15-640 Distributed Systems Lab2 - RMI Facility in Java

-Amey Ghadigaonkar

6/24/2014

15 - 640 Distributed Systems

Lab 2 – RMI Facility

-Amey Ghadigaonkar.

Index:

1. IMPORTANT INSTRUCTIONS:

(Imp: windows uses "." While unix uses "/". This manual is created for AFS i.e. unix. For example "bin.client.Client" vs. "bin/client/Client")

Go to "/cmu.ds.lab2/bin/" which contains compiled files. The source files are inside "/cmu.ds.lab2/src/". You can compile the src files one folder at a time if you want to. This project can also be run on Eclipse by downloading, unzipping, creating a java project with these files as source files.

IMPORTANT: Suppose the name of the class file is Calci.java, then the name of the interface should be CalciInterface.java and stub will eventually be named CalciInterface_Stub.class by the stub compiler

I have provided a skimmed version of the client on **"/cmu.ds.lab2.client/bin/"**

This version has same source files but server and registry class files have deleted. You may use this for client program deployment or just use the complete lab2 compilation. All instructions for both cases are same. Just navigate into these respective folders for server/registry and client.

The project has been deployed several times on unix.andrew.cmu.edu. It works. However, sometimes, it might throw a **"Connection refused" exception**. This is especially true for client.Client class. I don't know the reason for this strange behavior but it can be solved by:

1. Closing current terminal which throws this exception
2. Opening a fresh session in a new terminal window

2. HOW TO RUN MY PROGRAM:

Go to "/cmu.ds.lab2/bin/". It has precompiled class files. Another folder (/cmu.ds.lab2/src/) has all the source files.

i. **Start registry server**

```
java registry.RegistryServer
```

ii. **Start server**

```
java server.Server <registryIP> <registryPort>
```

If not given registryIP and registryPort, it starts with default values

iii. **Run test case 1 (data file is bin/test1/data.txt)**

Create object: At server, create an object of test1 and name it. (Select option 2 on server)

```
Enter Class Name (example1.Calci OR test1.ZipCodeServerImpl OR test2.ZipCodeRListImpl OR test3.NameServerImpl):
>>> test1.ZipCodeServerImp
Enter new bindname:
>>> Zip1
```

Now object of type test1.ZipCodeServerImp is created with bindname "Zip1". We use this object for RMI.

Create a new client program to run test case. Enter in command line:

```
java test1/ZipCodeClient <registry_IP> <registry_port> <object_name> <outputfile>
```

For example, write:

```
java test1/ZipCodeClient 128.2.13.161 1099 Zip1 test1/data.txt
```

iv. **Run test case 2 (data file is bin/test1/data.txt)**

Create object: At server, create an object of test1 and name it. (Select option 2 on server)

```
Enter Class Name (example1.Calci OR test1.ZipCodeServerImpl OR test2.ZipCodeRListImpl OR test3.NameServerImpl):
>>> test2.ZipCodeRListImpl
Enter new bindname:
>>> Zip2
```

Now object of type test2.ZipCodeRListImpl is created with bindname "Zip2". We use this object for RMI.

Create a new client program to run test case. Enter in command line:

```
java test2/ZipCodeRListClient <registry_IP> <registry_port> <object_name> <outputfile>
```

For example, write:

java test2/ZipCodeRListClient 128.2.13.161 1099 Zip2 test1/data.txt

v. Run test case 3 (data file is bin/test1/data.txt)

Create object: At server, create an object of test1 and name it. (Select option 2 on server)

Enter Class Name (example1.Calci OR test1.ZipCodeServerImpl OR test2.ZipCodeRListImpl OR test3.NameServerImpl):

>>> test3.NameServerImpl

Enter new bindname:

>>> Zip3

Now object of type test3.NameServerImpl is created with bindname "Zip3". We use this object for RMI.

Create a new client program to run test case. Enter in command line:

java test3/NameServerClient <registry_IP> <registry_port> <object_name> <outputfile>

For example, write:

java test3.NameServerClient 128.2.13.161 1099 Zip3 test1/data.txt

vi. Run simple Calculator example

Start client at command line interface:

java client.Client <registryIP> <registryPort>

For example,

java client/Client 128.2.13.161 1099

Now run option 3 on server. Then run this option multiple times on client. You will see how the state variable for the calculator can be incremented by 5 each time and also be used for stateless addition.

Note: "__dummy__" variables are created by test1, test2 and test3 whereas Calculator example uses Calci1, Calci2, Calci3 and Calci4.

If not given registryIP and registryPort, it starts with default values (localhost, 1099).

I have provided sample sample test cases (option 3 on the both, server and client) to create sample calculator objects and register them into the registry so that they can be looked up conveniently.

3. My RMI Design:

I designed my RMI framework with an aim to simplify the implementation as much as possible for the Application Programmer. Thus, I don't let the user mess with the received RoRs. I abstract away this handling like the actual RMI does by encapsulating this RoR inside a "Stub" object.

However, I tweaked the framework as described later in the report so that the user can also explicitly pass RoRs as parameters if he gets his hands on a RoR (I have a method getRoR() that returns the RoR inside the stub if the user wants to do this).

My framework can:

A. Ability to lookup existing remote objects. The bindnames are assigned at the server and the client does not have the power to name/rename remote objects.
B. Invoke methods on remote objects and pass/return remote/local objects.
C. A stub compiler to create simple stubs that are compatible with my framework

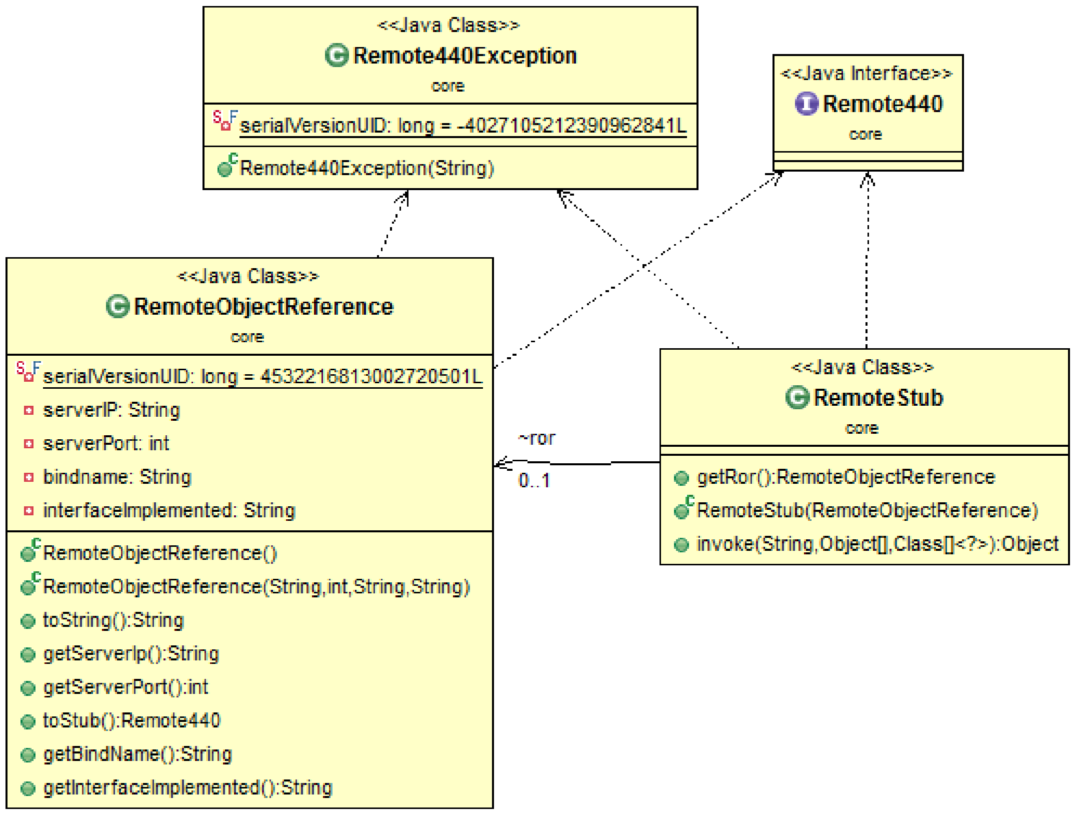
Things I did not have time to implement:

A. Transferring ".class" files. However, I had a simple mechanism in mind to transfer the ".class" file using sockets (the server will have a listening port to transfer these class files).
B. Distributed garbage collector. However, I read about this on the official Java docs (where I read about all the other stuff, too). I have a rough idea about it and would have loved to do it. But, shortage of time.

Important Design Decisions and tradeoffs:

I implemented this framework by referring primarily to the existing Java RMI framework. Like it, I also pass the Remote Object References implicitly without the knowledge of the Application Programmer. It is achieved by using the concept of "stub" as described below in point 7. The major decisions I took while developing the framework have been specified below:

1. Having a generic Remote440 interface:
   This Remote440 interface is analogous to java.rmi.Remote class. However, my Remote440 class is just a wrapper for the objects to be handled by my framework. It does not specify any business logic or any methods.
2. Having a generic RemoteStub class:
   This class is the pseudo object I will store at the client. This contains a generic method "invoke" which handles all the communication with the server and marshals and unmarshals arguments and gets return values. This stub is needed to create a generic stub which is not concerned with implementation of the method. It only concerns itself with abstraction of the implementation from the application programmer.

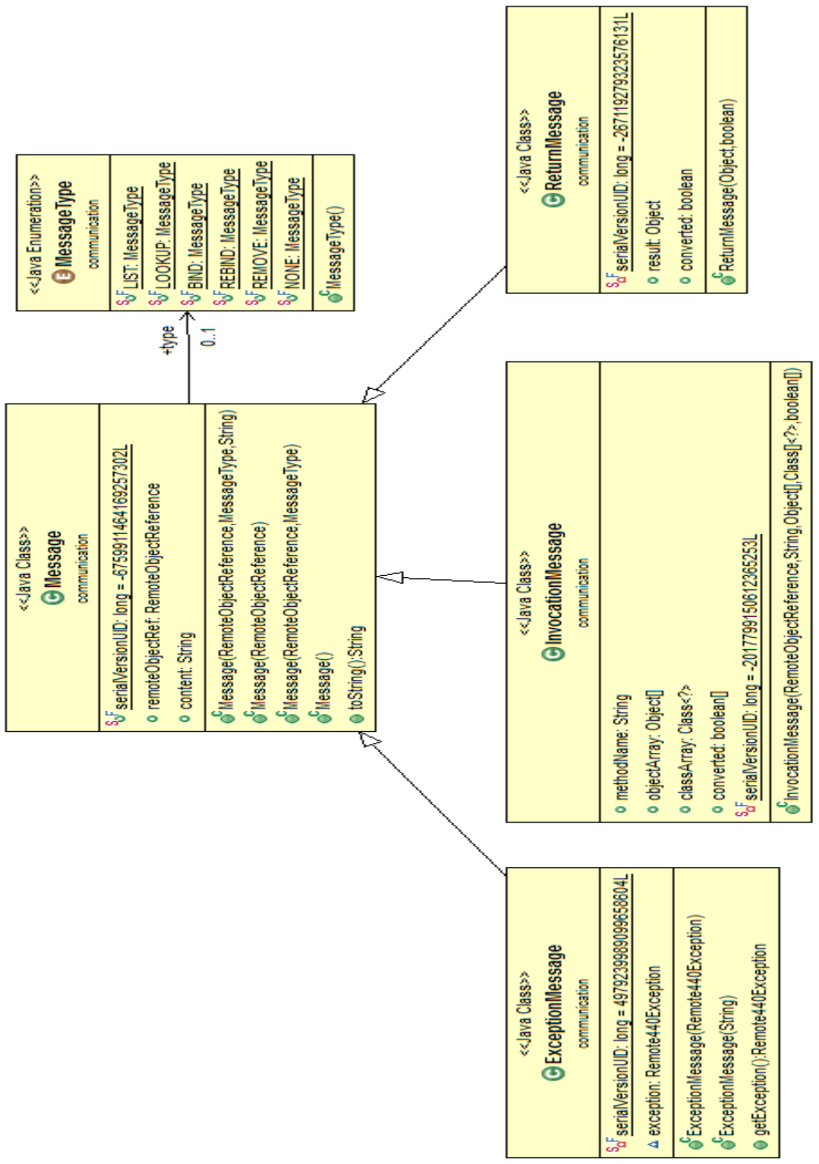


<<Java Class>>
Remote440Exception
core
serialVersionUID: long = -4027105212390962841L
Remote440Exception(String)
<<Java Interface>>
Remote440
core
<<Java Class>>
RemoteObjectReference
core
serialVersionUID: long = 4532216813002720501L
serverIP: String
serverPort: int
bindname: String
interfaceImplemented: String
RemoteObjectReference()
RemoteObjectReference(String,int,String,String)
toString():String
getServerIp():String
getServerPort():int
toStub():Remote440
getBindName():String
getInterfaceImplemented():String
~ror
0..1
<<Java Class>>
RemoteStub
core
getRor():RemoteObjectReference
RemoteStub(RemoteObjectReference)
invoke(String,Object[],Class[]<?>):Object

Figure 1 Remote440 class and its children

3. Having a generic Remote440Exception: This is a common exception that the server throws in order to hide the things that went wrong at the server or during communication. In my view, these errors are implementation level details that the user does not need to be aware of.
4. Having a standard Remote Object Reference: The RoR is an important part of the implementation that needs the required information to identify and communicate with the server which is holding the remote object. These RoRs are generated only at the server and thus can be used at only the server since the information required to use them is accessible only there.
5. Having a common Message class: One of the main lessons from the last assignment was that the network communication needs to be simplified to a point that I know exactly what is being transferred over the network at any point in time and that the contents can be extracted without much thinking.
6. Inheriting from the super Message class: Once I had a standard Message class, I had difficulty holding content for all types of messages. Thus, I inherited from the main Message class and the inherited classes have the specialized information. E.g. the ExceptionMessage holds the exception that occurred and the ReturnMessage holds the returned object. We have 3 kinds of messages:
    a. InvocationMessage: invoking a remote method
    b. ReturnMessage: returning results from server

c. ExceptionMessage: returning exceptions (I encapsulate them into Remote440Exception later)



<<Java Enumeration>>
MessageType
communication
LIST: MessageType
LOOKUP: MessageType
BIND: MessageType
REBIND: MessageType
REMOVE: MessageType
NONE: MessageType
MessageType()
+type
0..1
<<Java Class>>
Message
communication
serialVersionUID: long = -6759911464169257302L
remoteObjectRef: RemoteObjectReference
content: String
Message(RemoteObjectReference,MessageType,String)
Message(RemoteObjectReference)
Message(RemoteObjectReference,MessageType)
Message()
toString():String
<<Java Class>>
ReturnMessage
communication
serialVersionUID: long = -267119279323576131L
result: Object
converted: boolean
ReturnMessage(Object,boolean)
<<Java Class>>
InvocationMessage
communication
methodName: String
objectArray: Object[]
classArray: Class<?>
converted: boolean[]
serialVersionUID: long = -2017799150612365253L
InvocationMessage(RemoteObjectReference,String,Object[],Class[]<?>,boolean[])
<<Java Class>>
ExceptionMessage
communication
serialVersionUID: long = 4979239989099653604L
exception: Remote440Exception
ExceptionMessage(Remote440Exception)
ExceptionMessage(String)
getException():Remote440Exception

Figure 2: Message SuperClass and its Children

The figure above shows how different messages are derived from a single Message class.

I was also planning on a **MissingClassMessage** to hold the actual class file that needs to be transferred over the network when we require a missing ".class" file in RMI. This is a do-able task but I am limited by time.

7. Passing RemoteObjectReferences as parameters: When I was doing a final overview of my project, I realized that my framework was passing RoRs implicitly. But later, on looking at the

write-up, and referring the ZIPCodeServer example, I realized that RoRs also need to be sent explicitly. The problem with that would be that the explicitly passed parameters would be converted to local objects at the server. This was a caveat that I had missed. I corrected it by storing Boolean values that signify whether the RoR needs to be converted into a Local Object at the server. The same for return message for the reverse process.

4. Features Implemented

RMI Framework

My RMI framework implements managing local references, remote references, explicit RoRs, state variables, etc. However, in order to use the framework, a class must inherit my Remote440 class. This is used to refer to all objects inside the framework.

Implicit RoR Handling (core/RemoteStub)

I don’t let the application programmer worry about handling Remote Object References (RoRs). These remote references are converted to local objects and marshaled/unmarshalled by my stub and proxy dispatcher. Thus, RoRs are not exposed unless the programmer wants to expose them. Everything is handled automatically.

Server Proxy Dispatcher (server/ServerProcessor)

This module handles (un)marshalling and conversion to local object at the server. This is similar to a server-side skeleton but it much leaner due to generics.

Extensible Message Format (communication/Message)

Messages have been standardized to extend from this class to allow for smooth communication. Specialized messages can be developed by extending this class and including more fields. I provide a String and RoR field in the default Message class. I use specializations for inherited class e.g. ReturnMessage also has a returned Object field. This also facilitates remote exception handling using a consistent interface.

Stub Compiler (client/StubCompiler)

The stub compiler takes an interface and outputs a stub which works in my framework. It has not been tested thoroughly due to lack of time but can convert simple interfaces to their respective stubs.

The stub compiler should be run on the interface of the concerned class file.

Eg. Java client/StubCompiler ExampleInterface

This will give a new file ExampleInterface_Stub that needs to be compiled into a class file.

IMPORTANT: As mentioned earlier, if the name of the class file is Calci.java, then the name of the interface should be CalciInterface.java and stub will eventually be named CalciInterface_Stub.class.

5. Things not implemented due to shortage of time:

1. Robust StubCompiler capable of handling bounded type parameters

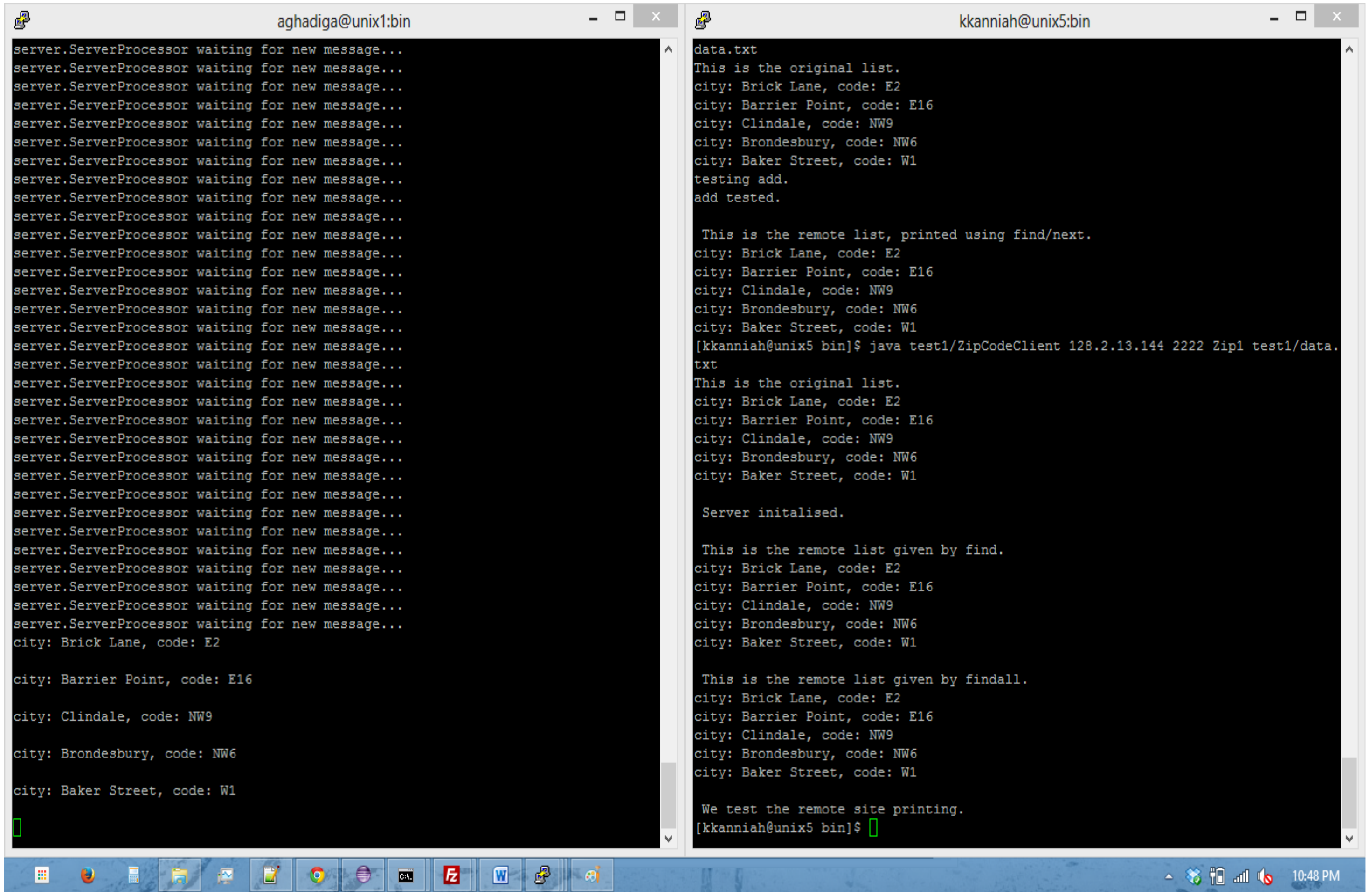
   e.g. public <U extends Number> void inspect(U u)

   refer: http://docs.oracle.com/javase/tutorial/java/generics/bounded.html

2. HTTP transfer of class files
   I had a plan to transfer class files directly using sockets without using HTTP. But, time was insufficient.

3. Distributed Garbage Collector
   I read up material on it but did not get time to implement it.

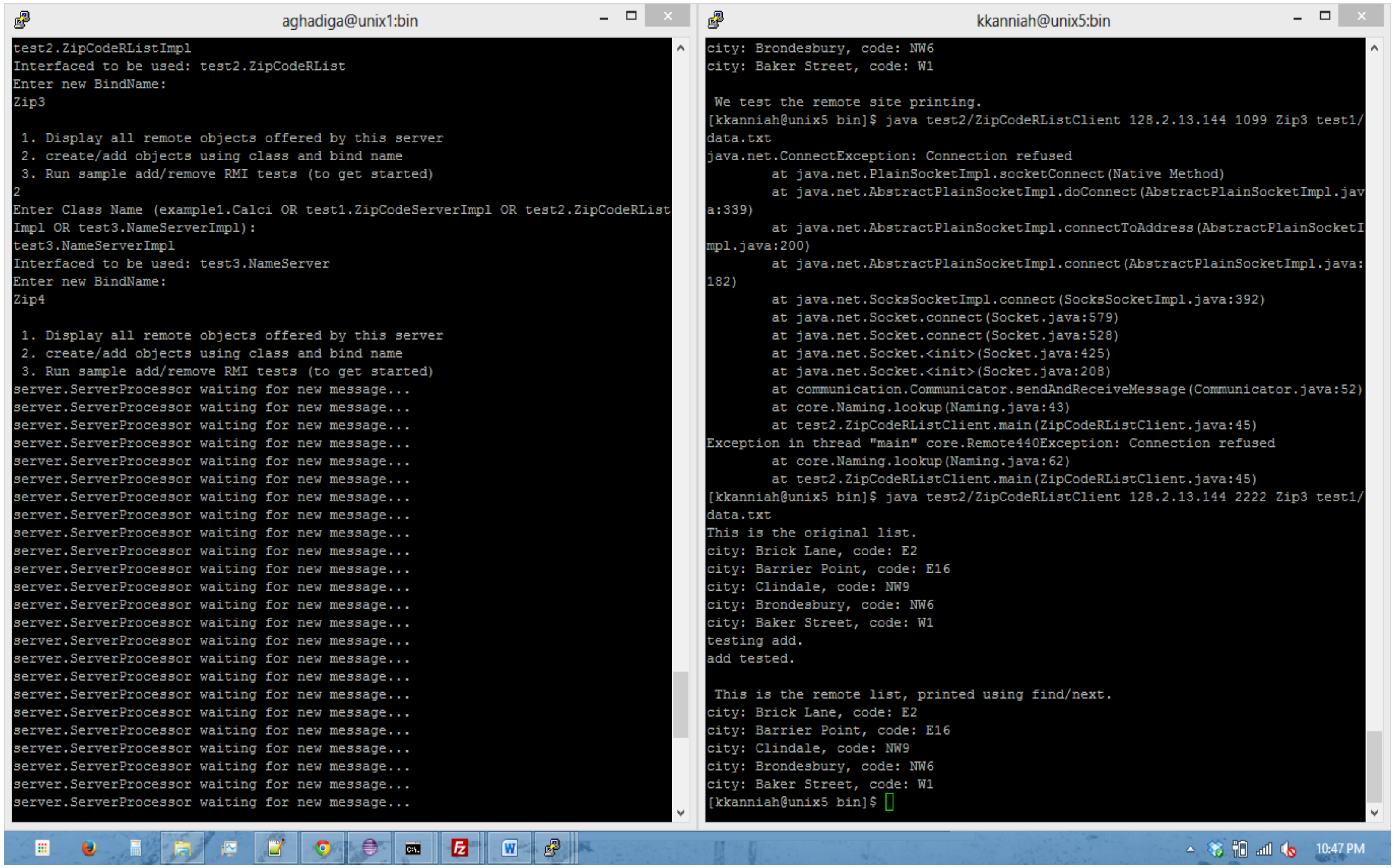
6. Screenshots of programs running on AFS with distributed user logins



```
server.ServerProcessor waiting for new message...
server.ServerProcessor waiting for new message...
server.ServerProcessor waiting for new message...
server.ServerProcessor waiting for new message...
server.ServerProcessor waiting for new message...
server.ServerProcessor waiting for new message...
server.ServerProcessor waiting for new message...
server.ServerProcessor waiting for new message...
server.ServerProcessor waiting for new message...
server.ServerProcessor waiting for new message...
server.ServerProcessor waiting for new message...
server.ServerProcessor waiting for new message...
server.ServerProcessor waiting for new message...
server.ServerProcessor waiting for new message...
server.ServerProcessor waiting for new message...
server.ServerProcessor waiting for new message...
server.ServerProcessor waiting for new message...
server.ServerProcessor waiting for new message...
server.ServerProcessor waiting for new message...
server.ServerProcessor waiting for new message...
server.ServerProcessor waiting for new message...
server.ServerProcessor waiting for new message...
server.ServerProcessor waiting for new message...
server.ServerProcessor waiting for new message...
server.ServerProcessor waiting for new message...
server.ServerProcessor waiting for new message...
server.ServerProcessor waiting for new message...
server.ServerProcessor waiting for new message...
server.ServerProcessor waiting for new message...
server.ServerProcessor waiting for new message...
server.ServerProcessor waiting for new message...
server.ServerProcessor waiting for new message...
city: Brick Lane, code: E2

city: Barrier Point, code: E16

city: Clindale, code: NW9

city: Brondesbury, code: NW6

city: Baker Street, code: W1

```

```
data.txt
This is the original list.
city: Brick Lane, code: E2
city: Barrier Point, code: E16
city: Clindale, code: NW9
city: Brondesbury, code: NW6
city: Baker Street, code: W1
testing add.
add tested.

 This is the remote list, printed using find/next.
city: Brick Lane, code: E2
city: Barrier Point, code: E16
city: Clindale, code: NW9
city: Brondesbury, code: NW6
city: Baker Street, code: W1
[kkanniah@unix5 bin]$ java test1/ZipCodeClient 128.2.13.144 2222 Zip1 test1/data.
txt
This is the original list.
city: Brick Lane, code: E2
city: Barrier Point, code: E16
city: Clindale, code: NW9
city: Brondesbury, code: NW6
city: Baker Street, code: W1

 Server initalised.

 This is the remote list given by find.
city: Brick Lane, code: E2
city: Barrier Point, code: E16
city: Clindale, code: NW9
city: Brondesbury, code: NW6
city: Baker Street, code: W1

 This is the remote list given by findall.
city: Brick Lane, code: E2
city: Barrier Point, code: E16
city: Clindale, code: NW9
city: Brondesbury, code: NW6
city: Baker Street, code: W1

 We test the remote site printing.
[kkanniah@unix5 bin]$
```

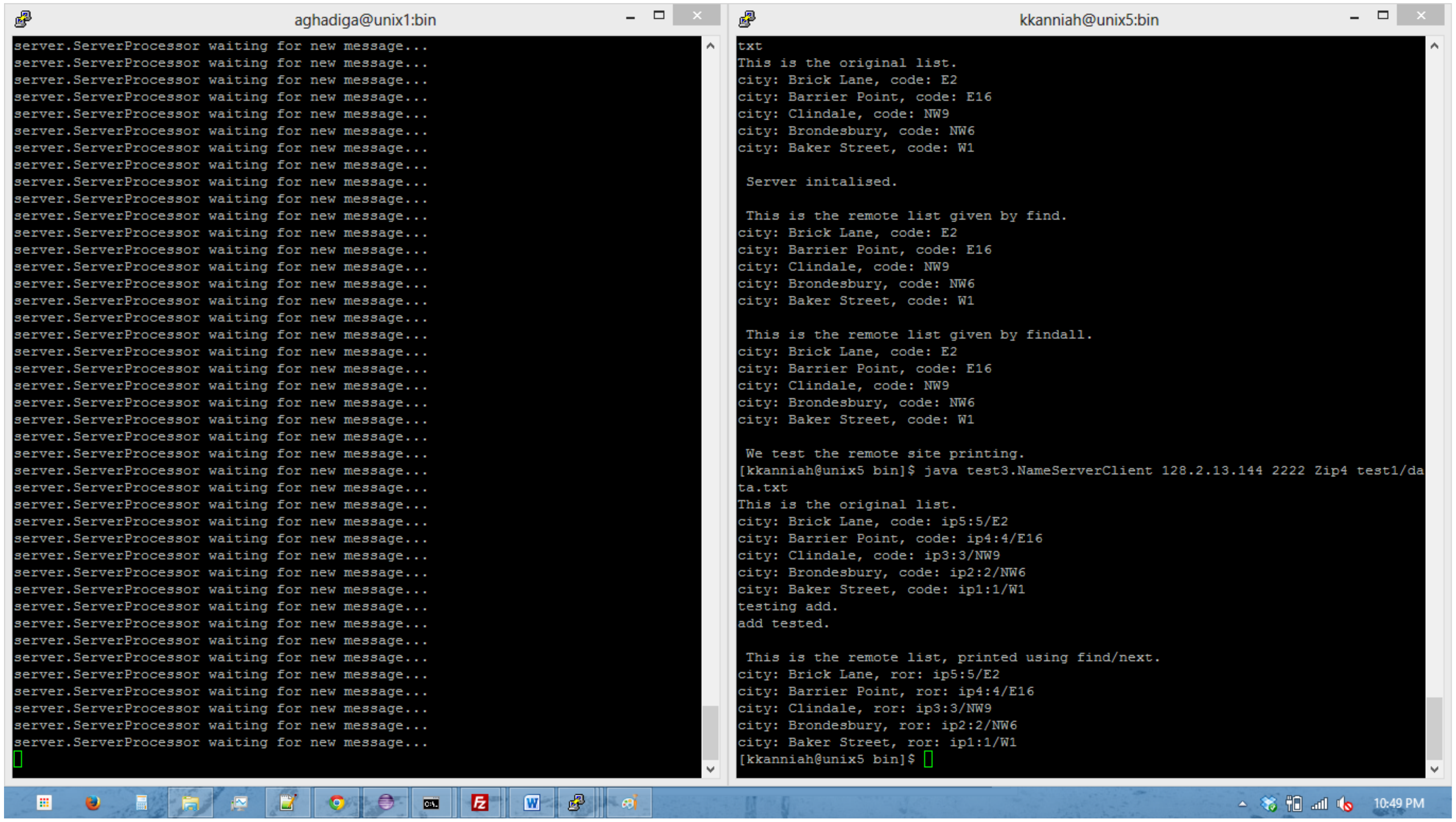
Figure 3 Test 1 Running on AFS



```
test2.ZipCodeRListImpl
Interfaced to be used: test2.ZipCodeRList
Enter new BindName:
Zip3

 1. Display all remote objects offered by this server
 2. create/add objects using class and bind name
 3. Run sample add/remove RMI tests (to get started)
2
Enter Class Name (example1.Calci OR test1.ZipCodeServerImpl OR test2.ZipCodeRList
Impl OR test3.NameServerImpl):
test3.NameServerImpl
Interfaced to be used: test3.NameServer
Enter new BindName:
Zip4

 1. Display all remote objects offered by this server
 2. create/add objects using class and bind name
 3. Run sample add/remove RMI tests (to get started)
server.ServerProcessor waiting for new message...
server.ServerProcessor waiting for new message...
server.ServerProcessor waiting for new message...
server.ServerProcessor waiting for new message...
server.ServerProcessor waiting for new message...
server.ServerProcessor waiting for new message...
server.ServerProcessor waiting for new message...
server.ServerProcessor waiting for new message...
server.ServerProcessor waiting for new message...
server.ServerProcessor waiting for new message...
server.ServerProcessor waiting for new message...
server.ServerProcessor waiting for new message...
server.ServerProcessor waiting for new message...
server.ServerProcessor waiting for new message...
server.ServerProcessor waiting for new message...
server.ServerProcessor waiting for new message...
server.ServerProcessor waiting for new message...
server.ServerProcessor waiting for new message...
server.ServerProcessor waiting for new message...
server.ServerProcessor waiting for new message...
server.ServerProcessor waiting for new message...
server.ServerProcessor waiting for new message...
server.ServerProcessor waiting for new message...
server.ServerProcessor waiting for new message...
```

```
city: Brondesbury, code: NW6
city: Baker Street, code: W1

 We test the remote site printing.
[kkanniah@unix5 bin]$ java test2/ZipCodeRListClient 128.2.13.144 1099 Zip3 test1/
data.txt
java.net.ConnectException: Connection refused
        at java.net.PlainSocketImpl.socketConnect(Native Method)
        at java.net.AbstractPlainSocketImpl.doConnect(AbstractPlainSocketImpl.jav
a:339)
        at java.net.AbstractPlainSocketImpl.connectToAddress(AbstractPlainSocketI
mpl.java:200)
        at java.net.AbstractPlainSocketImpl.connect(AbstractPlainSocketImpl.java:
182)
        at java.net.SocksSocketImpl.connect(SocksSocketImpl.java:392)
        at java.net.Socket.connect(Socket.java:579)
        at java.net.Socket.connect(Socket.java:528)
        at java.net.Socket.<init>(Socket.java:425)
        at java.net.Socket.<init>(Socket.java:208)
        at communication.Communicator.sendAndReceiveMessage(Communicator.java:52)
        at core.Naming.lookup(Naming.java:43)
        at test2.ZipCodeRListClient.main(ZipCodeRListClient.java:45)
Exception in thread "main" core.Remote440Exception: Connection refused
        at core.Naming.lookup(Naming.java:62)
        at test2.ZipCodeRListClient.main(ZipCodeRListClient.java:45)
[kkanniah@unix5 bin]$ java test2/ZipCodeRListClient 128.2.13.144 2222 Zip3 test1/
data.txt
This is the original list.
city: Brick Lane, code: E2
city: Barrier Point, code: E16
city: Clindale, code: NW9
city: Brondesbury, code: NW6
city: Baker Street, code: W1
testing add.
add tested.

 This is the remote list, printed using find/next.
city: Brick Lane, code: E2
city: Barrier Point, code: E16
city: Clindale, code: NW9
city: Brondesbury, code: NW6
city: Baker Street, code: W1
[kkanniah@unix5 bin]$
```

Figure 4 Test 2 running on AFS



aghadiga@unix1:bin

```
server.ServerProcessor waiting for new message...
server.ServerProcessor waiting for new message...
server.ServerProcessor waiting for new message...
server.ServerProcessor waiting for new message...
server.ServerProcessor waiting for new message...
server.ServerProcessor waiting for new message...
server.ServerProcessor waiting for new message...
server.ServerProcessor waiting for new message...
server.ServerProcessor waiting for new message...
server.ServerProcessor waiting for new message...
server.ServerProcessor waiting for new message...
server.ServerProcessor waiting for new message...
server.ServerProcessor waiting for new message...
server.ServerProcessor waiting for new message...
server.ServerProcessor waiting for new message...
server.ServerProcessor waiting for new message...
server.ServerProcessor waiting for new message...
server.ServerProcessor waiting for new message...
server.ServerProcessor waiting for new message...
server.ServerProcessor waiting for new message...
server.ServerProcessor waiting for new message...
server.ServerProcessor waiting for new message...
server.ServerProcessor waiting for new message...
server.ServerProcessor waiting for new message...
server.ServerProcessor waiting for new message...
server.ServerProcessor waiting for new message...
server.ServerProcessor waiting for new message...
server.ServerProcessor waiting for new message...
server.ServerProcessor waiting for new message...
server.ServerProcessor waiting for new message...
server.ServerProcessor waiting for new message...
server.ServerProcessor waiting for new message...
server.ServerProcessor waiting for new message...
server.ServerProcessor waiting for new message...
server.ServerProcessor waiting for new message...
server.ServerProcessor waiting for new message...
server.ServerProcessor waiting for new message...
server.ServerProcessor waiting for new message...
server.ServerProcessor waiting for new message...
server.ServerProcessor waiting for new message...
server.ServerProcessor waiting for new message...
server.ServerProcessor waiting for new message...
```

kkanniah@unix5:bin

```
txt
This is the original list.
city: Brick Lane, code: E2
city: Barrier Point, code: E16
city: Clindale, code: NW9
city: Brondesbury, code: NW6
city: Baker Street, code: W1

 Server initalised.

 This is the remote list given by find.
city: Brick Lane, code: E2
city: Barrier Point, code: E16
city: Clindale, code: NW9
city: Brondesbury, code: NW6
city: Baker Street, code: W1

 This is the remote list given by findall.
city: Brick Lane, code: E2
city: Barrier Point, code: E16
city: Clindale, code: NW9
city: Brondesbury, code: NW6
city: Baker Street, code: W1

 We test the remote site printing.
[kkanniah@unix5 bin]$ java test3.NameServerClient 128.2.13.144 2222 Zip4 test1/da
ta.txt
This is the original list.
city: Brick Lane, code: ip5:5/E2
city: Barrier Point, code: ip4:4/E16
city: Clindale, code: ip3:3/NW9
city: Brondesbury, code: ip2:2/NW6
city: Baker Street, code: ip1:1/W1
testing add.
add tested.

 This is the remote list, printed using find/next.
city: Brick Lane, ror: ip5:5/E2
city: Barrier Point, ror: ip4:4/E16
city: Clindale, ror: ip3:3/NW9
city: Brondesbury, ror: ip2:2/NW6
city: Baker Street, ror: ip1:1/W1
[kkanniah@unix5 bin]$
```

Figure 5 Test 3 Running on AFS