

Assignment 3

Sayali Kudale

Documentation:

This program is to create a Mobile.jar which can be used by the agents to hop from one place to another. Place is the RMI server from which agent program can land and hop to the next environment. Agent is the class which will help user defined agents will travel from one place to another.

Place.java

Main Method:

To instantiate the Place class port number needs to be given as input. This port will be used to create a place that will keep on listening the incoming agents on the same port.

The main method will verify whether the given port number is valid.

If port is valid then RMI registry process is started which will keep on listening on the given port.

Then new Place object is instantiated and register to the rmiRegistry through Naming.rebind option.

After this step, Place is ready to accept the incoming agents.

Transfer method:

Transfer accepts the incoming agents and execute them as independent threads. To do that it follows below steps:

- It registers the agent into the agentloader. AgentLoader identifies the class of an incoming agent and registers it into its local class hash. By performing this we get a bytecode of a class into memory.
- Then we deserialise the entity and get the Object of Agent class.
- We set the unique identifier to identify the class. To do that we are using the hostaddress and agentsequencer. Hostaddress is converted into integer by removing the dot characters in it. After assigning id to the agent object the agentSequencer is incremented so that next agent from the same host will get new id.
- With the object of Agent thread is instantiated using thread.start().

Agent.java

Agent.java is the base class of user defined agents. This class contains identifier of the agent such as hostname, port, next host, and function to invoke. Below methods are changed to complete the implementation:

Run method:

When transfer function of a Place initiates the thread.start of Agent then that agent migrates as a new host and start execution as an independent thread.

In this run method, we check for the arguments , If arguments are null then we invoke method without having any parameters otherwise method is invoked by passing the arguments.

We are using reflection to get the method from the class object and function name. Then, same method is invoked using method.invoke().

Hop method:

Hop_method load agents byte into the memory again by using the class loader.

Serialize the object of class into the entity byte array.

Get the place Object by using naming.lookup.

Invoke the RMI transfer method of place object and pass the classname, bytecode and entity.

At the end, this thread object is killed.

MyAgent.java

My agent is simple user defined agent derived from Agent class.

It has defined function init, jump, step to demonstrate how agent is migrating from one host to another.

Source Code:

Kindly refer the **src** folder which contains the below files:

Mobile/Agent.java

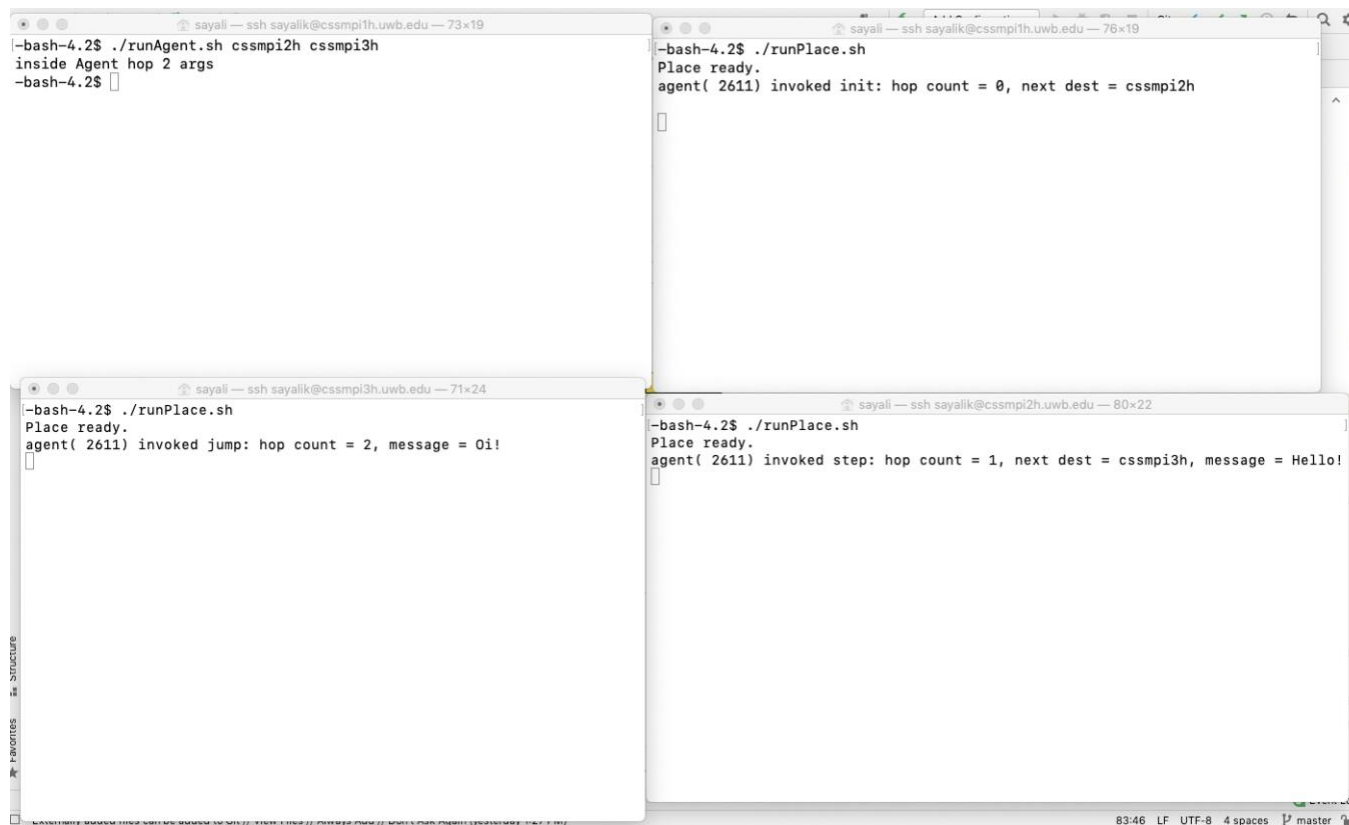
Mobile/Place.java

MyAgent.java

TestAgent.java

Execution Output:

1. Basic Implementation with 3 different Places:



```
-bash-4.2$ ./runAgent.sh cssmpi2h cssmpi3h
inside Agent hop 2 args
-bash-4.2$
```

```
-bash-4.2$ ./runPlace.sh
Place ready.
agent( 2611) invoked init: hop count = 0, next dest = cssmpi2h

```

```
-bash-4.2$ ./runPlace.sh
Place ready.
agent( 2611) invoked jump: hop count = 2, message = Oi!
```

```
-bash-4.2$ ./runPlace.sh
Place ready.
agent( 2611) invoked step: hop count = 1, next dest = cssmpi3h, message = Hello!
```

Figure 1 : Basic implementation

2. TestAgent with 4 different Places:

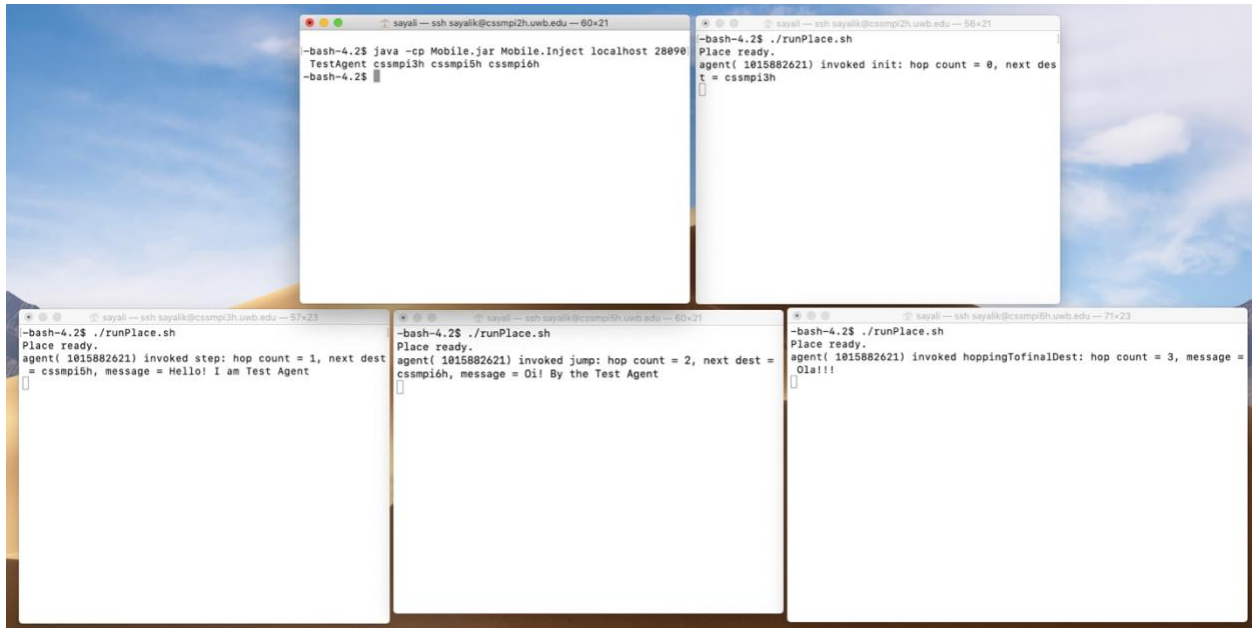


Figure 2: TestAgent with 4 Places

3. Implementation of Extra Feature:

a. Indirect inter-agent communication via Place

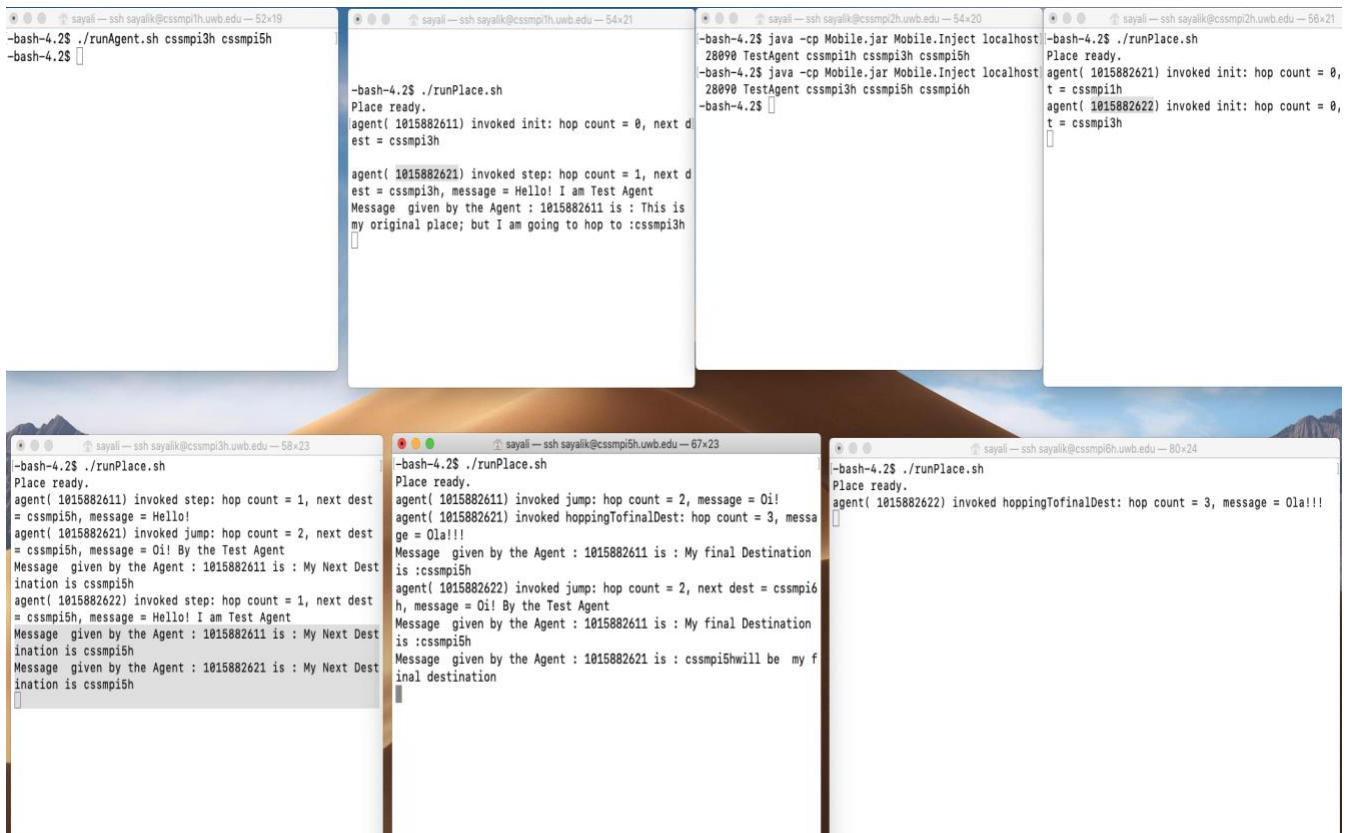


Figure 3 : Inter-agent communication via place

Discussions:

The program 3 was great hands-on RPC, dynamic linking and object serialization and deserialization. We created a library which can be used for mobile execution platform and create user defined agents.

Additional Feature:

We have implemented the indirect inter agent communication as an additional feature. In this feature agents can be communicated via place. Whenever agents come to a place it sees if there are any messages present to read if yes then reads it and then agent deposit message for other agents to read.

While depositing message agent also set a key for it and next agent needs to know a key to access the message.

Multiple agents will come to the transfer method of the Place. In transfer method we will do the reading and writing of the messages. To implement this feature, we have made below changes:

Added static Map *lstMessage* which will store the keys as passcode to read message and another map with key as host name and message from this agent.

Added static *msges* which is the list of messages at this place by multiple agents.

In transfer function we are reading whether previous agents have any messages for the given key and hostname. Also, this agent deposits the messages for upcoming agents for this key.

Agent's *messagesReceived* property gets updated by transfer function in a place. We are accessing this property and printing the messages in the myAgent in jump, step etc.

To demonstrate this functionality, we have created the TestAgent.java. TestAgent.java is implemented which can be hopped to 4 different places.

In Figure 3,

MyAgent is executed on cssmpi1h with destination as cssmpi3h and cssmpi5h.

MyAgent instantiated on cssmpi1h and it leaves message as "This is my original place; but I am going to hop to : cssmpi3h" with passcodeForMessage as "XXX".

MyAgent goes to cssmpi3h and leaves message as "My Next Destination is cssmpi5h" with the same key.

MyAgent goes to cssmpi5h and leaves message as "My final Destination is : cssmpi5h" with the same key.

TestAgent.java is executed on cssmpi2h with destination as cssmpi1h, cssmpi3h and cssmpi5h. TestAgent has the passcodeForMessage same as MyAgent hence all the messages that are given by MyAgent can be accessible to the TestAgent.

When TestAgent comes to cssmpi1h it read the message "This is my original place; but I am going to hop to : cssmpi3h"

When TestAgent comes to cssmpi3h it read the message "My Next Destination is cssmpi5h"

When TestAgent comes to cssmpi5h it read the message "My final Destination is : cssmpi5h"

Additionally, we are running another instance of the TestAgent.java on cssmpi2h it generates the new agent Id. We are passing destination as cssmpi3h, cssmpi5h, cssmpi6h.

This new agent (1015882622) can access messages by both the agents (1015882611, 1015882621) which were earlier present on places.

Improvements in the program:

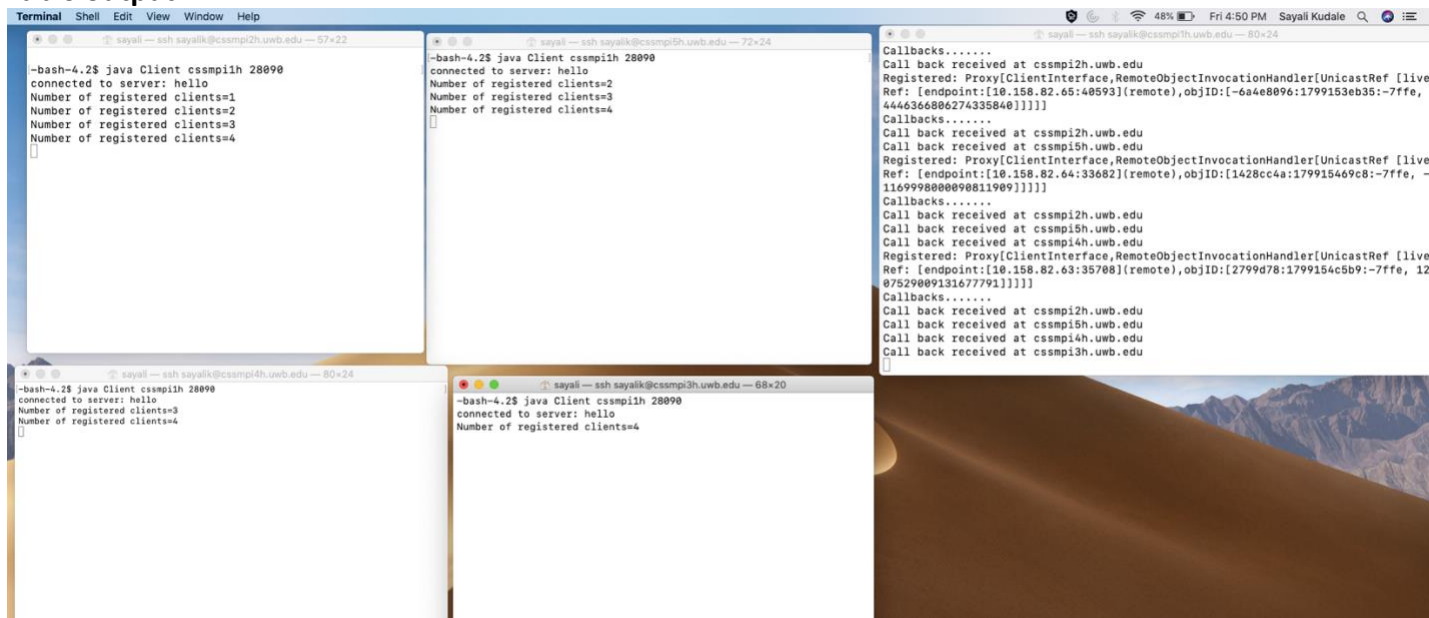
1. In current implementation of mobile execution platform, we have implemented indirect inter agent communication, where agents can deposit messages on the places for upcoming agents. We can improve this program by implementing the direct agent communication where agents which are on different places can communicate with each other.
2. If any of the agents crashed in one of the places, then that place needs to be restarted. We should improve this program so that even though one of the agent crashes then also place should be active for other agents.
3. In this program there are multiple exceptions such as targetInvocationExecetion that is arising, and there should be better exception handing implementation for such exceptions.

Limitations:

1. This mobile execution platform is based on RMI and can only be used in java supported platforms.
2. This mobile execution platform has limited functionality because of the security restrictions. The mobile Agents can be injected to any machine without any security or authentication.
3. It is difficult to identify which objects are remote and which are local.

Lab Sessions:

Lab 6 output:



```
Terminal Shell Edit View Window Help
sayali - ssh sayalik@cssmpi1h.uwb.edu - 57x22
-bash-4.2$ java Client cssmpi1h 28090
connected to server: hello
Number of registered clients=1
Number of registered clients=2
Number of registered clients=3
Number of registered clients=4

sayali - ssh sayalik@cssmpi1h.uwb.edu - 72x24
-bash-4.2$ java Client cssmpi1h 28090
connected to server: hello
Number of registered clients=2
Number of registered clients=3
Number of registered clients=4

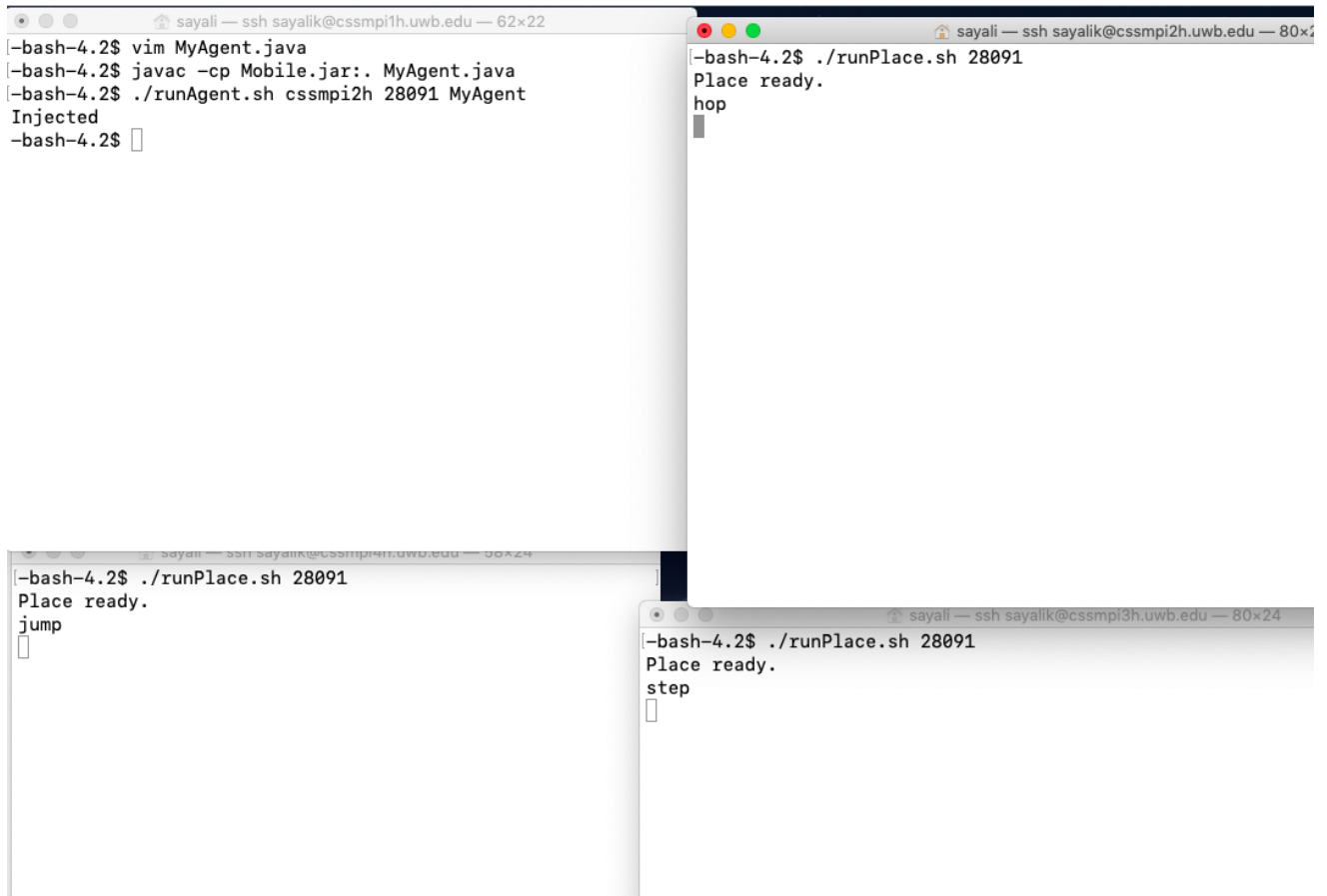
sayali - ssh sayalik@cssmpi1h.uwb.edu - 80x24
Callbacks.....
Call back received at cssmpi1h.uwb.edu
Registered: Proxy[ClientInterface,RemoteObjectInvocationHandler[UnicastRef [live
Ref: [endpoint:[10.158.82.65:40593](remote),objID:[-6a4e8896:179915eb35:-7ffe,
444636e006274335840]]]]]
Callbacks.....
Call back received at cssmpi1h.uwb.edu
Call back received at cssmpi1h.uwb.edu
Registered: Proxy[ClientInterface,RemoteObjectInvocationHandler[UnicastRef [live
Ref: [endpoint:[10.158.82.64:33682](remote),objID:[1428cc4a:179915469c8:-7ffe, -
116999800090811909]]]]]
Callbacks.....
Call back received at cssmpi1h.uwb.edu
Call back received at cssmpi1h.uwb.edu
Call back received at cssmpi1h.uwb.edu
Registered: Proxy[ClientInterface,RemoteObjectInvocationHandler[UnicastRef [live
Ref: [endpoint:[10.158.82.63:35708](remote),objID:[2799d78:1799154c5b9:-7ffe, 12
07529009131677791]]]]]
Callbacks.....
Call back received at cssmpi1h.uwb.edu
Call back received at cssmpi1h.uwb.edu
Call back received at cssmpi1h.uwb.edu
Call back received at cssmpi1h.uwb.edu
Call back received at cssmpi1h.uwb.edu

sayali - ssh sayalik@cssmpi1h.uwb.edu - 80x24
-bash-4.2$ java Client cssmpi1h 28090
connected to server: hello
Number of registered clients=3
Number of registered clients=4

sayali - ssh sayalik@cssmpi1h.uwb.edu - 68x20
-bash-4.2$ java Client cssmpi1h 28090
connected to server: hello
Number of registered clients=4
```

Figure 4 : Lab 6 output

Lab7 output:



The image displays four terminal windows from a macOS environment, each showing the execution of a Java-based agent and its subsequent actions. The windows are arranged in a 2x2 grid. Each window has a title bar with the user 'sayali' and the host 'cssmpi' followed by a port number (1h, 2h, 3h, 4h) and a resolution (62x22, 80x24, 80x24). The first window (top-left) shows the compilation of 'MyAgent.java' and its execution on 'cssmpi2h' port '28091', resulting in an 'Injected' status. The second window (top-right) shows the execution of './runPlace.sh 28091' on 'cssmpi2h', outputting 'Place ready.' and 'hop'. The third window (bottom-left) shows the execution of './runPlace.sh 28091' on 'cssmpi4h', outputting 'Place ready.' and 'jump'. The fourth window (bottom-right) shows the execution of './runPlace.sh 28091' on 'cssmpi3h', outputting 'Place ready.' and 'step'.

```
sayali — ssh sayalik@cssmpi1h.uwb.edu — 62x22
-bash-4.2$ vim MyAgent.java
-bash-4.2$ javac -cp Mobile.jar:. MyAgent.java
-bash-4.2$ ./runAgent.sh cssmpi2h 28091 MyAgent
Injected
-bash-4.2$

sayali — ssh sayalik@cssmpi2h.uwb.edu — 80x24
-bash-4.2$ ./runPlace.sh 28091
Place ready.
hop

sayali — ssh sayalik@cssmpi4h.uwb.edu — 80x24
-bash-4.2$ ./runPlace.sh 28091
Place ready.
jump

sayali — ssh sayalik@cssmpi3h.uwb.edu — 80x24
-bash-4.2$ ./runPlace.sh 28091
Place ready.
step
```

Figure 5 : Lab 7 output