CS 5523 Operating Systems Fall '15

Project 3: MyBooks.com, A console based Distributed Client Server program based on RMI Communication

A Project Report by Venkat Rahul Dantuluri (fpj626/@01518869)

1. This is to report the successful completion of Project 3. As part of the project I have implemented traditional Client Server model of communication. This communication is unique as it supports distributed communication. Server can be on one machine and client can be on the other. The notion of RMI or remote Method Invocation is that, clients will initiate a procedure call. But this procedure is run on the server machine and the results are communicated back to the client by the server. Since these results travel across a network to the client, they are, most of the time, Serializable. The Server initially registers a Service in a local Registry. It associates a name against this service which the client uses to identify and utilize the service. Please read the README.txt to understand how to execute this program. All server related files are packaged into a SERVER folder. So we did the same for CLIENT. There are Makefiles to compile the JAVA source files and there are shell-scripts to execute the compiled bytecode. There are 3 front end servers and one backend data server.

   Note: If server and client are run on different machines Then we need to communicate the server host to the client while client execution. The host and port specification is present in ServerService.java file. In case we need to test on different machine please edit the file. Please follow instructions in README.txt that is provided along with

2. In this application the front end servers will synchronously communicate with the backend data server to update the list of books. In my application all the front end servers will ping the data server once every 2 seconds.

3. Apart from synchronously pinging the backend server, the front end servers FE SERVER2 and FE SERVER3 will constantly keep pinging the front end server FE SERVER1 with their system times. The FE SERVER1 will act as the leader and will always evaluate the correction in time for each of the front end server. This correction of time is added to their own system time while the front end server calls a service remotely pertaining to the backend data server. The backend server then uses this information to synchronize the requests among the front end servers. The leader front end Server FE SERVER1 uses Berkeley's algorithm to compute the averages of the clock cycle time and compute the corrections. The clocks being synced up can be viewed in the logs.

4. Each service has corresponding counter and a variable that stores the execution times of each service in nanoseconds. I measure the performance by finding the average execution times of the service over the number of times it has been called. You can see the performance of a service by calling one of the reporting methods of the server.

5. The initial problem that I faced is how to design the datasource so as to allow multiple threads a common access and at the same time protect the data consistencies. This lead me to create BookCatalog. Most of the data in BookCatalog is static hence it is stored in the data segment of the address space. Hence all the threads can access it. But since the methods that manipulate this static data are synchronized we won't have a problem of multiple concurrent threads making the data inconsistent.

   Another problem that I faced is the environmental differences between code developed in an IDE and executed in command line. I had to remove packages to ensure each class to smoothly locate dependent classes.

   Another difficulty I faced is during the execution of Client and Server on different machines, each on a different network. Here we were forced to switch off the Host based intrusion detection systems like firewall. Even that did not resolve our problem. Many Network based intrusion detection systems would have blocked our connection. We could only get the client and server to work properly in an enclosed network like UTSA elk machines.

   Testing the clock synchronization is very tiresome.

   Lastly I faced this problems on linux machine in which my RMI server would still grab hold of the port despite killing the server program. This is because, when we start the server, the JRE will start a RMI service that listens to port of the Registry. This has to manually be stopped and hence I created the "stopServer.sh". Run this script to shut the RMI service down too.

6. For test case details please refer to the PDF inside TEST folder.