Parallel and Distributed Computing – Spring 2021

Momina Atif Dar
P18-0030
Section: B

Assignment 02

Remote Procedure Call (RPC)

Remote Procedure Calls are like a function which are needed to perform some task on another machine i.e. remote PC, by a program located on that remote PC. It uses client-server model where the requesting program is client and the program that will carry out the service is called server. When RPC request is being carried out, the client is sent to blocking state until the request is processed and results are returned. Threads are used to manage multiple RPCs that share the same address space.
When a procedure call is executed, the client calls the client stub which takes the parameters from client and packs them into a message and sends this message to remote server machine. On the server end, the message is received by server stub which unpacks the message and sends it to server. After the service is done, server stub receives the return values and packs them in a message. This message is sent to client stub which unpacks the message and sends the return values to client machine.

Code:

Server.py

server.py
~/Documents/Semester-6/PDC/pyRpc-master/examples

```python
#!/usr/bin/env python

import time
import logging

import pyRpc

logging.basicConfig(level=logging.DEBUG)


def main():
    # using local IPC communication
    server = pyRpc.PyRpc("Server", workers=2)

    # Could have used a TCP connection if we wanted:
    # server = pyRpc.PyRpc("Server", tcpaddr="127.0.0.1:40000")

    time.sleep(.1)

    server.publishService(slow)
    server.publishService(fast)
    server.publishService(noReturn)
    server.start()

    try:
        while True:
            time.sleep(.5)
    except KeyboardInterrupt:
        server.stop()


counter = 0


def slow(*args, **kwargs):
    "This does something and returns values"

    global counter

    print("slow() called! Doing some stuff.")
    time.sleep(3)
    print("Done!")
    counter += 1
    return counter


```

```python
def fast(*args, **kwargs):
    "This does something and returns values"

    global counter

    print("fast() called! Doing some stuff.")
    counter += 1
    return counter


def noReturn(value=1):
    "This does something and returns nothing"
    print("noReturn() called!")
    time.sleep(2)
    print("noReturn() done!")
    return 1


if __name__ == "__main__":
    main()
```

Client.py

client.py
~/Documents/Semester-6/PDC/pyRpc-master/examples

```python
#!/usr/bin/env python

"""
In this example, make sure to start the server first,
as this client will try and communicate immediately.
"""

import time
from pyRpc import RpcConnection

ASYNC_CALLS = 0


def callback(resp, *args, **kwargs):
    global ASYNC_CALLS
    print("Got slow response:", resp.result)
    ASYNC_CALLS += 1


if __name__ == "__main__":

    remote = RpcConnection("Server", workers=1)

    # if the server were using a TCP connection:
    # remote = RpcConnection("Server", tcpaddr="127.0.0.1:40000")

    time.sleep(.1)

    print("Calling slow()")

    for i in range(5):
        remote.call("slow", is_async=True, callback=callback)

    print("Calling fast()")
    resp = remote.call("fast")
    print("Got fast response:", resp.result)

    print("Waiting on async calls to finish")
    while ASYNC_CALLS < 5:
        time.sleep(.1)
```

Output:

```
(base) momina@death-eater:~/Documents/Semester-6/PDC/pyRpc-master/examples$ python server.py
DEBUG:pyRpc.server:Starting RPC thread loop w/ 2 worker(s)
DEBUG:pyRpc.server:Listening @ ipc:///tmp/Server.ipc
DEBUG:pyRpc.server:request received by thread RPC-Worker-1: <RpcRequest: slow (#args:0, #kwargs:0)>
slow() called! Doing some stuff.
DEBUG:pyRpc.server:request received by thread RPC-Worker-2: <RpcRequest: fast (#args:0, #kwargs:0)>
fast() called! Doing some stuff.
DEBUG:pyRpc.server:sent response: <RpcResponse: status:0>
Done!
DEBUG:pyRpc.server:sent response: <RpcResponse: status:0>
DEBUG:pyRpc.server:request received by thread RPC-Worker-1: <RpcRequest: slow (#args:0, #kwargs:0)>
slow() called! Doing some stuff.
Done!
DEBUG:pyRpc.server:sent response: <RpcResponse: status:0>
DEBUG:pyRpc.server:request received by thread RPC-Worker-2: <RpcRequest: slow (#args:0, #kwargs:0)>
slow() called! Doing some stuff.
Done!
DEBUG:pyRpc.server:sent response: <RpcResponse: status:0>
DEBUG:pyRpc.server:request received by thread RPC-Worker-1: <RpcRequest: slow (#args:0, #kwargs:0)>
slow() called! Doing some stuff.
Done!
DEBUG:pyRpc.server:sent response: <RpcResponse: status:0>
DEBUG:pyRpc.server:request received by thread RPC-Worker-2: <RpcRequest: slow (#args:0, #kwargs:0)>
slow() called! Doing some stuff.
Done!
DEBUG:pyRpc.server:sent response: <RpcResponse: status:0>
```

```
(base) momina@death-eater:~/Documents/Semester-6/PDC/pyRpc-master/examples$ python client.py
Calling slow()
Calling fast()
Got fast response: 1
Waiting on async calls to finish
Got slow response: 2
Got slow response: 3
Got slow response: 4
Got slow response: 5
Got slow response: 6
```

Explanation:

In this example, client side is invoking methods like ‘fast’ and ‘slow’ that are implemented on server side.

Remote Method Invocation (RMI)

Remote Method Invocation is like RPC but it is an API that provides distributed computing in java. It allows objects to invoke methods on an object running in another Java Virtual Machine (JVM). It has two important objects for communication with remote object; stub on client side and skeleton on server side. Stub initiates connection with the other JVM and sends parameters from client to that JVM and waits for result. On the server end, skeleton reads the received parameter and invokes the method and sends it that parameter. When the service is done, it receives return values and send it to stub on the client side. Stub reads the return values and sends them to client.

Code:

Interface.java

Open ▾ Interface.java ~/Documents/Semester-6/PDC/Java-remote-method-invocation-RMI--master/src Save

```java
import java.net.UnknownHostException;
import java.rmi.Remote;
import java.rmi.RemoteException;


public interface Interface extends Remote {


public String getnom()throws RemoteException, UnknownHostException;



public String   getip() throws RemoteException, UnknownHostException;



public String getmac()throws RemoteException, UnknownHostException;



//public String getram()throws RemoteException, UnknownHostException;


public String getprocesseur()throws RemoteException, UnknownHostException;

//public void addition()

}
```

Server.java:

*Server.java
~/Documents/Semester-6/PDC/Java-remote-method-invocation-RMI--master/src

```java
import java.net.InetAddress;
import java.net.UnknownHostException;
import java.rmi.*;

import java.rmi.server.*;


public class Server extends UnicastRemoteObject implements Interface {

public Server() throws RemoteException {

super(); }


public String getnom() throws RemoteException, UnknownHostException{

        InetAddress address = InetAddress.getLocalHost();

        return address.getHostName();
}


public  String getip() throws RemoteException, UnknownHostException{
        InetAddress address = InetAddress.getLocalHost();

return address.getHostAddress();

}


public String getmac() throws RemoteException, UnknownHostException{

         InetAddress address = InetAddress.getLoopbackAddress();

         return address.getHostAddress();
}

//public String getram(){
//      InetAddress address = InetAddress.getByName(getnom());

//      return address;
//}


public String getprocesseur(){

        return " ";
}
```

```java
public static void main(String[] args) {

try {

Server objet = new Server();

Naming.rebind("rmi://localhost/Server", objet);

System.out.println("Server part");

}

catch(Exception e) {

System.err.println("Error: " + e.getMessage());

}}}

```

Client.java:

Open ▾ *Client.java ~/Documents/Semester-6/PDC/Java-remote-method-invocation-RMI--master/src

```java
 import java.rmi.Naming;
public class Client {

  static String resultat1;
  static String resultat2;
  static String resultat3;
  static String resultat4;

public static void main(String[] args) {

try {

Interface reference = (Interface)Naming.lookup("rmi://localhost/Server");

resultat1 = reference.getnom() ;
System.out.println("Nom: " + resultat1);


resultat2 = reference.getip() ;
System.out.println("IP: " + resultat2);


resultat3 = reference.getmac() ;
System.out.println("MAC: " + resultat3);


//resultat4 = reference.getram() ;
//System.out.println("RAM: " + resultat4);


}


catch(Exception e) { System.err.println("Error: " + e.getMessage()); }


} }
```

Output:

There's some error while running the files on terminal as well as Eclipse IDE that is why output is not generated.

```
(base) momina@death-eater:~/Documents/Semester-6/PDC/Java-remote-method-invocat
ion-RMI--master/src$ java Server
Error: LinkageError occurred while loading main class Server
        java.lang.UnsupportedClassVersionError: Server has been compiled by a m
ore recent version of the Java Runtime (class file version 58.0), this version
of the Java Runtime only recognizes class file versions up to 55.0
(base) momina@death-eater:~/Documents/Semester-6/PDC/Java-remote-method-invocat
ion-RMI--master/src$ _
```

Explanation:

In this example, the client side is provoking methods to get to know about server side's specifications like IP, MAC address and Host name.
