

# Distributed Systems Report

## Practical Work 2: RPC File Transfer

Nguyen Minh Cuong - 23BI14467

December 4, 2025

## 1 Introduction

The objective of this laboratory work is to upgrade the previous TCP file transfer system to use **Remote Procedure Calls (RPC)**. We utilized **ONC RPC (Sun RPC)** to abstract the networking layer, allowing the client to invoke a function on the server as if it were a local call. The system was implemented in C on the Kali Linux environment.

## 2 System Design

### 2.1 Protocol Design (IDL)

Unlike Lab 1 (Socket), we defined the protocol using an Interface Description Language (IDL) file named `transfer.x`.

- **Data Structure:** We defined a struct `file_data` containing:
  - `filename`: A string (max 256 chars) to identify the file.
  - `content`: An opaque type (variable-length bytes) to hold binary data.
  - `bytes_sent`: Integer indicating the file size.
- **Procedure:** The function `UPLOAD_FILE` takes `file_data` as input and returns an integer (1 for success).

### 2.2 System Organization

The architecture relies on the Stub/Skeleton model generated by `rpcgen`.

[Client Code] → [Client Stub] → (Network/XDR) → [Server Skeleton] → [Server Code]

## 3 Implementation

### 3.1 RPC Definition (`transfer.x`)

```

1 struct file_data {
2     string filename<256>;
3     opaque content<>;
4     int bytes_sent;
5 };
6
7 program TRANSFER_PROG {
8     version TRANSFER_VERS {
9         int UPLOAD_FILE(file_data) = 1;
10    } = 1;
11 } = 0x31230000;

```

Listing 1: The IDL file content

### 3.2 Client Logic (transfer\_client.c)

The client reads the file into a buffer and calls the remote function.

```

1 // 1. Read file to buffer
2 file_buffer = (char *)malloc(file_size);
3 fread(file_buffer, 1, file_size, fp);
4
5 // 2. Pack data into RPC struct
6 args.filename = filename;
7 args.content.content_val = file_buffer;
8 args.content.content_len = file_size;
9
10 // 3. Call Remote Procedure
11 result = upload_file_1(&args, clnt);

```

Listing 2: Client packing data

### 3.3 Server Logic (transfer\_server.c)

The server receives the data structure and writes it to the disk.

```

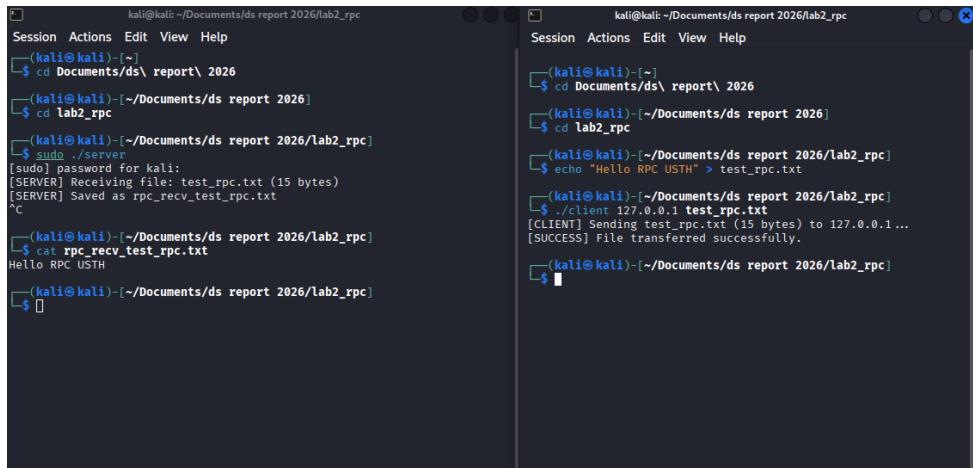
1 int * upload_file_1_svc(file_data *argp, struct svc_req *rqstp) {
2     static int result;
3     char new_name[300];
4
5     sprintf(new_name, "rpc_recv_%s", argp->filename);
6
7     FILE *fp = fopen(new_name, "wb");
8     fwrite(argp->content.content_val, 1, argp->content.content_len, fp);
9     fclose(fp);
10
11     result = 1;
12     return &result;
13 }

```

Listing 3: Server writing file

## 4 Results

The system successfully transferred a text file named `test_rpc.txt`. The server received it as `rpc_recv_test_rpc.txt`.



The image shows two terminal windows side-by-side, both titled "Session Actions Edit View Help".

**Terminal 1 (Left):**

```
kali@kali: ~/Documents/ds report 2026/lab2_rpc
Session Actions Edit View Help
[(kali㉿kali)-~]
$ cd Documents/ds\ report\ 2026
[(kali㉿kali)-~] 
$ cd lab2_rpc
[(kali㉿kali)-~/Documents/ds report 2026/lab2_rpc]
$ sudo ./server
[sudo] password for kali:
[SERVER] Receiving file: test_rpc.txt (15 bytes)
[SERVER] Saved as rpc_recv_test_rpc.txt
^C
[(kali㉿kali)-~/Documents/ds report 2026/lab2_rpc]
$ cat rpc_recv_test_rpc.txt
Hello RPC USTH
[(kali㉿kali)-~/Documents/ds report 2026/lab2_rpc]
$
```

**Terminal 2 (Right):**

```
kali@kali: ~/Documents/ds report 2026/lab2_rpc
Session Actions Edit View Help
[(kali㉿kali)-~]
$ cd Documents/ds\ report\ 2026
[(kali㉿kali)-~] 
$ cd lab2_rpc
[(kali㉿kali)-~/Documents/ds report 2026/lab2_rpc]
$ echo "Hello RPC USTH" > test_rpc.txt
[(kali㉿kali)-~/Documents/ds report 2026/lab2_rpc]
$ ./client 127.0.0.1 test_rpc.txt
[CLIENT] Sending test_rpc.txt (15 bytes) to 127.0.0.1 ...
[SUCCESS] File transferred successfully.
[(kali㉿kali)-~/Documents/ds report 2026/lab2_rpc]
$
```

Figure 1: Screenshot of successful transfer on Kali Linux