State Diagram and Protocol Design

For Project 2, you are asked to design a communication protocol between the client and the server. This guide is designed to provide additional help in your initial design doc.

The main components of the design doc for project 2 consists of a state diagram for the client and the server, and the protocol these two communicate in. We will use a toy example similar to a key-value server to demonstrate this. We will provide the protocol design for a VERY simple key-value store and the client state diagram for it.

Consider a database server that supports four operations. The database table will allow any one to connect to it, and an unlimited number of connection requests (unlike our project). The "$" character is reserved, and can not be part of a table name , key, or value.

- OPEN(tablename)
- CLOSE(tablename)
- GET(key)
- SET(key, value)

Next, we are going to describe the message format for these commands. For the sake of simplicity, we ignored a lot of error conditions. You will be expected to show how you handle errors in your design.

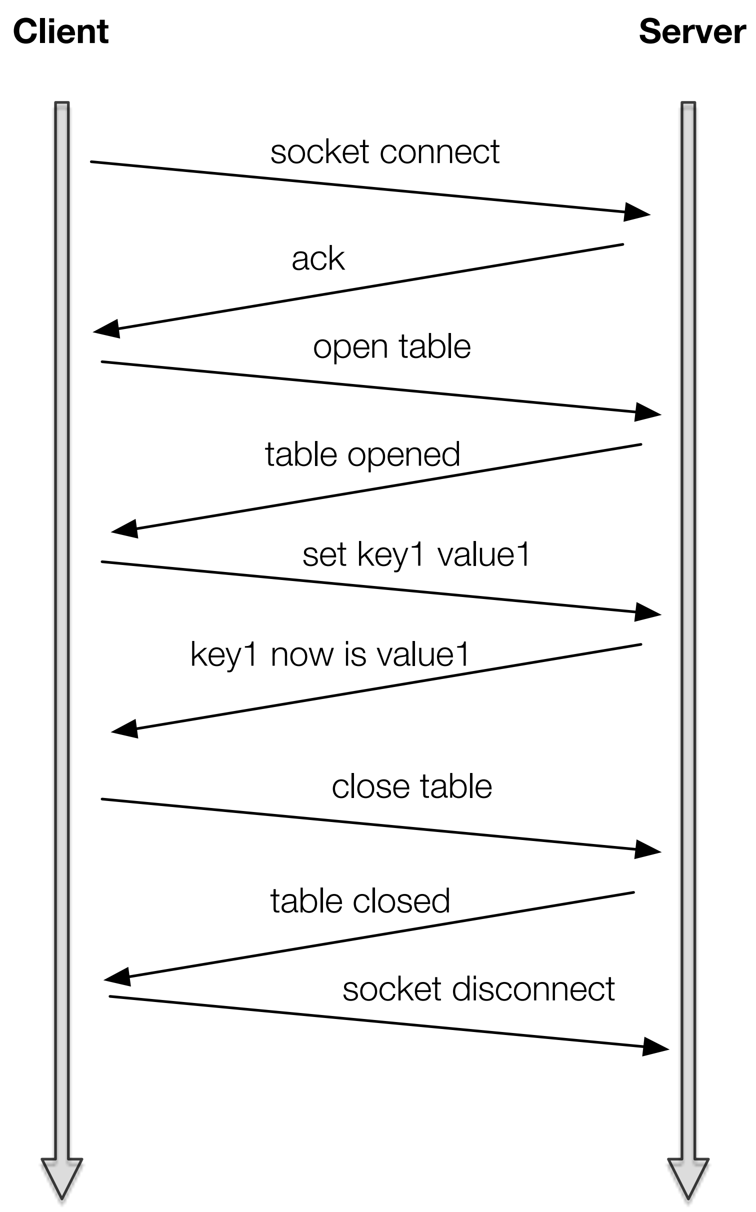
Message Format

The message contains of the following data in the order which they are specified.
int: Command Type (1-4)
int: Number of Parameters
int: Parameter 1 size in bytes
bytes[size]: Parameter 1
int: Parameter 2 size in bytes
bytes[size]: Parameter 2 in bytes
…

Example of a GET("ab") message would be 312ab, assuming GET maps to a command code of 3. Obviously this protocol is not very human readable, but reasonably efficient. In your design, you may choose a protocol format that is more readable, and easier to debug.

Message Sequence Diagram

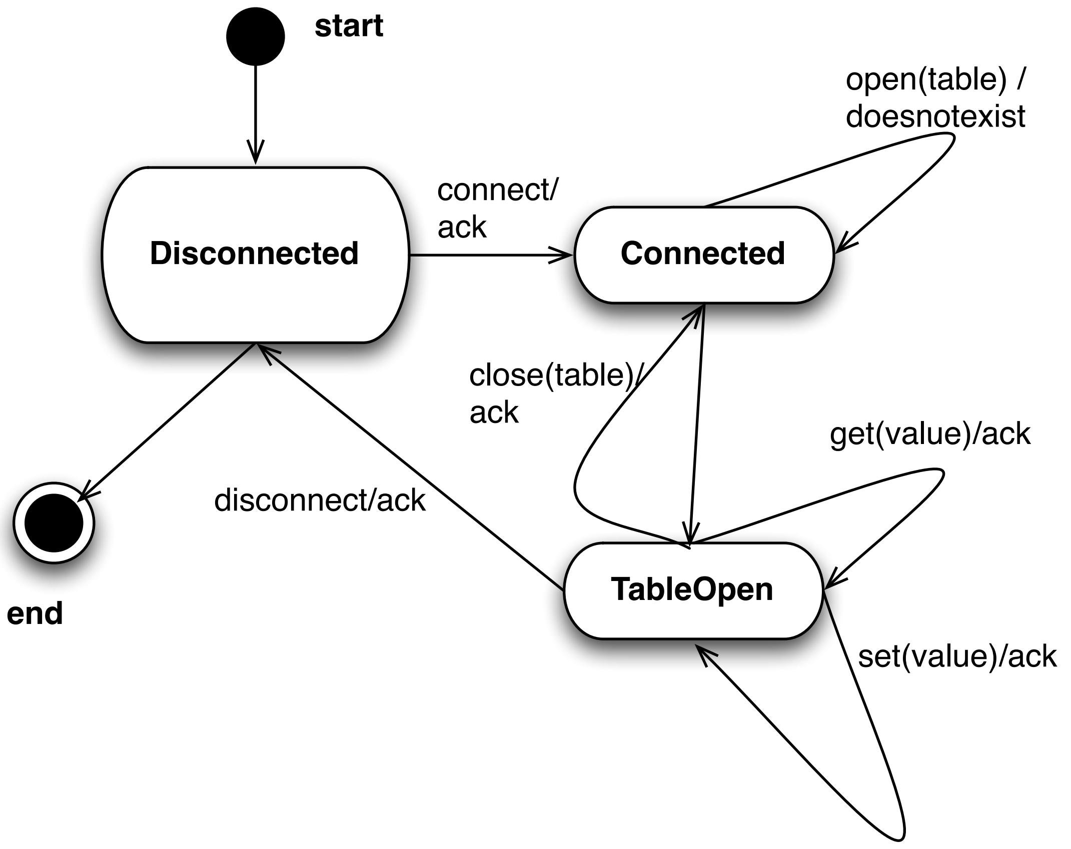
The following is a message sequence diagram showing sequence of events that may happen. This type of diagram is very useful to demonstrate the interaction between the server and the client, especially if there are any asynchronous messages.



State Diagram for the Client and the server

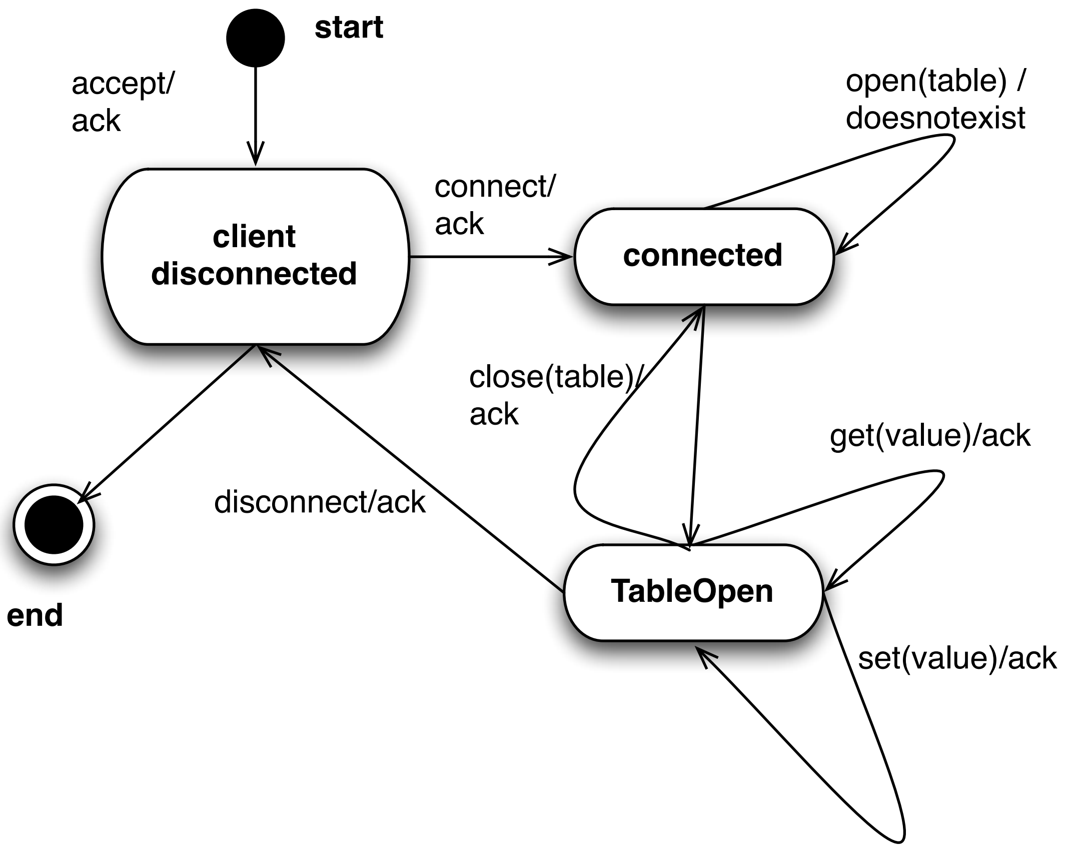
In the following diagram, there are three states. Each edge has a request/reply corresponding to it, which indicates the conditions for the state transition.

Client State Diagram



The server state diagram is fairly similar to this since in many ways, the server is mirroring the client state in one of its thread. However, note that the edges are no longer described by request/reply, since the server in this case does not initiate conversations. In this case, event/action pairs describe the edges. Not all events are initiated by the clients, as they are in this case. For example, local timer can trigger time out that would not be found in the client state diagram, but is in the server state diagram.

Server State Diagram


start
accept/ ack
client disconnected
connect/ ack
connected
open(table) / doesnotexist
close(table)/ ack
get(value)/ack
disconnect/ack
TableOpen
set(value)/ack
end