Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India (Autonomous College Affiliated to University of Mumbai)

Keyur Pancholi (2020300047)
Jai Patel (2020300052)
Aaryan Purohit (2020300055)

Aim:

Implementation of Client Server Communication using RPC/RMI

Theory

RPC

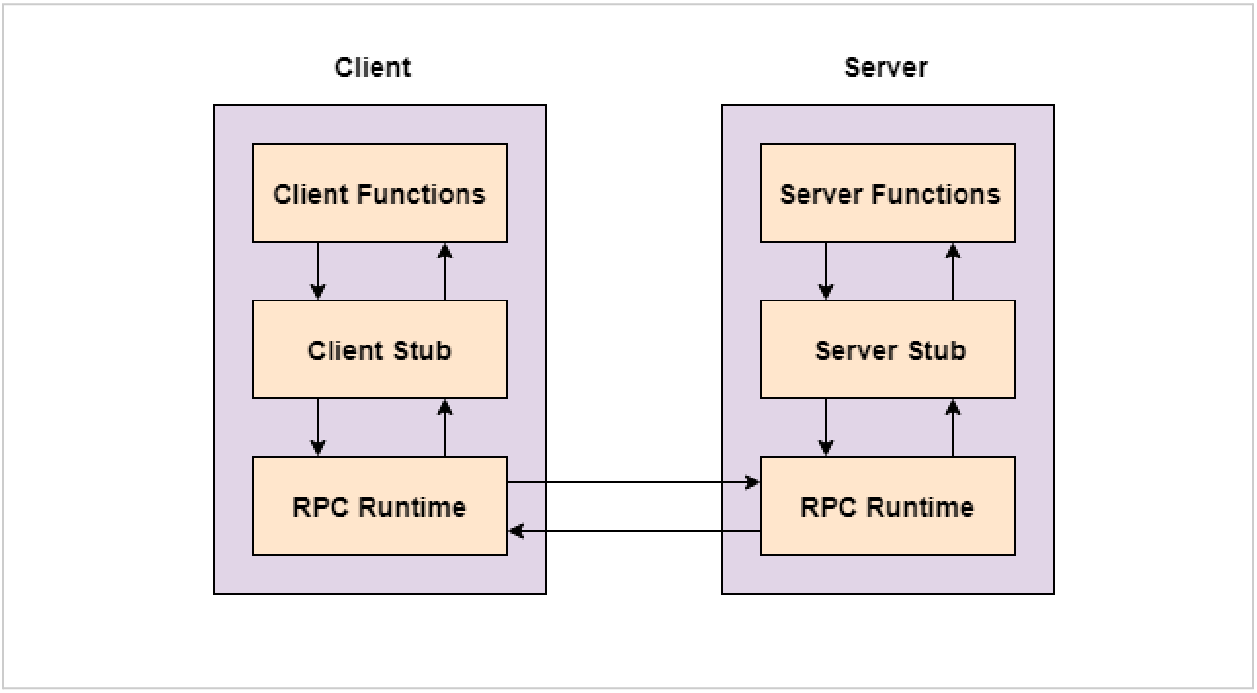
A remote procedure call is an interprocess communication technique that is used for client-server based applications. It is also known as a subroutine call or a function call.

A client has a request message that the RPC translates and sends to the server. This request may be a procedure or a function call to a remote server. When the server receives the request, it sends the required response back to the client. The client is blocked while the server is processing the call and only resumed execution after the server is finished.

The sequence of events in a remote procedure call are given as follows –

- The client stub is called by the client.
- The client stub makes a system call to send the message to the server and puts the parameters in the message.
- The message is sent from the client to the server by the client's operating system.
- The message is passed to the server stub by the server operating system.
- The parameters are removed from the message by the server stub.
- Then, the server procedure is called by the server stub.

A diagram that demonstrates this is as follows –



Advantages of Remote Procedure Call

Some of the advantages of RPC are as follows –

- Remote procedure calls support process oriented and thread oriented models.
- The internal message passing mechanism of RPC is hidden from the user.
- The effort to re-write and re-develop the code is minimum in remote procedure calls.
- Remote procedure calls can be used in distributed environment as well as the local environment.
- Many of the protocol layers are omitted by RPC to improve performance.

Disadvantages of Remote Procedure Call

Some of the disadvantages of RPC are as follows –

- The remote procedure call is a concept that can be implemented in different ways. It is not a standard.
- There is no flexibility in RPC for hardware architecture. It is only interaction based.
- There is an increase in costs because of remote procedure call.

RMI

The **RMI** (Remote Method Invocation) is an API that provides a mechanism to create distributed application in java. The RMI allows an object to invoke methods on an object running in another JVM.

The RMI provides remote communication between the applications using two objects *stub* and *skeleton*.

Understanding stub and skeleton

RMI uses stub and skeleton object for communication with the remote object.

A **remote object** is an object whose method can be invoked from another JVM. Let's understand the stub and skeleton objects:

stub

The stub is an object, acts as a gateway for the client side. All the outgoing requests are routed through it. It resides at the client side and represents the remote object. When the caller invokes method on the stub object, it does the following tasks:

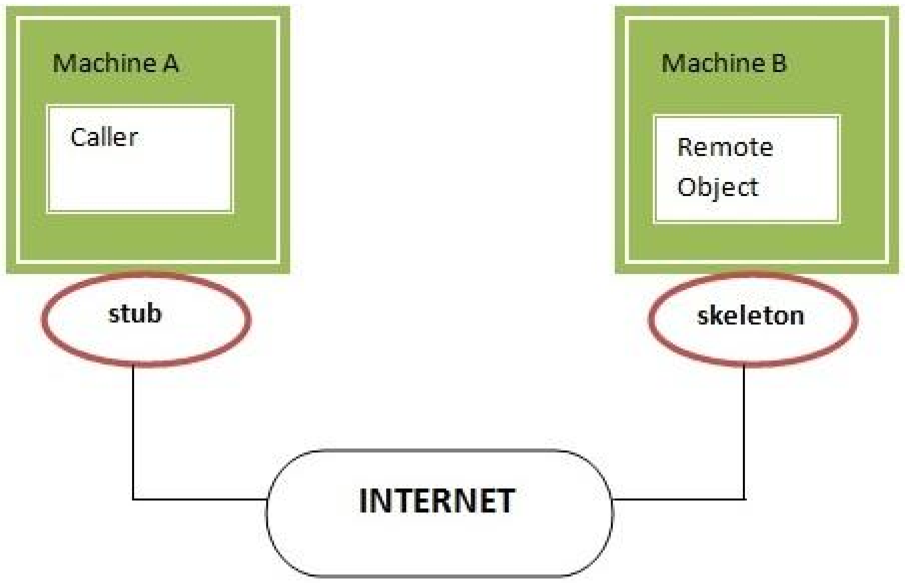
1. It initiates a connection with remote Virtual Machine (JVM),
2. It writes and transmits (marshals) the parameters to the remote Virtual Machine (JVM),
3. It waits for the result
4. It reads (unmarshals) the return value or exception, and
5. It finally, returns the value to the caller.

skeleton

The skeleton is an object, acts as a gateway for the server side object. All the incoming requests are routed through it. When the skeleton receives the incoming request, it does the following tasks:

1. It reads the parameter for the remote method
2. It invokes the method on the actual remote object, and
3. It writes and transmits (marshals) the result to the caller.

In the Java 2 SDK, an stub protocol was introduced that eliminates the need for



skeletons.

RPC vs RMI

Sr. No.	RPC	RMI
1	RPC is a library and OS dependent platform.	Whereas it is a java platform.
2	RPC supports procedural programming.	RMI supports object oriented programming.
3	RPC is less efficient in comparison of RMI.	While RMI is more efficient than RPC.
4	RPC creates more overhead.	While it creates less overhead than RPC.
5	The parameters which are passed in RPC are ordinary or normal data.	While in RMI, objects are passed as parameter.

6	RPC is the older version of RMI.	While it is the successor version of RPC.
7	There is high Provision of ease of programming in RPC.	While there is low Provision of ease of programming in RMI.
8	RPC does not provide any security.	While it provides client level security.
9	It's development cost is huge.	While it's development cost is fair or reasonable.
10	There is a huge problem of versioning in RPC.	While there is possible versioning using RDMI.
11	There is multiple codes are needed for simple application in RPC.	While there is multiple codes are not needed for simple application in RMI.

SOCKET

Socket programming is a way of connecting two nodes on a network to communicate with each other. One socket(node) listens on a particular port at an IP, while the other socket reaches out to the other to form a connection. The server forms the listener socket while the client reaches out to the server.

They are the real backbones behind web browsing. In simpler terms, there is a server and a client.

Socket programming is started by importing the socket library and making a simple socket.

```
import socket
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

Here we made a socket instance and passed it two parameters. The first parameter is **AF_INET** and the second one is **SOCK_STREAM**. AF_INET refers to the address-family ipv4. The SOCK_STREAM means connection-oriented TCP protocol.

Now we can connect to a server using this socket.3

Connecting to a server:

Note that if any error occurs during the creation of a socket then a socket. error is thrown and we can only connect to a server by knowing its IP. You can find the IP of the server by using this :

```
$ ping www.google.com
```

Code

Client

```
import socket
import random
from threading import Thread
from datetime import datetime
from colorama import Fore, init, Back

init()

colors = [Fore.BLUE, Fore.CYAN, Fore.GREEN,
Fore.LIGHTBLACK_EX,
          Fore.LIGHTBLUE_EX, Fore.LIGHTCYAN_EX,
Fore.LIGHTGREEN_EX,
          Fore.LIGHTMAGENTA_EX, Fore.LIGHTRED_EX,
Fore.LIGHTWHITE_EX,
          Fore.LIGHTYELLOW_EX, Fore.MAGENTA, Fore.RED,
Fore.WHITE, Fore.YELLOW
          ]
```

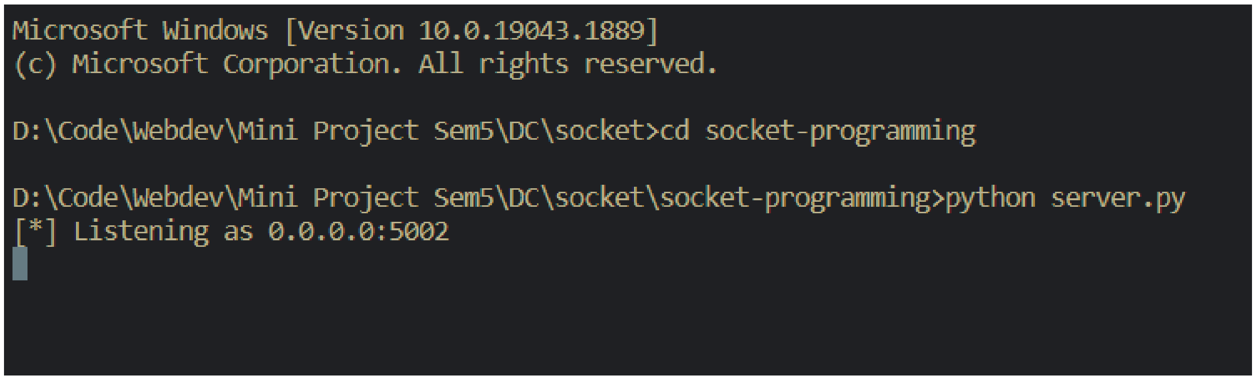
```python
client_color = random.choice(colors)


SERVER_HOST = "127.0.0.1"
SERVER_PORT = 5002
separator_token = "<SEP>"

s = socket.socket()
print(f"[*] Connecting to {SERVER_HOST}:{SERVER_PORT}...")
s.connect((SERVER_HOST, SERVER_PORT))
print("[+] Connected.")
name = input("Enter your name: ")
s.send(("1"+name).encode())


def listen_for_messages():
    while True:
        message = s.recv(1024).decode()
        if message[0] == '1' and message[1:] != name:
            print(f"{client_color}{message[1:]} connected")
            continue

        if message[1:] == name:
            continue

        print("\n" + message)


t = Thread(target=listen_for_messages)
t.daemon = True
t.start()

while True:
    to_send = input()
    if to_send.lower() == 'q':
        break
    date_now = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
    to_send = f"{client_color}[{date_now}]
{name}{separator_token}{to_send}{Fore.RESET}"
    s.send(to_send.encode())

s.close()
```

Server

```
import socket
from threading import Thread

SERVER_HOST = "0.0.0.0"
SERVER_PORT = 5002
separator_token = "<SEP>"

client_sockets = set()
s = socket.socket()
s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
s.bind((SERVER_HOST, SERVER_PORT))
s.listen(5)
print(f"[*] Listening as {SERVER_HOST}:{SERVER_PORT}")


def listen_for_client(cs):
    while True:
        try:
            msg = cs.recv(1024).decode()
            if msg[0] == '1':
                print(msg[1:], "connected")
        except Exception as e:
            print(f"[!] Error: {e}")
            client_sockets.remove(cs)
        else:
            msg = msg.replace(separator_token, ": ")
        for client_socket in client_sockets:
            client_socket.send(msg.encode())


while True:
    client_socket, client_address = s.accept()
    print(f"[+] {client_address} connected.")
    client_sockets.add(client_socket)
    t = Thread(target=listen_for_client,
args=(client_socket,))
    t.daemon = True
    t.start()
```

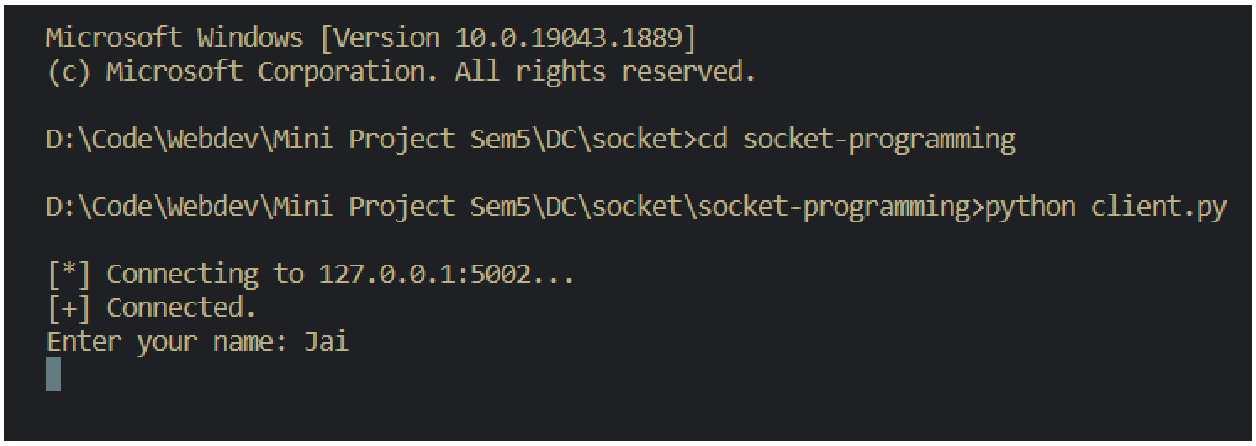
Output

Binding Socket to a Port

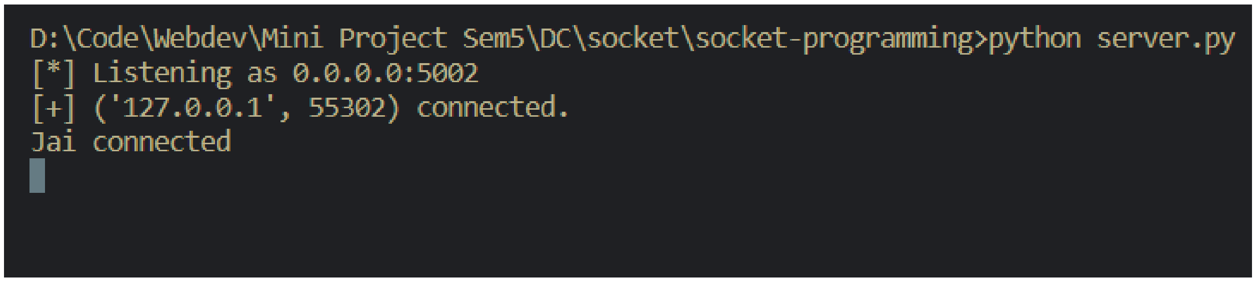


```
Microsoft Windows [Version 10.0.19043.1889]
(c) Microsoft Corporation. All rights reserved.

D:\Code\Webdev\Mini Project Sem5\DC\socket>cd socket-programming

D:\Code\Webdev\Mini Project Sem5\DC\socket\socket-programming>python server.py
[*] Listening as 0.0.0.0:5002
```

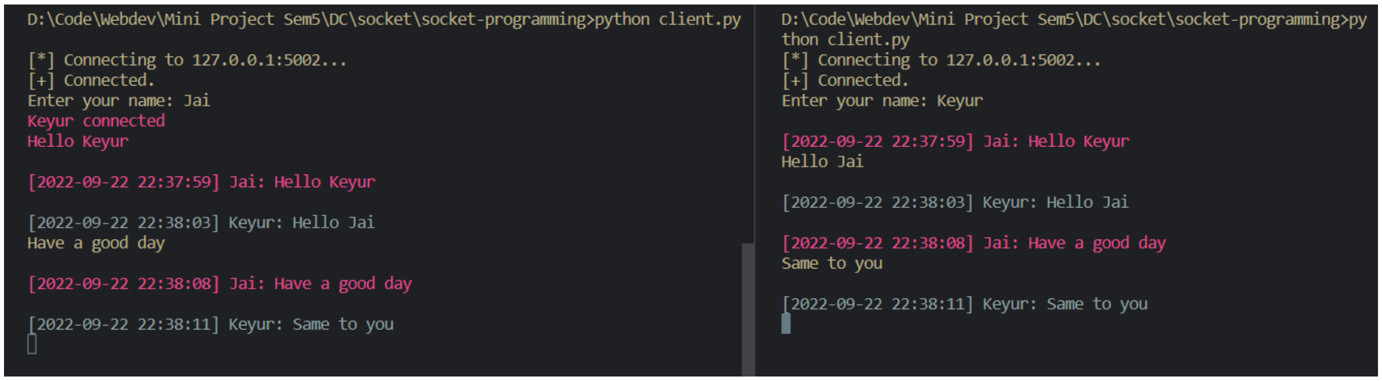
User Joins



```
Microsoft Windows [Version 10.0.19043.1889]
(c) Microsoft Corporation. All rights reserved.

D:\Code\Webdev\Mini Project Sem5\DC\socket>cd socket-programming

D:\Code\Webdev\Mini Project Sem5\DC\socket\socket-programming>python client.py

[*] Connecting to 127.0.0.1:5002...
[+] Connected.
Enter your name: Jai
```



```
D:\Code\Webdev\Mini Project Sem5\DC\socket\socket-programming>python server.py
[*] Listening as 0.0.0.0:5002
[+] ('127.0.0.1', 55302) connected.
Jai connected
```



```
D:\Code\Webdev\Mini Project Sem5\DC\socket\socket-programming>python client.py

[*] Connecting to 127.0.0.1:5002...
[+] Connected.
Enter your name: Jai
Keyur connected
Hello Keyur

[2022-09-22 22:37:59] Jai: Hello Keyur

[2022-09-22 22:38:03] Keyur: Hello Jai
Have a good day

[2022-09-22 22:38:08] Jai: Have a good day

[2022-09-22 22:38:11] Keyur: Same to you
```

```
D:\Code\Webdev\Mini Project Sem5\DC\socket\socket-programming>py
thon client.py
[*] Connecting to 127.0.0.1:5002...
[+] Connected.
Enter your name: Keyur

[2022-09-22 22:37:59] Jai: Hello Keyur
Hello Jai

[2022-09-22 22:38:03] Keyur: Hello Jai

[2022-09-22 22:38:08] Jai: Have a good day
Same to you

[2022-09-22 22:38:11] Keyur: Same to you
```