2012

Universidad Carlos III de Madrid

Antón García Dosil
100277382

[LAB ASSIGNMENT 3: REMOTE PROCEDURE CALLS (RPC)]

Implementation of an RPC service using rpcgen. Service provider is developed in C. A C client is developed to test the service provider.

Table of Contents

1. Assignment Overview

In the third assignment for Distributed Systems, an RPC service must be implemented. The requirements for each service are the same as for previous assignments.

The service provider must implement:

- Ping service: Answer a ping request.

- Swap service: The server will receive as input a text from a client. This service swaps high case to lower case and vice versa, returning swapped text as well as an integer specifying the number of swapped letters. Non alphabetic characters will remain the same.

- Hash service: The server will receive as input a text from a client. This service calculates a hash value for the determined text with the function:

  $$hash = (hash + char)\%\ 1000000000$$

  The hash obtained is returned to the client as an integer value.

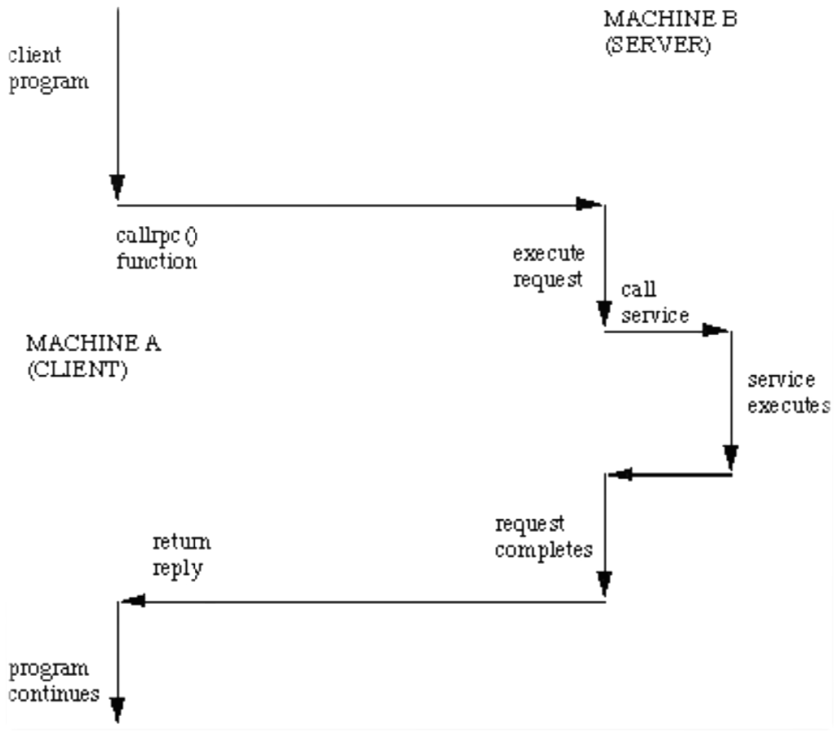
- Check service: The server will receive as input a text from a client as well as an integer hash value. The server creates the hash value from the text using the same function as in the hash service. Once the hash is obtained, it is compared with the input hash and OK will be outputted if correct and FAIL if it is incorrect.

- Stat service: No input values are required. Stat service outputs number of times each procedure has been called.

A file may have any size.

2. Program General Design

RPC as its name indicates allows calling functions on a remote system. There is a server that has all the remote procedures and there may be one or more clients that may call these remote procedures as if it were a local function. Each method has its own parameters and return value defined in an interface, common to both server and client.

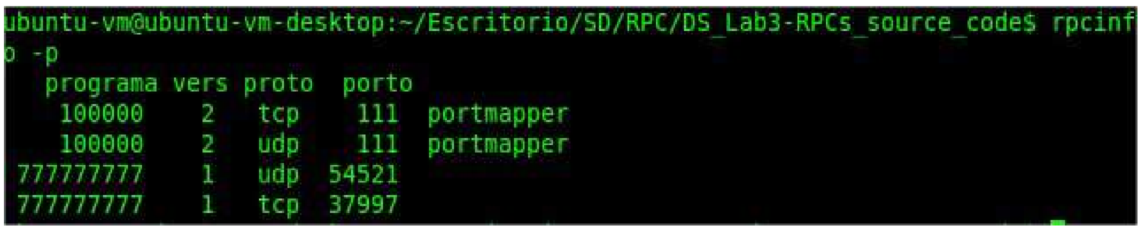


http://www.cs.cf.ac.uk/Dave/C/rpc.gif

This functionality is achieved by generating the client and server stubs using rpcgen. The stubs give us the feeling of locality due to their conversion of parameters on a RPC call.
The main functions of the stubs are:

- Find the remote system
- Transform parameters to send them in one unique message.
- Send the remote system the encapsulated parameters and wait for the response.

Client connection may be made either in UDP or in TCP.


```
ubuntu-vm@ubuntu-vm-desktop:~/Escritorio/SD/RPC/DS_Lab3-RPCs_source_code$ rpcinf
o -p
   programa vers proto   porto
    100000    2   tcp    111  portmapper
    100000    2   udp    111  portmapper
 777777777    1   udp  54521
 777777777    1   tcp  37997
```
Assignment server running. - program 777777777; udp port- 54521; tcp port-37997;

In this case I have chosen TCP to ensure client connection, preventing possible data corruption caused by UDP transport protocol.

3. Method Implementation

3.1Pingproc

3.1.1 Server

Receives client IP and connection port. Prints both and returns 0 to client.

3.1.2 Client

Obtains its ip and connection port and stores it in the request structure.

Calls the remote procedure

Receives response

3.2 Swapproc

3.2.1 Server

Receives client IP and port, a text fragment, the file size, an integer indicating whether the request shall print to server terminal, an integer indicating the corresponding text part and in case of more than one part swap, the accumulated number of swapped letters.

Swaps upper case to lower case using tolower function and swaps to lower case using toupper function. Each time a letter is swapped a counter is incremented. The swapped counter is added to the accumulated number of swaps (input). In case of only being one request, the accumulated number of swapped letters will be the same as the total number of swapped letters.
Once the server has finished swapping the characters, it returns the swapped text and the number of swapped letters to the client.

3.2.2 Client

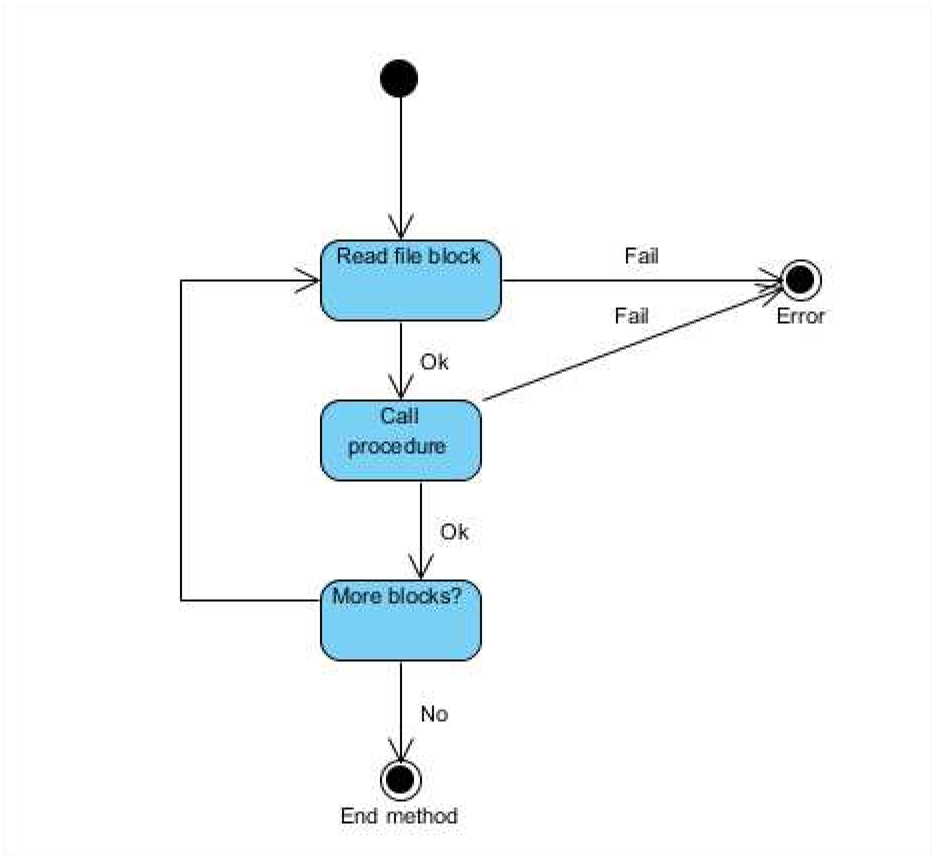
Obtains it's IP and connection port and stores it in the request structure. It opens the source file given through parameters and determines its file size. The file size is stored in the request structure as well.

If the file is greater than 1KB, only 1KB will be taken. This chunk will be stored in the request structure to be swapped. The length of the text must be stored as well as we are using an opaque variable. Each time a chunk is taken, a part counter is incremented. This counter is stored as well in the request structure.

The remote swap procedure is called.

The client obtains the swapped text and stores it in a file given by command line argument. The client takes the number of swapped letters and stores it in a local variable.

The client checks if there are any chunks left to obtain the final result. If there are more chunks it will repeat the steps above and if there aren't it will close its file descriptors and print the final swap value stored in the local variable.



3.3 Hashproc

3.3.1 Server

Receives client IP and port, a text fragment, the file size, an integer indicating whether the request shall print to server terminal, an integer indicating the corresponding text part and, in case of more than one part hash, the accumulated hash value.
Calculates the hash value from the input text fragment and calculates the total hash value with the previous hash value.

$$outputHash = (previousHash + calculatedHash)\%1000000000$$

The calculated hash is returned to the client.

3.3.2 Client

Obtains it's IP and connection port and stores it in the request structure. It opens the source file given through parameters and determines its file size. The file size is stored in the request structure as well.

If the file is greater than 1KB, only 1KB will be taken. This chunk will be stored in the request structure obtain its hash value. The length of the text must be stored as well as we are using an opaque variable. Each time a chunk is taken, a part counter is incremented. This counter is stored as well in the request structure.

The remote hash procedure is called.

The client obtains the hashed text value and stores it in a local variable. If there are more chunks it will repeat the steps above and if there aren't it will close its file descriptors and print the final hash value stored in the local variable. (As seen in swap image).

3.4 Checkproc

3.4.1 Server

Receives client IP and port, a text fragment, the file size, an integer indicating whether the request shall print to server terminal, an integer indicating the corresponding text part and, in case of more than one part hash, the accumulated hash value. In this case, an additional hash value is given by the client to verify it.
First, the server calculates the hash value as done in hashproc.

Once the hash value is obtained, it is compared with the hash value the client has provided. If both hashes match 1 is returned to the client, otherwise, 0 (both sent in a single byte).

3.4.2 Client

The client does the same operations as for Hashproc but also stores the hash value entered by the user through keyboard to the request.

Once all chunks are received, the client receives the procedure's verdict, receiving 1 if the hash inserted by keyboard is correct or 0 if it is not.

3.5 Statproc

3.5.1 Server

Each time a procedure finishes, it is logged to an auxiliary stat structure on the server. When stat is called, this structure is accessed.
Receives client IP and port for printing.
Saves the auxiliary stat structure to the output and returns it to the client.

3.5.2 Client

Obtains IP and port and saves them to the request.

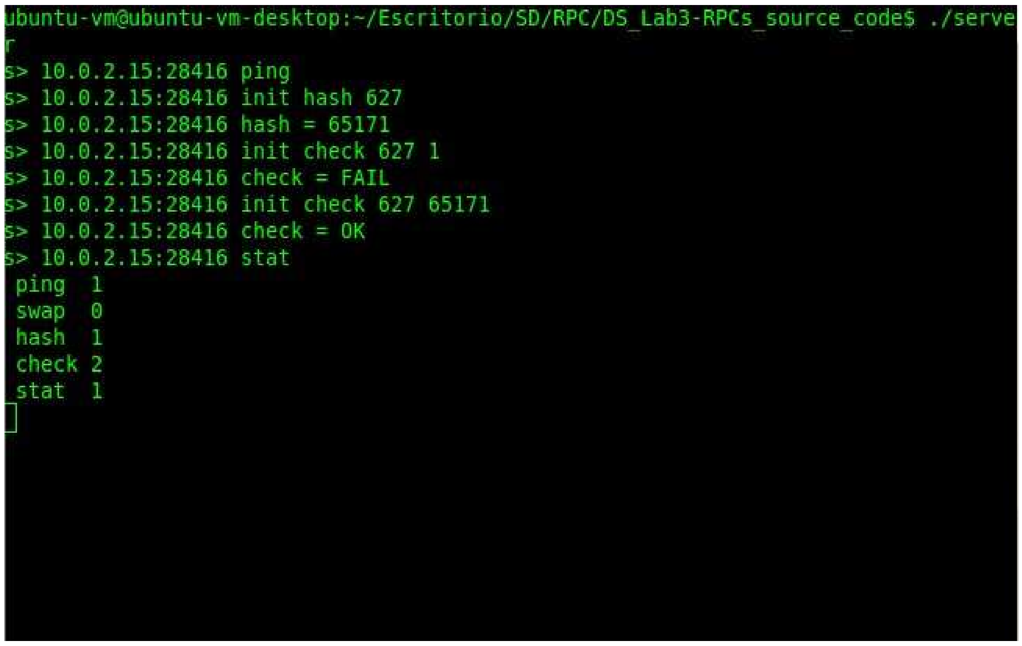
Calls the remote procedure.

Receives structure with all stat results.

4. Tests Carried out

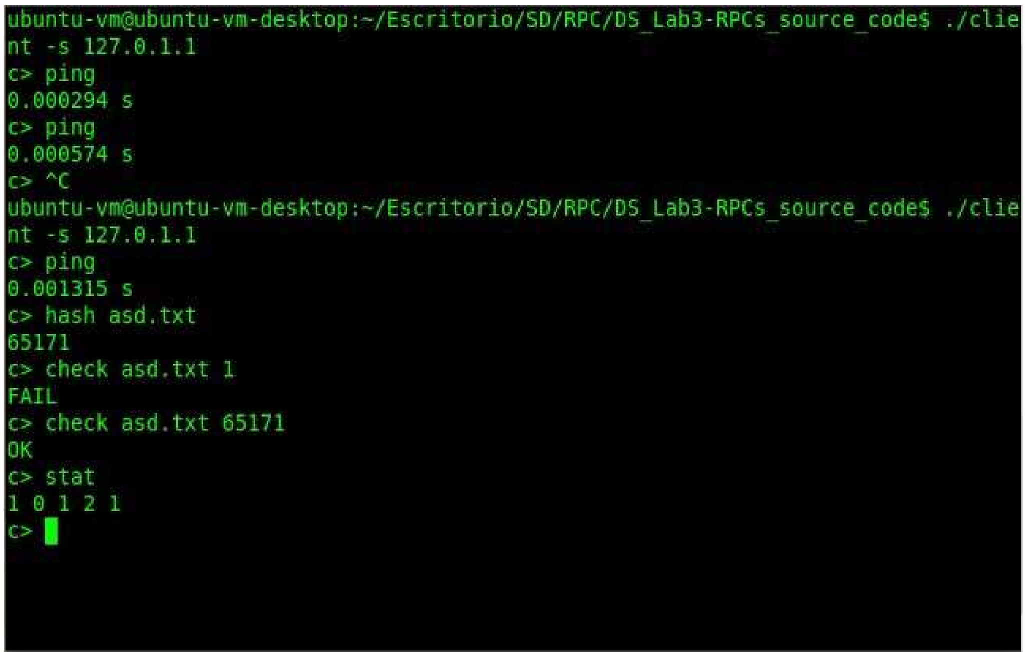
Terminal Output correctness tests (For same client input)

Server



```
ubuntu-vm@ubuntu-vm-desktop:~/Escritorio/SD/RPC/DS_Lab3-RPCs_source_code$ ./serve
r
s> 10.0.2.15:28416 ping
s> 10.0.2.15:28416 init hash 627
s> 10.0.2.15:28416 hash = 65171
s> 10.0.2.15:28416 init check 627 1
s> 10.0.2.15:28416 check = FAIL
s> 10.0.2.15:28416 init check 627 65171
s> 10.0.2.15:28416 check = OK
s> 10.0.2.15:28416 stat
 ping  1
 swap  0
 hash  1
 check 2
 stat  1
```

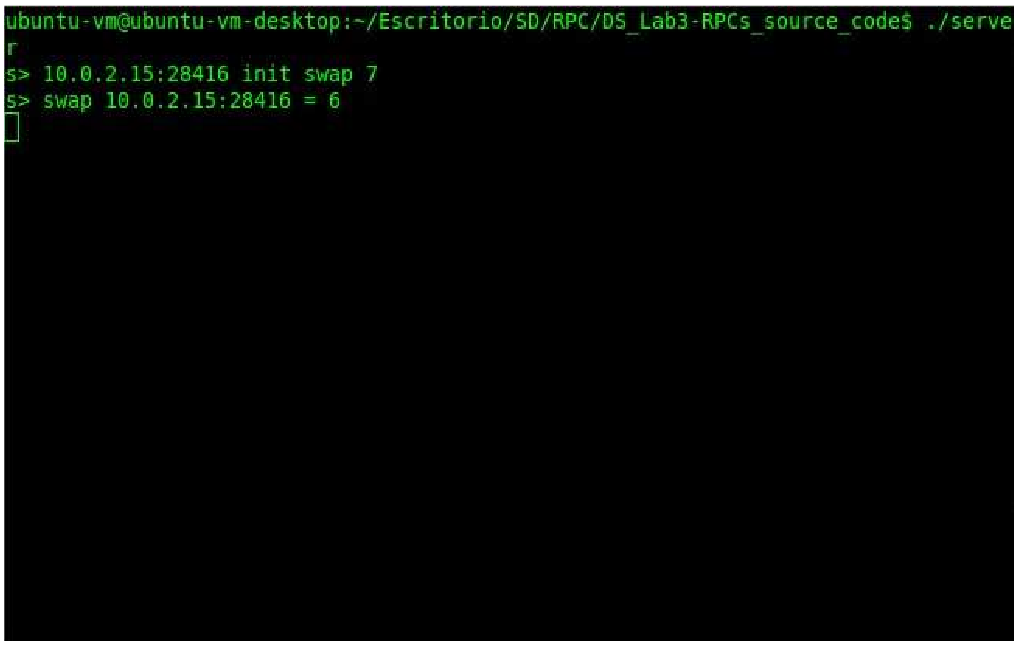
Client



```
ubuntu-vm@ubuntu-vm-desktop:~/Escritorio/SD/RPC/DS_Lab3-RPCs_source_code$ ./clie
nt -s 127.0.1.1
c> ping
0.000294 s
c> ping
0.000574 s
c> ^C
ubuntu-vm@ubuntu-vm-desktop:~/Escritorio/SD/RPC/DS_Lab3-RPCs_source_code$ ./clie
nt -s 127.0.1.1
c> ping
0.001315 s
c> hash asd.txt
65171
c> check asd.txt 1
FAIL
c> check asd.txt 65171
OK
c> stat
1 0 1 2 1
c>
```

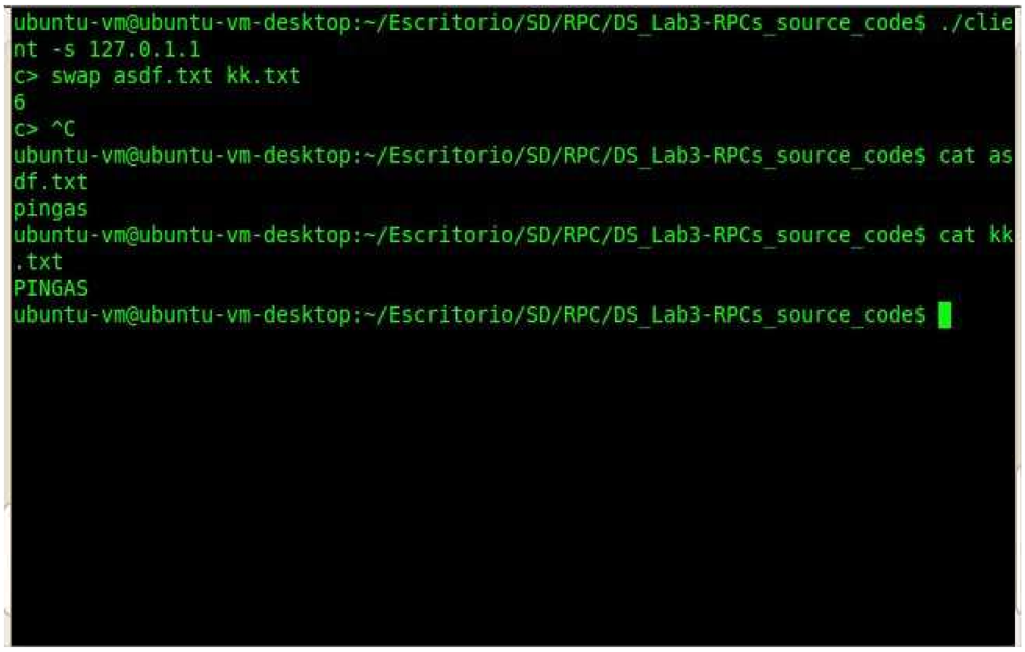
File output correctness (swap)

Server



```
ubuntu-vm@ubuntu-vm-desktop:~/Escritorio/SD/RPC/DS_Lab3-RPCs_source_code$ ./serve
r
s> 10.0.2.15:28416 init swap 7
s> swap 10.0.2.15:28416 = 6
```

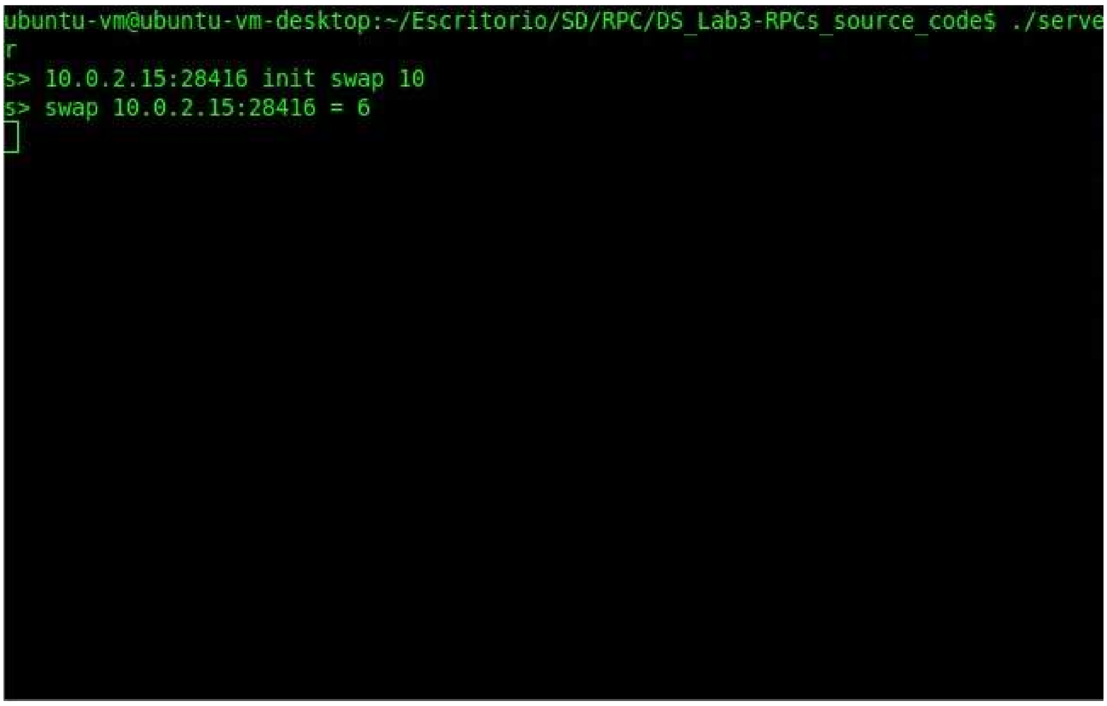
Client + File



```
ubuntu-vm@ubuntu-vm-desktop:~/Escritorio/SD/RPC/DS_Lab3-RPCs_source_code$ ./clie
nt -s 127.0.1.1
c> swap asdf.txt kk.txt
6
c> ^C
ubuntu-vm@ubuntu-vm-desktop:~/Escritorio/SD/RPC/DS_Lab3-RPCs_source_code$ cat as
df.txt
pingas
ubuntu-vm@ubuntu-vm-desktop:~/Escritorio/SD/RPC/DS_Lab3-RPCs_source_code$ cat kk
.txt
PINGAS
ubuntu-vm@ubuntu-vm-desktop:~/Escritorio/SD/RPC/DS_Lab3-RPCs_source_code$
```

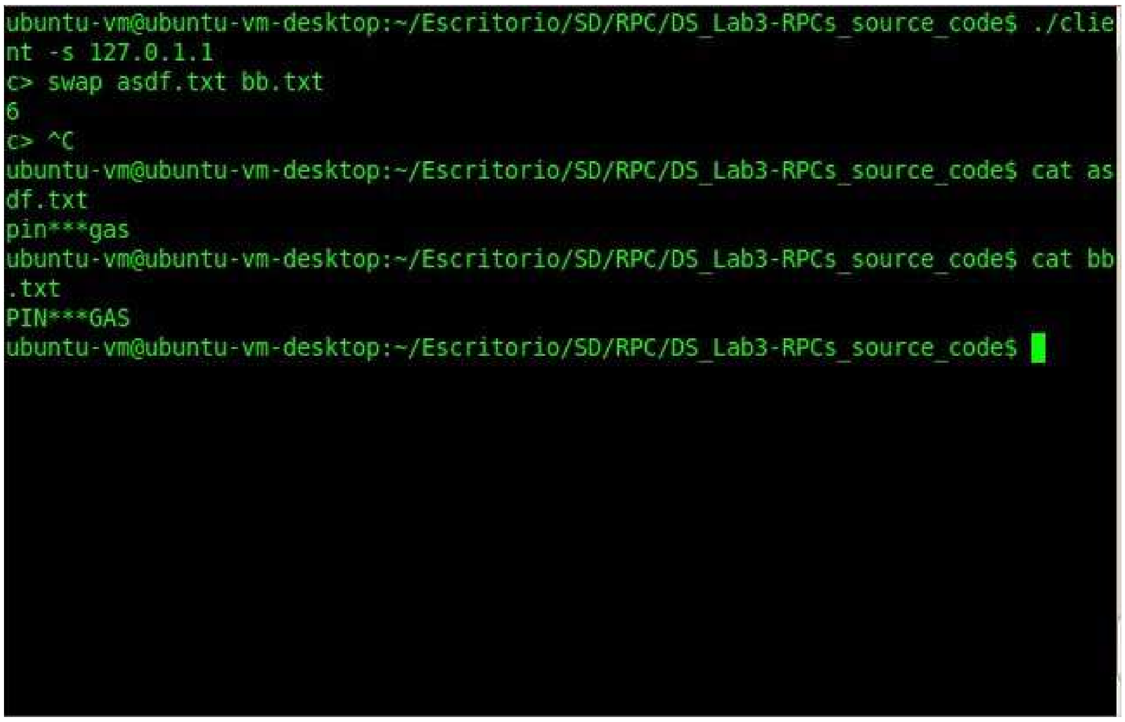
File output correctness (swap – non swappable chars)

Server



```
ubuntu-vm@ubuntu-vm-desktop:~/Escritorio/SD/RPC/DS_Lab3-RPCs_source_code$ ./serve
r
s> 10.0.2.15:28416 init swap 10
s> swap 10.0.2.15:28416 = 6
```

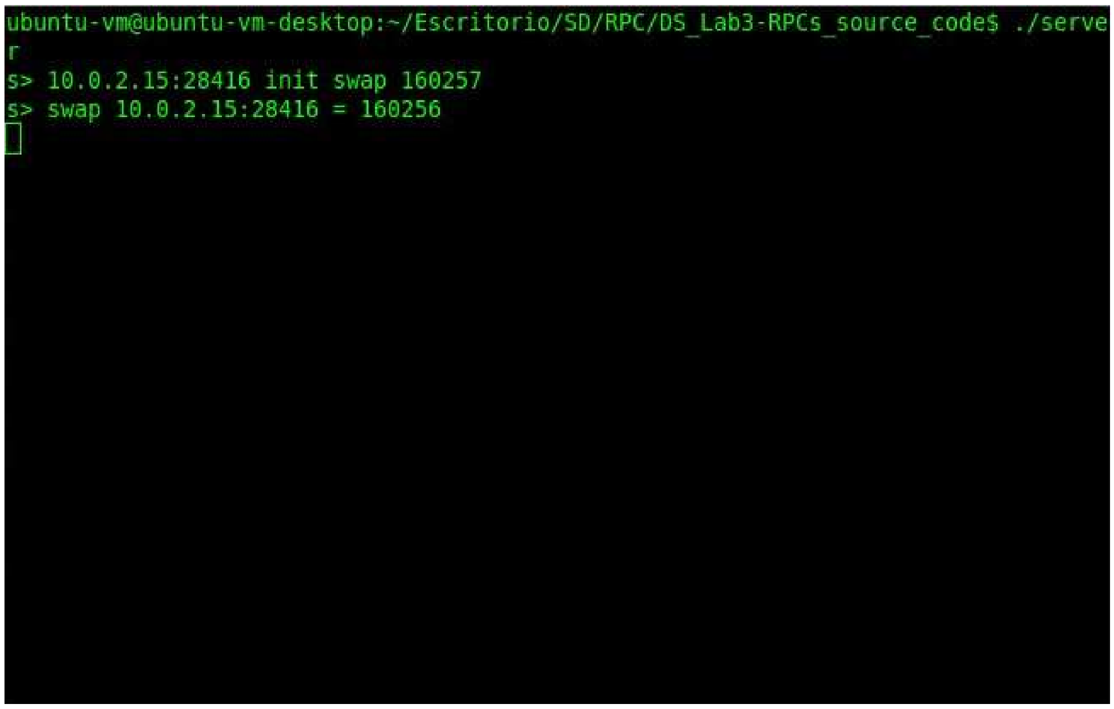
Client + File



```
ubuntu-vm@ubuntu-vm-desktop:~/Escritorio/SD/RPC/DS_Lab3-RPCs_source_code$ ./clie
nt -s 127.0.1.1
c> swap asdf.txt bb.txt
6
c> ^C
ubuntu-vm@ubuntu-vm-desktop:~/Escritorio/SD/RPC/DS_Lab3-RPCs_source_code$ cat as
df.txt
pin***gas
ubuntu-vm@ubuntu-vm-desktop:~/Escritorio/SD/RPC/DS_Lab3-RPCs_source_code$ cat bb
.txt
PIN***GAS
ubuntu-vm@ubuntu-vm-desktop:~/Escritorio/SD/RPC/DS_Lab3-RPCs_source_code$
```

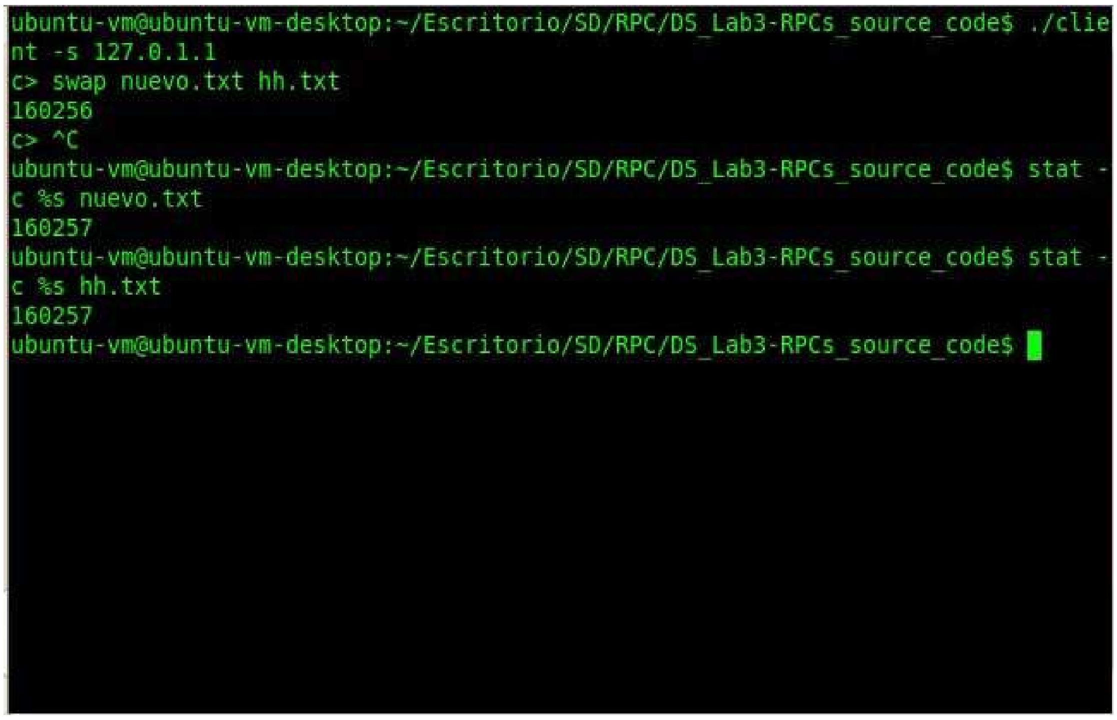
File output correctness (swap – big file)

Server



```
ubuntu-vm@ubuntu-vm-desktop:~/Escritorio/SD/RPC/DS_Lab3-RPCs_source_code$ ./serve
r
s> 10.0.2.15:28416 init swap 160257
s> swap 10.0.2.15:28416 = 160256
```

Client + Output File sizes



```
ubuntu-vm@ubuntu-vm-desktop:~/Escritorio/SD/RPC/DS_Lab3-RPCs_source_code$ ./clie
nt -s 127.0.1.1
c> swap nuevo.txt hh.txt
160256
c> ^C
ubuntu-vm@ubuntu-vm-desktop:~/Escritorio/SD/RPC/DS_Lab3-RPCs_source_code$ stat -
c %s nuevo.txt
160257
ubuntu-vm@ubuntu-vm-desktop:~/Escritorio/SD/RPC/DS_Lab3-RPCs_source_code$ stat -
c %s hh.txt
160257
ubuntu-vm@ubuntu-vm-desktop:~/Escritorio/SD/RPC/DS_Lab3-RPCs_source_code$
```

5. Student Conclusions

The completion of this practice resulted simpler than the previous ones because most of the method logic was already done and there was no need to manually transform data.

The main difficulty was using rpcgen to generate the stubs from the text.x file, specifically generating the XDR necessary for method parameters and return structures. The problem was the difference between XDR and C types. I had a big stall due to difference between opaque<> and string, but the course slides and playing a little with the code helped a lot.

Use of portmap and rpcinfo was useful for knowing the state of the server.

The use of structures was done to make the design of the application easier to understand for an object-programmed brain as my own. I believe that parameter encapsulation in structures should be done for everything in exception of a very trivial service. This encapsulation creates cleaner and more intuitive code as each service has a request structure and a response structure.

I believe that the practice was worth it because it enhanced my knowledge on RPC theory concepts on XDR and stubs that were not so clear before the completion of the assignment.

6. Tasks done and time required

	Done	Tested	Time
Remote Procedure Calls (RPC)	-	-	**14h**
Develop the ping service	✓	✓	1h
Communicate client and server using the ping service	✓	✓	1h
Develop swap service	✓	✓	1h
Communicate client and server using the swap service	✓	✓	4h
Develop the hash service	✓	✓	1h
Communicate client and server using the hash service	✓	✓	1h
Develop check service	✓	✓	30min
Communicate client and server using the check service	✓	✓	30min
Develop stat service	✓	✓	30min
Communicate client and server using the stat service	✓	✓	30min
Report	✓	-	3h