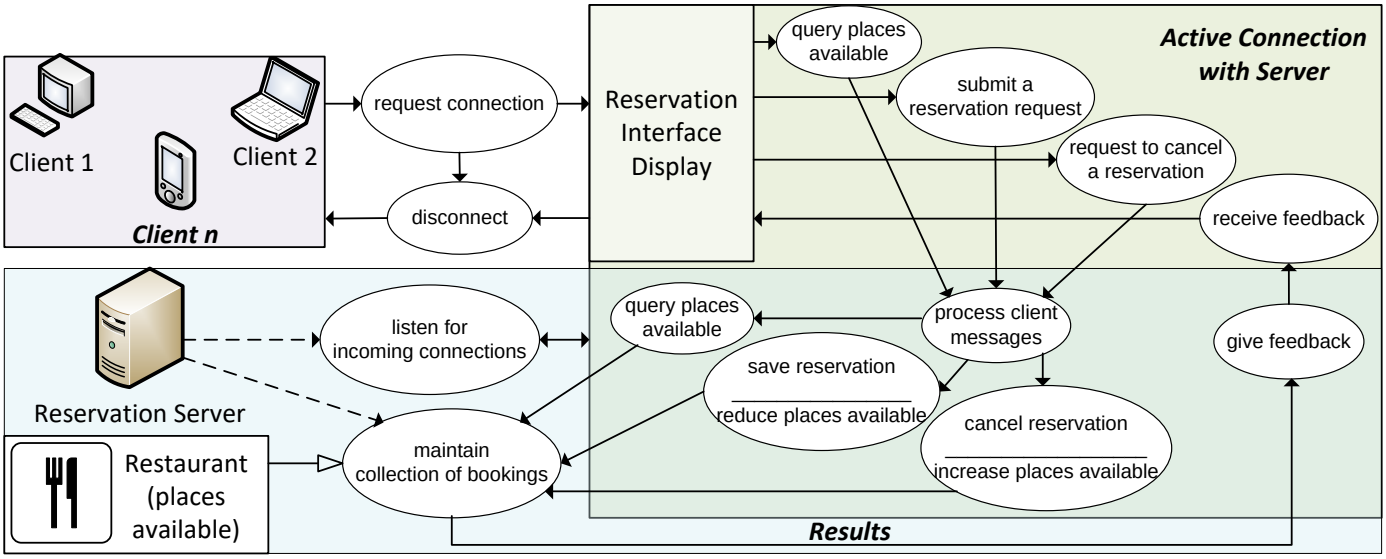


Use Cases

To best understand a typical use case of the reservation system, it is necessary to first identify an exhaustive study of the processes required:

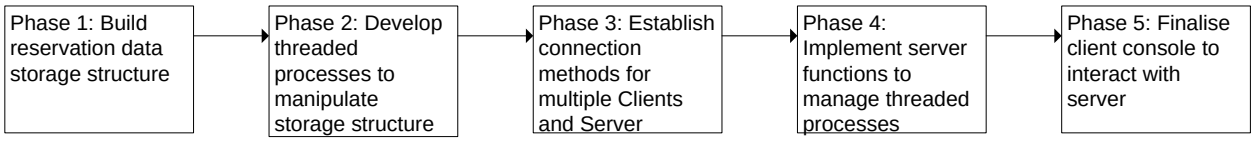
Task	Client 1	Client 2	Client 3
1	Request connection	Request connection	Request connection
2	Query places available	Reserve 10 seats for sitting 2 under the name “Ricky Ponting”	Query places available
3	Reserve 2 seats for sitting 1 under the name of “Cathy Freeman”	Receive a confirmation code	Cancel a reservation for sitting 1 using key “*101IANTHORPE5*”
4	Receive a confirmation code	Cancel the reservation for sitting 2 using key “*22RICKYPONTING10*”	Query places available
5	Disconnect	Disconnect	Disconnect

Based on these required tasks, it is now possible to identify the required functionality of the system as shown:



Development Plan

To successfully implement the solution previously illustrated, the following phases must be adhered to:



Phase 1 and 2 are primarily concerned with implementing the data storage layer shown in the architecture above (Phase 1 the data structures themselves, Phase 2 the methods that interact with these structures). Referring to this same architecture, Phase 3 will focus solely on implementing the transport layer for both client and server. Finally, the application layer for the server will be completed in Phase 4; the client application layer completed in Phase 5.

Although this development plan is illustrated sequentially, it may be necessary during development to revisit earlier phases if functionality requires changes to previously written code. If this is the case, changes should be minimal and fully documented.

Acceptance Testing

The following risks must be tested at each phase to ensure system reliability. These risks must be tested (and mitigated where possible) by the **software developer**:

Phase	Test Condition (indexed by letter)
1	a. Reservation structure can record name and places reserved for each sitting. b. No duplicate reservation KEY(s) will be assigned. c. Methods to query seating, as well as to add or remove bookings are publicly available. d. Critical member variables and functions are restricted with private access. e. Entire data storage layer is free of redundancy and operates efficiently.
2	a. Race conditions do not exist when accessing place availability data structure. b. Concurrency control enables new reservations to take place of cancelled reservations. c. Seating cannot be over allocated. d. Correct modelling of processing times as per software specifications.
3	a. Client tries to connect using an incorrect IP address. b. Client tries to connect using an incorrect Port. c. Either client or server application is non-existent at run-time. d. Either client or server application fails during any communication at any time. e. Entire system can tolerate multiple clients accessing the system concurrently.
4	a. Server behaves as a user would expect, with messages sent correctly interpreted. b. Server manages multiple concurrent but independently executed threads developed in Phase 2. c. Server ensures transactions made are consistent with ACID properties of a reliable data system. d. Feedback is given that is specific, correct and suggestive when necessary.
5	a. Client or server does not end up in deadlock waiting for each other to send information. b. Functionality meets requirements of use case specifications. c. Information is efficiently packaged for sending and handled correctly when receiving.

Future Directions

Assuming the client is in agreement with the above detailed specifications, the reservation system must now be developed and thoroughly tested. Results of the above acceptance testing will be documented in the proceeding Acceptance documentation.

References

Unless otherwise specified, this assignment is entirely my own work, utilising only these references:

Fraser, S. 2006. Pro Visual C++/CLI and the .NET 2.0 Platform. Berkeley, CA.

Jarvis, J. and Jarvis. D. 2009. Application Development using Visual C++ 2005.