

CSE 532 Distributed Operating Systems

Remote Procedure Call

Batuhan Düzgün – 20163505004

1. Introduction

This example illustrates the basics of using Remote Procedure Calls. There will be a distributed calculator program which depends on Remote Procedure Call mechanism. This calculator program will have four basic functions which they are named as “Sum”, “Subtract”, “Divide” and “Multiply”. Client program will accept two numbers from the command line, parse the ASCII text to convert them to numbers, and will call a remote procedure call which can be one of four calculation functions. Server program will accept this request and will make calculation. After that, the server program will return a value to the client. This application will be developed in a Linux Operating System which is named as Ubuntu.

2. Remote Procedure Call Definition

RPC is a powerful technique for constructing distributed, client-server based applications. It is based on extending the notion of conventional, or 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. By using RPC, programmers of distributed applications avoid the details of the interface with the network. The transport independence of RPC isolates the application from the physical and logical elements of the data communications mechanism and allows the application to use a variety of transports.

3. IDL File

The first step involves defining the interface. This has to abide by Sun's format for its Interface Definition Language (IDL). An IDL is a file (suffixed with .x) which optionally begins with a bunch of type definitions and then defines the remote procedures. For calculator program, “calculator.x” IDL file is created.

```
struct numbers {  
    float first_number;  
    float second_number;  
};  
  
program CALCULATOR {  
    version CALCULATOR_FUNCTIONS {  
        float SUM(numbers) = 1;  
        float SUBTRACT(numbers) = 2;  
        float DIVIDE(numbers) = 3;  
        float MULTIPLY(numbers) = 4;  
    } = 1;  
} = 0x23451111;
```

In this example, there is one type definition to define a structure that holds two floats. Its name is “numbers” which is a C based data structure. This structure will be input parameters for RPC calls. Also, the interface have one version number and one program number. For each function, there is a procedure number. For this example:

Program Number : 0x23451111
Version Number : 1
Procedure Numbers : SUM = 1, SUBSTRACT = 2, DIVIDE = 3, MULTIPLY = 4

4. Generate RPC Stubs

“**rpcgen**” command is used to generate RPC stubs.

```
>> rpcgen -a -C calculator.x
```

The “-a” flag tells rpcgen to generate all files, the -C flag (upper-case C) tells rpcgen to generate C code that conforms to ANSI C.

5. “calculator.h” Header File

This is the header file that it will be usedn in both client and server code.

6. Server Program

There will be a server program which will accept RPC requests from clients. This server program will include four calculation functions.

Functions Signatures :

```
float * sum_1_svc(numbers *argp, struct svc_req *rqstp) : sums two float values.  
float * subtract_1_svc(numbers *argp, struct svc_req *rqstp) : subtracts two float values.  
float * divide_1_svc(numbers *argp, struct svc_req *rqstp) : divides two float values.  
float * multiply_1_svc(numbers *argp, struct svc_req *rqstp) : multiply two float values.
```

All of these functions accepts “number” C structure as an input.

Sample Sum Function :

```

#include "calculator.h"
#include <stdio.h>

float *
sum_1_svc(numbers *argp, struct svc_req *rqstp)
{
    static float result;

    float firstargument = 0.0f;
    float secondargument = 0.0f;

    firstargument = argp->first_number;
    secondargument = argp->second_number;

    result = firstargument + secondargument;

    printf("----- Add Calculation Information Log -----");
    printf("[ First Arg   :   %f ]", firstargument);
    printf("[ Second Arg  :   %f ]", secondargument);
    printf("[ Result      :   %f ]", result);
    printf("-----");

    return &result;
}

```

7. Client Program

There will be a client program which will make RPC calls to server program. It will call one of calculation functions as RPC call. First, We will create a connection via TCP. After that, It shows remote procedure call code blocks.

```

calculator_client = clnt_create(host, CALCULATOR, CALCULATOR_FUNCTIONS, "tcp");

switch(operation_type) {
    case 1: {
        operation_result = sum_1(&input_parameters_container, calculator_client);
        break;
    }
    case 2: {
        operation_result = subtract_1(&input_parameters_container, calculator_client);
        break;
    }
    case 3: {
        operation_result = divide_1(&input_parameters_container, calculator_client);
        break;
    }
    case 4: {
        operation_result = multiply_1(&input_parameters_container, calculator_client);
        break;
    }
}

```

8. Compile C Files

After, “rpcgen” command, C files are created. And also, There is a “Make File”. C files are compiled via this make file.

```
>> make -f Makefile.calculator
```

Command output from Linux Console :

```
cc -g -c -o calculator_clnt.o calculator_clnt.c
```

```
cc -g -c -o calculator_client.o calculator_client.c
```

```
cc -g -c -o calculator_xdr.o calculator_xdr.c
```

```
cc -g -o calculator_client calculator_clnt.o calculator_client.o calculator_xdr.o -lnsl
```

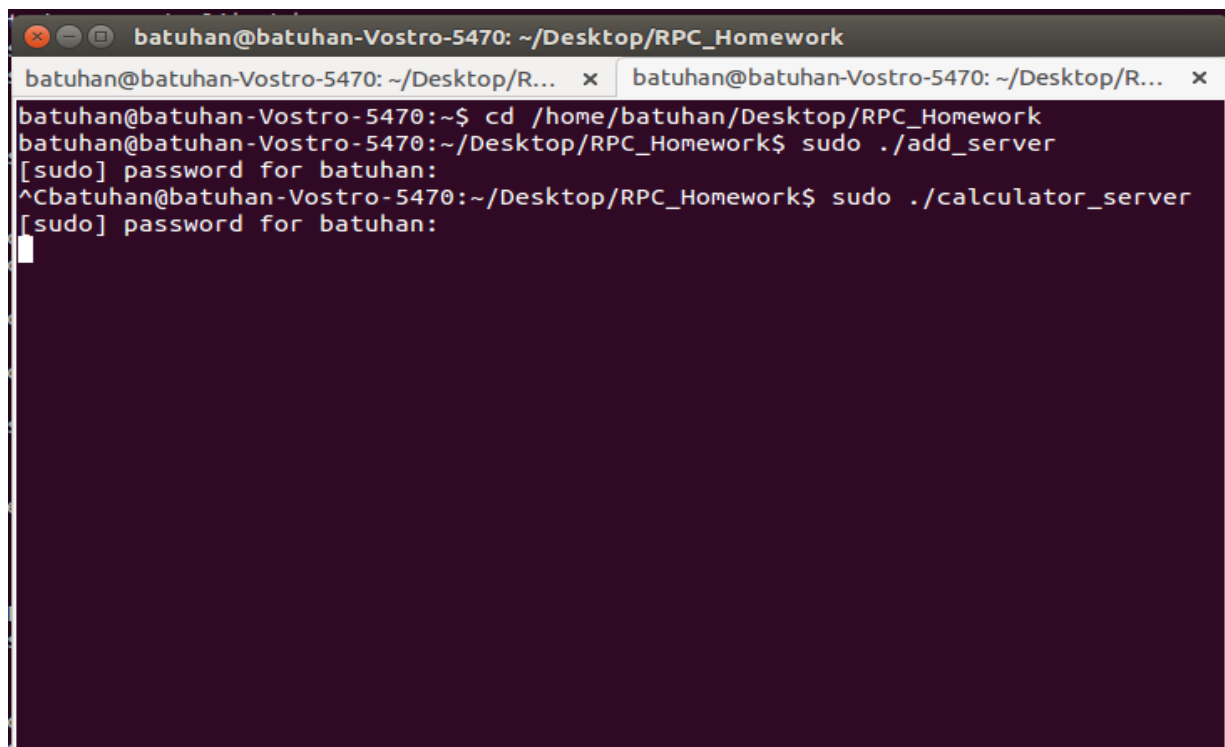
```
cc -g -c -o calculator_svc.o calculator_svc.c
```

```
cc -g -c -o calculator_server.o calculator_server.c
```

```
cc -g -o calculator_server calculator_svc.o calculator_server.o calculator_xdr.o -lnsl
```

The server and client stubs are created.

9. Run Server Program



```
batuhan@batuhan-Vostro-5470: ~/Desktop/RPC_Homework
batuhan@batuhan-Vostro-5470: ~/Desktop/R... x batuhan@batuhan-Vostro-5470: ~/Desktop/R... x
batuhan@batuhan-Vostro-5470:~$ cd /home/batuhan/Desktop/RPC_Homework
batuhan@batuhan-Vostro-5470:~/Desktop/RPC_Homework$ sudo ./add_server
[sudo] password for batuhan:
^Cbatuhan@batuhan-Vostro-5470:~/Desktop/RPC_Homework$ sudo ./calculator_server
[sudo] password for batuhan:
```

10. Run Client Program

```
batuhan@batuhan-Vostro-5470: ~/Desktop/RPC_Homework
batuhan@batuhan-Vostro-5470: ~/Desktop/R... x batuhan@batuhan-Vostro-5470: ~/Desktop/R... x
batuhan@batuhan-Vostro-5470:~/Desktop/RPC_Homework$ sudo ./calculator_client localhost 8.3 94.2 1
FIRST NUMBER : 8.300000
SECOND NUMBER : 94.199997
OPERATION TYPE : 1
Calculation result is : 102.500000
batuhan@batuhan-Vostro-5470:~/Desktop/RPC_Homework$ sudo ./calculator_client localhost 8.3 94.2 2
FIRST NUMBER : 8.300000
SECOND NUMBER : 94.199997
OPERATION TYPE : 2
Calculation result is : -85.899994
batuhan@batuhan-Vostro-5470:~/Desktop/RPC_Homework$ sudo ./calculator_client localhost 80.9 8.1 3
FIRST NUMBER : 80.900002
SECOND NUMBER : 8.100000
OPERATION TYPE : 3
Calculation result is : 0.000000
batuhan@batuhan-Vostro-5470:~/Desktop/RPC_Homework$ sudo ./calculator_client localhost 80.9 8.1 4
FIRST NUMBER : 80.900002
SECOND NUMBER : 8.100000
OPERATION TYPE : 4
Calculation result is : 655.290039
batuhan@batuhan-Vostro-5470:~/Desktop/RPC_Homework$
```

11. RPC Architecture for Calculator Project

