

Distributed System Labwork 2



Group 1 - ICT

University of Science and Technology of Hanoi

January, 2022

Contents

1	Introduction	2
1.1	Overview	2
1.2	Protocol	2
1.3	System organization	3
1.4	Implementation	3
1.5	Contribution	6

1 Introduction

1.1 Overview

In this labwork, we try to build a file transfer using RPC. We will use C in this labwork.

1.2 Protocol

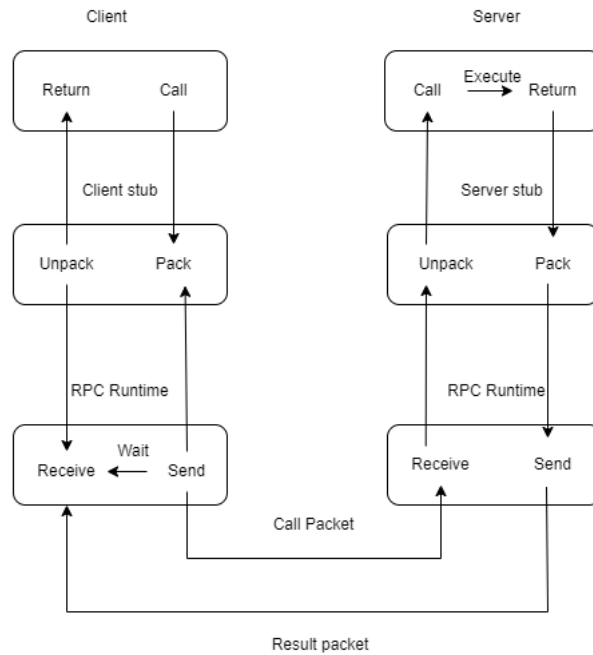


Figure 1: Protocol diagram

- The client calls the client stub. The call is a local procedure call
- The client stub packs the parameters into a message and makes a system call to send the message
- The client's local operating system sends the message from the client machine to the server machine
- The local operating system on the server machine passes the incoming packets to the server stub
- The server stub unpacks the parameters from the message.
- Finally, the server stub calls the server procedure.

1.3 System organization

The client and the server connects through UDP.

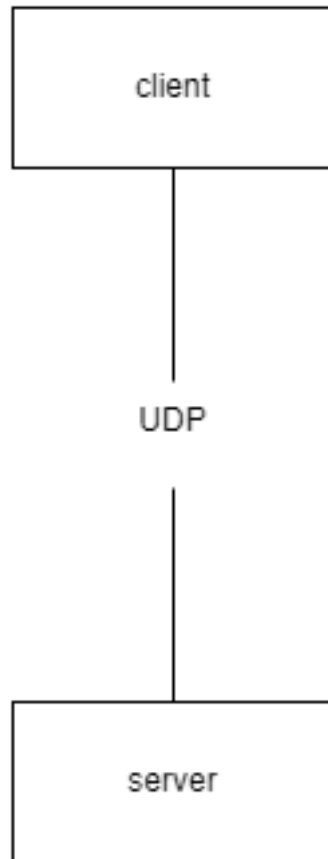


Figure 2: System organization

1.4 Implementation

At first, we created a file a named it file.x. The code in the file was implemented below:

```
const MAX_SIZE = 1024;
struct file{
    char filename[MAX_SIZE];
    char buffer[MAX_SIZE];
    int buffer_size;
};

program FILESTRANSFER {
```

```

        version FILESTRANSFER_1 {
            int transfer_file(file)=1;
        } = 1;
    } = 0x31230000;

```

Client and server stubs were generated when we typed "rpcgen -a -C file.x".
We have implemented the client side:

```

1  #include "file.h"
2  #include <time.h>
3
4  void filetransfer_1(char* host, char* filename)
5  {
6      CLIENT* clnt;
7      FILE* fp;
8      int* result_1;
9      file transfer_file_1_arg;
10     char ch;
11
12     #ifndef DEBUG
13         clnt = clnt_create(host, FILESTRANSFER, FILESTRANSFER_1, "udp");
14         if (clnt == NULL) {
15             clnt_pcreateerror(host);
16             exit(1);
17         }
18     #endif /* DEBUG */
19     clock_t begin = clock();
20     strcpy(transfer_file_1_arg.filename, filename);
21     fp = fopen("send.txt", "r");
22     if (fp == NULL) {
23         perror("Error in reading file..\n");
24         exit(1);
25     }
26     else {
27         printf("Reading file successfully..\n");
28     }
29     memset(transfer_file_1_arg.buffer, 0, sizeof(transfer_file_1_arg.buffer));
30     transfer_file_1_arg.buffer_size = 0;
31     while(1){
32         transfer_file_1_arg.buffer_size = fread(transfer_file_1_arg.buffer, 1, 1024,
33         ↪ fp);
34
35         result_1 = transfer_file_1(&transfer_file_1_arg, clnt);
36         if (result_1 == (int*)NULL) {
37             clnt_perror(clnt, "call failed");
38         }
39         if(transfer_file_1_arg.buffer_size < 1024) {
40             printf("\nUpload finished.\n");
41             break;

```

```

41     }
42 }
43 clock_t end = clock();
44 double upload_time = (double)(end - begin) / CLOCKS_PER_SEC;
45
46 printf("Upload time: %lf\n", upload_time);
47 printf("The file has been successfully sent..\n");
48
49
50
51 #ifndef DEBUG
52     clnt_destroy(clnt);
53 #endif /* DEBUG */
54     fclose(fp);
55 }
56
57
58 int main(int argc, char* argv[])
59 {
60     char* host;
61     char* filename;
62
63     if (argc < 3) {
64         printf("usage: %s server_host file_name\n", argv[0]);
65         exit(1);
66     }
67     host = argv[1];
68     filename = argv[2];
69     filestransfer_1(host, filename);
70     exit(0);
71 }

```

We have implemented the server side:

```

1  #include "file.h"
2
3  int* transfer_file_1_svc(file* argp, struct svc_req* rqstp)
4  {
5      static int result;
6      FILE* fp;
7      fp = fopen("received.txt", "w+");
8      if (fp == NULL) {
9          perror("Error in reading file..\n");
10         exit(1);
11     }
12     else {
13         printf("Reading file successfully..\n");
14     }
15 }

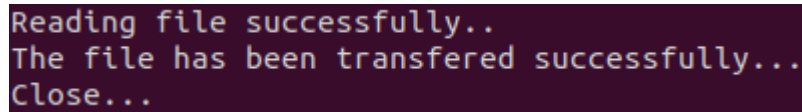
```

```

16  fwrite(argp->buffer, 1, argp->buffer_size, fp);
17  printf("%s \n", argp->buffer);
18  printf("Received file..\n");
19  fclose(fp);
20
21  return &result;
22 }

```

Here is the result:

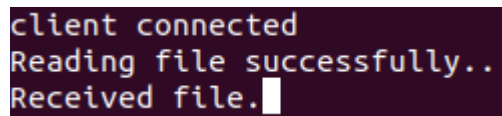


```

Reading file successfully..
The file has been transfered successfully..
Close...

```

Figure 3: Client side



```

client connected
Reading file successfully..
Received file.

```

Figure 4: Server side

1.5 Contribution

Member	Contribution
Nguyen Xuan Tung	Client code
Nguyen Quang Anh	Server code
Lu Khanh Huyen	Design Protocol
Tran Hong Quan	Design Architecture
Vu Duc Chinh	Report

Table 1: Contribution Table