University of Science and Technology of Hanoi

Distributed System

PRACTICAL WORK 2: RPC FILE TRANSFER

Group members:

Nguyen Phuong Thao (BI9-212)

Doan Tuyet Mai (BI9-162)

Trinh Thao Phuong (BI9-191)

Phung Kim Son (BI9-202)

Pham Minh Long (BI9-146)

Hanoi, Feb 2021

1. Protocol design

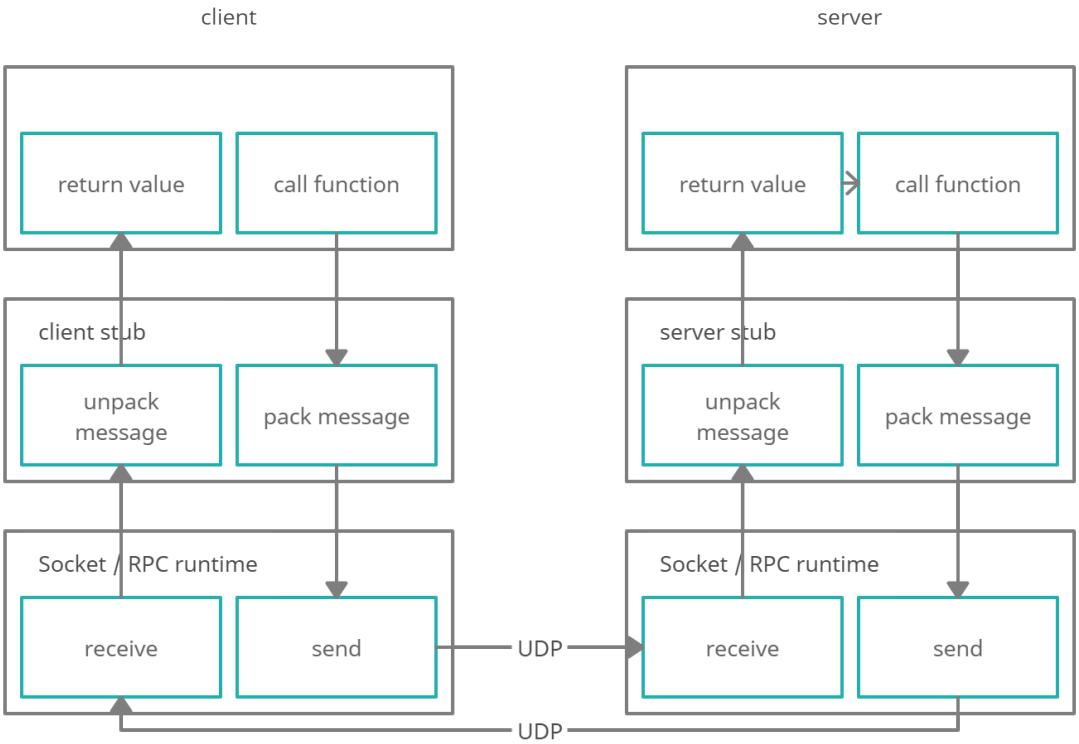

Figure 1: Protocol design

In RPC file transfer, the client makes a procedure call to send a data packet to the server. When the packet is received, the server executes functions to performs what the client requests, packs reply into a message and makes the procedure call to returns messages to the client.

2. System organization

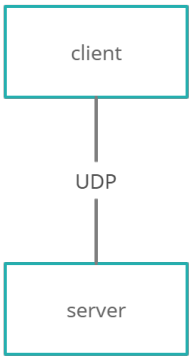

Figure 2: System organization

The server and client are connected together with the help of rpcgen through UDP.

3. File transfer implementation

Full code implementation could be found here, in the `code` directory.

Client and server stubs were generated by `rpcgen`, using an input file `transfer.x` which contains interface definitions:

```
#define SIZE 1024
typedef opaque filebytes[SIZE];

struct file {
   char name[SIZE];
   filebytes data;
   int nbytes;
};

program TRANSFER {
   version TRANSFER_1 {
      int TRANSF(file) = 1;
   } = 1;
} = 0x31230000;
```

Then, in the command line:

```
$ rpcgen -a -C transfer.x
```

After that, output files to implement an RPC protocol will be generated.

3.1 Client-side

File transfer code for client-side was mainly implemented in `transfer_client.c` file. Begin of the file is the include to the header file `transfer.h`

The client's RPC handle was created by `clnt_create()`, which took the name of the remote host `host`, the program number `TRANSFER` and the program version `TRANSFER_1` as parameters.

```
CLIENT *clnt;
clnt = clnt_create (host, TRANSFER, TRANSFER_1, "udp");
if (clnt == NULL) {
   clnt_pcreateerror (host);
   exit (1);
}
```

Next, to talk about the file transfer, let's call the file that will be sent `sample`. This file will be first opened by `fopen()` to get the buffered input stream.

```
fp = fopen(filetotransf, "rb");
if(fp == NULL) {
   printf("File not found.\n");
   exit(1);
}
```

If the file `sample` can't be opened, an alert will be printed.

After that, if the file size is larger than 1MB, it will be broken down into smaller parts, maximum size of each is 1MB, and then sent as arguments sequentially:

```
while(1) {
   transf_1_arg.nbytes = fread(transf_1_arg.data, 1, SIZE, fp);
   result_1 = transf_1(&transf_1_arg, clnt);

   if (result_1 == (int *) NULL) {
```

```
      clnt_perror (clnt, "call failed");
   }

   if(transf_1_arg.nbytes < SIZE) {
      printf("Completed file transfer.\n");
      break;
   }
}
```

3.2 Server-side

File transfer code for server-side was mainly implemented in `transfer_server.c` file. Begin of the file is the include to the header file `transfer.h`

The server's main function is `transf_1_svc`. First, we will check if we are receiving the file and ready to open the file.

```
if (strcmp(opened_file, "") == 0 && fp == NULL) {
   printf("Receiving new file %s.\n", argp->name);

   strcpy(opened_file, argp->name);
   fp = fopen("received_sample", "ab+");
}
```

Then, we will begin to flush stream, and write data from the file in the stream.

```
if (strcmp(opened_file, argp->name) == 0) {
      fflush(stdout);

      fwrite(argp->data, 1, argp->nbytes, fp);

      if (argp->nbytes < SIZE) {
         printf("\nFinished receiving %s.\n", argp->name);
         total = 0;
         fclose(fp);
         fp = NULL;
         strcpy(opened_file, "");
      }
   }
```

4. Contribution

Student	Student ID	Contribution
Pham Minh Long	BI9-146	Protocol design
Phung Kim Son	BI9-202	System organization
Trinh Thao Phuong	BI9-191	Draw figures, Brief explanation for protocol design
Doan Tuyet Mai	BI9-162	Code for file transfer Explanation for the implementation of Client-side
Nguyen Phuong Thao	BI9-212	Explanation for the implementation of Server-side