Name : Ansari M.Saeem M.Saleem

Uid : 2019430001

Subject : NAD

Expt no : 3

Aim : Implementation and analysis of Client Server Program (RPC) through parallel processing.

Aim:

Implementation and analysis of Client Server Program (RPC) through parallel processing.

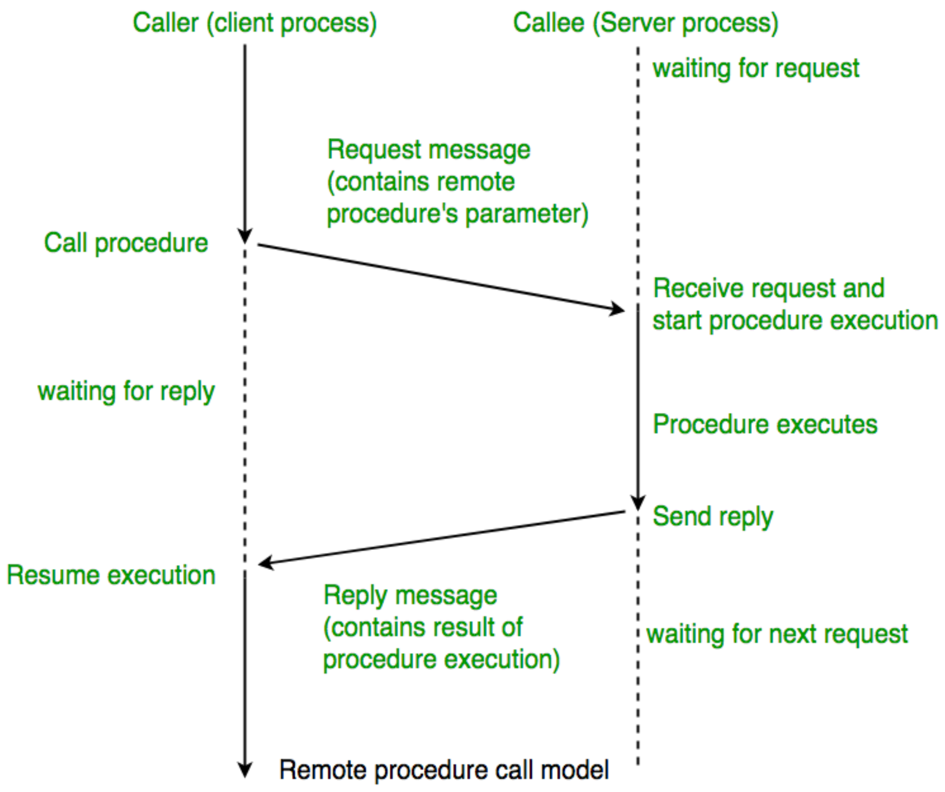
Objectives:

- To establish connections between multiple clients and a single server.
- To implement a inter process communication for client server applications.

Theory:

What is RPC?

Remote Procedure Call (RPC) is a powerful technique for constructing distributed, client-server-based applications. It is based on extending the conventional local procedure calling so that the called procedure need not exist in the same address space as the calling procedure. The two processes may be on the same system, or they may be on different systems with a network connecting them.



Remote procedure call model

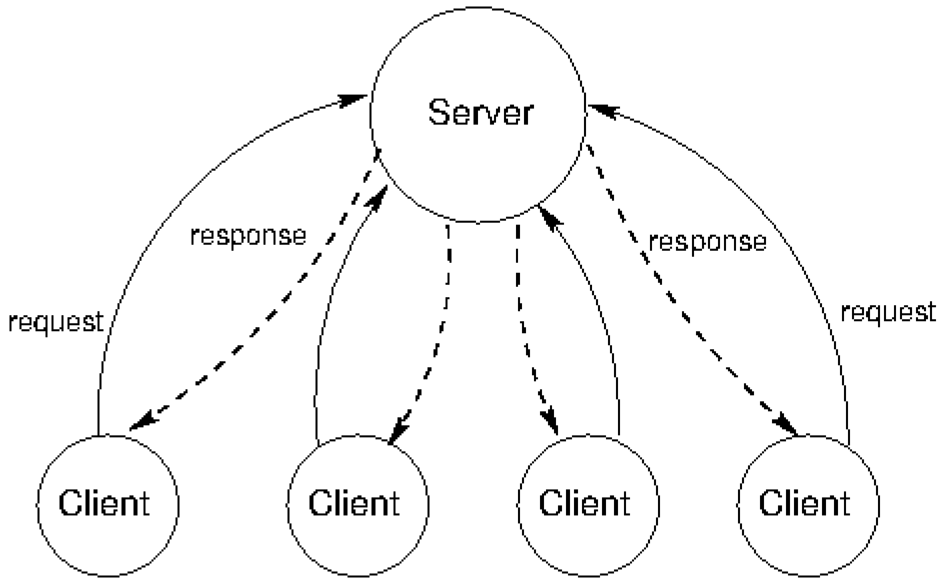
Working of RPC:

1. A client invokes a client stub procedure, passing parameters in the usual way. The client stub resides within the client’s own address space.
2. The client stub Marshalls(pack) the parameters into a message. Marshalling includes converting the representation of the parameters into a standard format, and copying each parameter into the message.
3. The client stub passes the message to the transport layer, which sends it to the remote server machine.
4. On the server, the transport layer passes the message to a server stub, which demarshalls(unpack) the parameters and calls the desired server routine using the regular procedure call mechanism.
5. When the server procedure completes, it returns to the server stub (e.g., via a normal procedure call return), which Marshalls the return values into a message. The server stub then hands the message to the transport layer.
6. The transport layer sends the result message back to the client transport layer, which hands the message back to the client stub.
7. The client stub demarshalls the return parameters and execution return to the caller.
8. When the server procedure completes, it returns to the server stub (e.g., via a normal procedure call return), which Marshalls the return values into a message. The server stub then hands the message to the transport layer.
9. The transport layer sends the result message back to the client transport layer, which hands the message back to the client stub.
10. The client stub demarshalls the return parameters and execution returns to the caller.

Client Server Systems

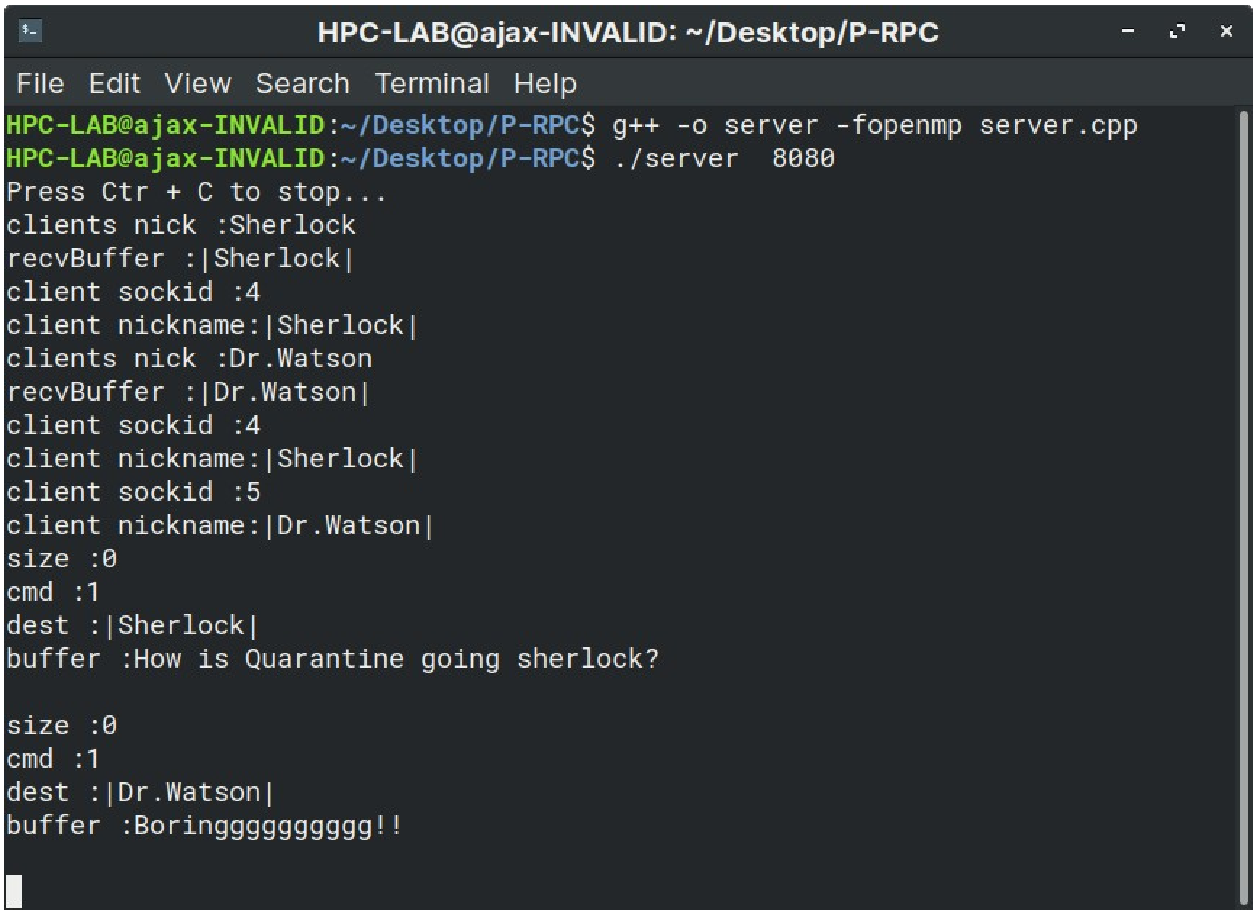
There is a single server that provides a service, and multiple clients that communicate with the server to consume its products. In this architecture, clients and servers have different jobs. The server's job is to respond to service requests from clients, while a client's job is to use the data provided in response in order to perform some task.

On the web, we think of clients and servers as being on different machines, but even systems on a single machine can have client/server architectures. For example, signals from input devices on a computer need to be generally available to programs running on the computer. The programs are clients, consuming mouse and keyboard input data. The operating system's device drivers are the servers, taking in physical signals and serving them up as usable input.



Results:

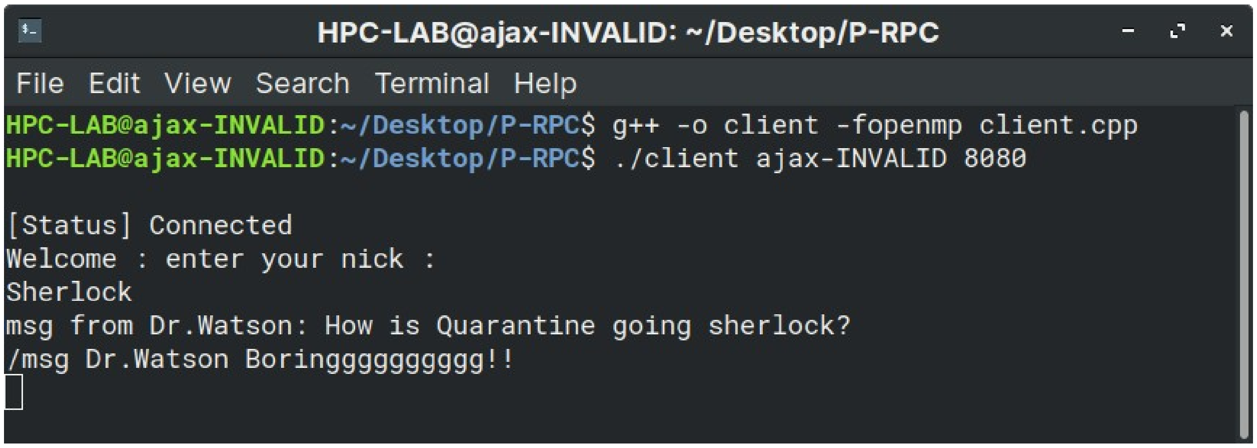
1. Server keeping the track of all clients old and newly added and their messages.


HPC-LAB@ajax-INVALID: ~/Desktop/P-RPC
File Edit View Search Terminal Help

```
HPC-LAB@ajax-INVALID:~/Desktop/P-RPC$ g++ -o server -fopenmp server.cpp
HPC-LAB@ajax-INVALID:~/Desktop/P-RPC$ ./server  8080
Press Ctr + C to stop...
clients nick :Sherlock
recvBuffer :|Sherlock|
client sockid :4
client nickname:|Sherlock|
clients nick :Dr.Watson
recvBuffer :|Dr.Watson|
client sockid :4
client nickname:|Sherlock|
client sockid :5
client nickname:|Dr.Watson|
size :0
cmd :1
dest :|Sherlock|
buffer :How is Quarantine going sherlock?

size :0
cmd :1
dest :|Dr.Watson|
buffer :Boringgggggggg!!
```

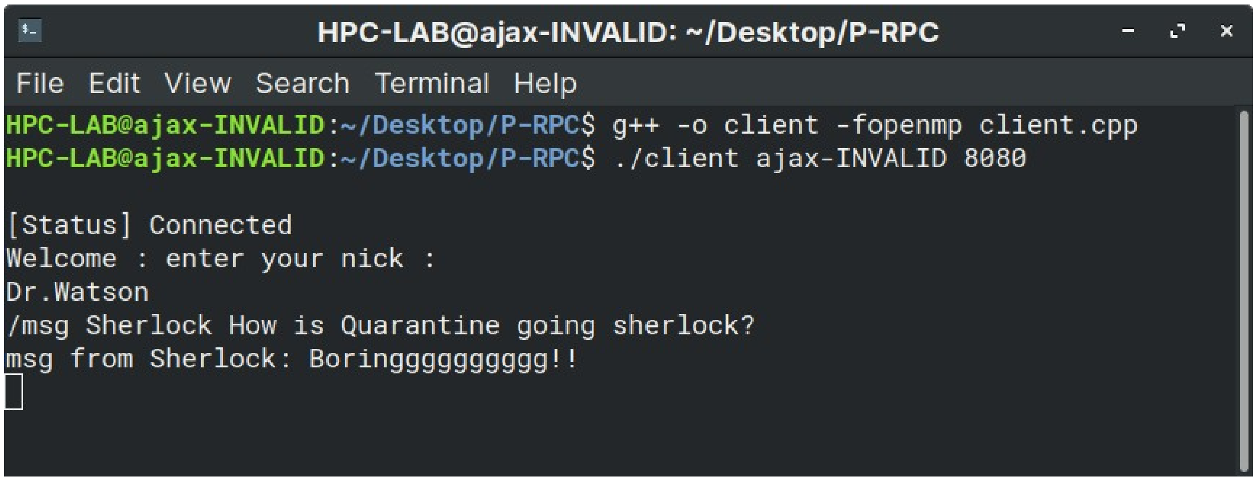
2. **Client (1) sending message to Client (2).**


HPC-LAB@ajax-INVALID: ~/Desktop/P-RPC
File Edit View Search Terminal Help

```
HPC-LAB@ajax-INVALID:~/Desktop/P-RPC$ g++ -o client -fopenmp client.cpp
HPC-LAB@ajax-INVALID:~/Desktop/P-RPC$ ./client ajax-INVALID 8080

[Status] Connected
Welcome : enter your nick :
Sherlock
msg from Dr.Watson: How is Quarantine going sherlock?
/msg Dr.Watson Boringggggggggg!!
```

3. **Client (2) receiving message from Client (1) and replying back .**


HPC-LAB@ajax-INVALID: ~/Desktop/P-RPC
File Edit View Search Terminal Help

```
HPC-LAB@ajax-INVALID:~/Desktop/P-RPC$ g++ -o client -fopenmp client.cpp
HPC-LAB@ajax-INVALID:~/Desktop/P-RPC$ ./client ajax-INVALID 8080

[Status] Connected
Welcome : enter your nick :
Dr.Watson
/msg Sherlock How is Quarantine going sherlock?
msg from Sherlock: Boringggggggggg!!
```

Conclusion: We have successfully implemented the Client Server Program (RPC) through parallel processing.