****
*
558 Distributed Yaqun Yu and Xiao Liang's Execute Summary for Project 2
****
*

****
Assignment Overview:
****
For this assignment we are asking to enable client and server to communicate using Remote Procedure Calls (RPC) instead of socket, which is much easier for communication. Remote Procedure Call (RPC) is a protocol that one program can use to request a service from a program located in another computer in a network without having to understand network details. (A procedure call is also sometimes known as a function call or a subroutine call.) RPC uses the client/server model. We were using java for project 1, so for this one we used java RMI for RPC communication. The Java Remote Method Invocation (Java RMI) is a Java API that performs the object-oriented equivalent of remote procedure calls (RPC), with support for direct transfer of serialized Java classes and distributed garbage collection. The RPC function can be called concurrently, consequently and can handle mutual exclusion. In the read-in test set, our client can do at least five of each operation: PUT, GET and DELET.

****
Technical Impression:
****
During the implementation, the first technique is to understand RPC and java RMI. RMI API comprises of two programs, a server and a client, a classic server creates remote objects, makes references to these objects accessible and wait for clients to invoke methods on these objects. The RMI distributed application uses the RMI Registry to obtain a reference to a remote object. The server calls the registry to associate a name with a remote object. The client looks up the remote object by its name in the server's registry and then invokes a method on it.
So I first create an interface, define all the functions that will be used by client. Each function throws RemoteException as requested. Then implement this interface in RMIserver.java, along with the detail definition for each function. In RMIclient, each function is implemented as a case. At first, implementation was using manual input. Then inspired by the file system that we used in last project, we changed it to read lines via txt test file. We find that the most difficult part was to understand the protocol working mechanism and how the API helps. At first we read through plant of materials just to get a better understanding for this protocol. Once we understand how that work, the implement structure become clear. During the implementation we run into some minor bugs like mismatch between

RMIclient object type and interface function return type, and switch our client structure from manual input to read file from local system etc. Gladly, we cleared all that up and finished our implementation finally.