FR. CONCEICAO RODRIGUES COLLEGE OF ENGINEERING

Department of Computer Engineering

Course, Subject & Experiment Details

Practical No:	2
Title:	Remote Procedure Call
Name of the Student:	Warren Fernandes
Roll No:	8940
Date of Performance:	03/02/2023
Date of Submission:	10/02/2023

Evaluation:

Sr. No.	Rubric	Grade
1	Timeliness (1)	
2	Documentation (2)	
3	Preparation (3)	
4	Performance (4)	

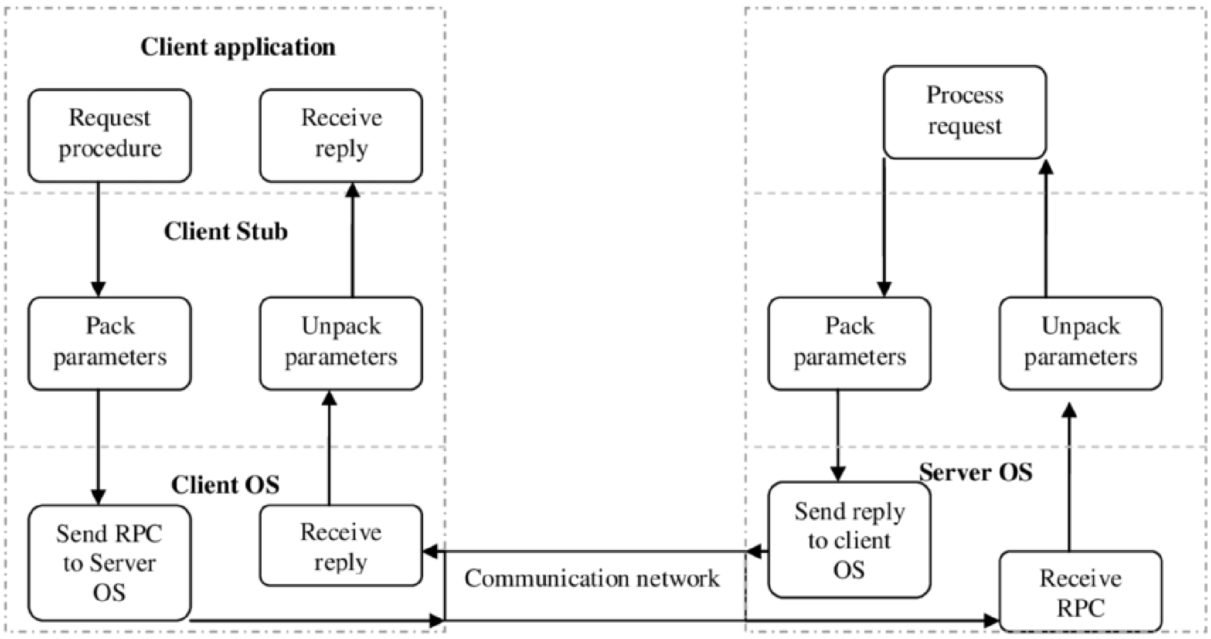
Signature of the Teacher

Remote Procedure Call

Remote Procedure Call (RPC) is a technology that enables a program to invoke a function or procedure in another address space, which could be running on a different machine or operating system, over a network. RPC is a client-server model in which the client initiates a request, and the server processes the request and returns the result.

The steps involved in the RPC mechanism are:

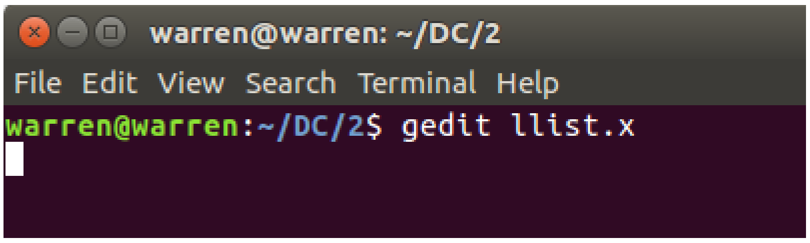
1. The client makes a function call to the stub procedure, which is a client-side program that acts as a proxy for the actual procedure residing on the server.
2. The stub procedure packages the arguments passed to the function call into a message and sends it over the network to the server.
3. The server receives the message and unpacks the arguments.
4. The server executes the actual procedure using the arguments and returns the result to the stub procedure.
5. The stub procedure receives the result from the server and returns it to the client.
6. The client receives the result and resumes its execution.



Aim: To design, implement, and test a client-server application using RPC in C, to demonstrate the principles of RPC and the use of standard RPC tools such as rpcgen, and to evaluate the scalability, robustness, and security of the application under different scenarios of load, network latency, and error conditions

Steps:

We can use gedit text editor to create a file named llist.x for RPC programming

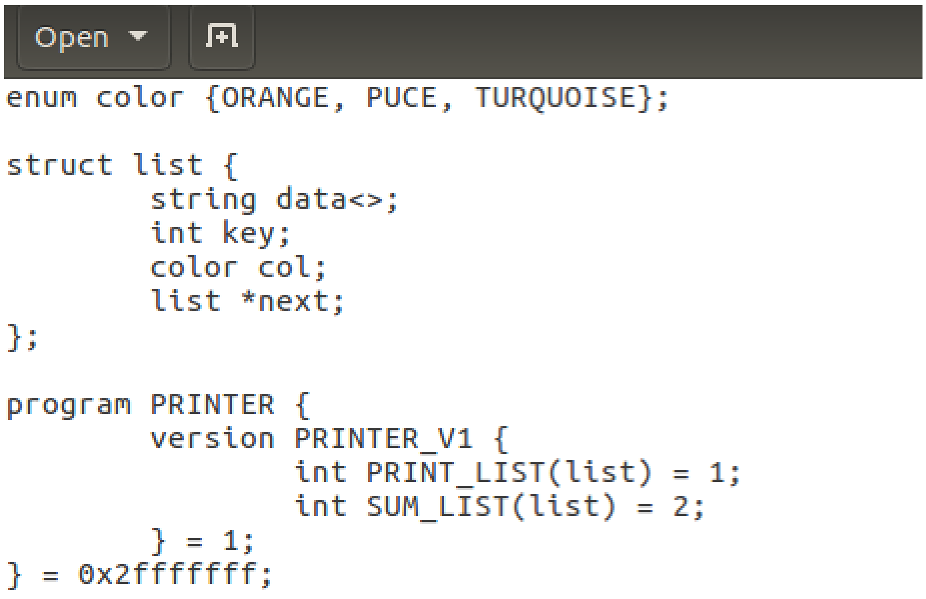


```
warren@warren:~/DC/2$ gedit llist.x
```

llist.x

To create a new Remote Procedure Call (RPC) program in C called PRINTER, with version PRINTER_V1, that can print a linked list and compute the sum of its elements, we can define the necessary data types and procedures using an Interface Definition Language (IDL) file. In this case, we also need to include an enum that defines three colors: orange, puce, and turquoise.

The IDL file looks like this:

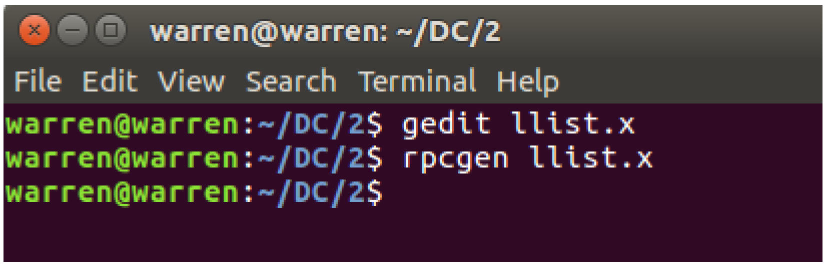


```
enum color {ORANGE, PUCE, TURQUOISE};

struct list {
        string data<>;
        int key;
        color col;
        list *next;
};

program PRINTER {
        version PRINTER_V1 {
                int PRINT_LIST(list) = 1;
                int SUM_LIST(list) = 2;
        } = 1;
} = 0x2fffffff;
```

Here, we define the enum color and the struct node that represents a single node of the linked list. We then define the list struct, which contains a pointer to the head of the list.

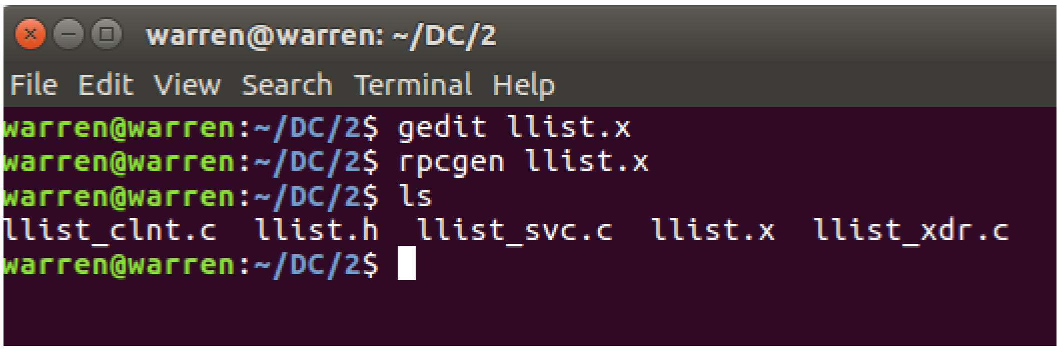
Next, we define the PRINTER program with version PRINTER_V1. This program has two procedures: PRINT_LIST and SUM_LIST. The PRINT_LIST procedure takes a list parameter and prints the contents of the linked list. The SUM_LIST procedure takes a list parameter and returns the sum of the elements in the linked list.

Once we have defined the IDL file, we can use the rpcgen tool to generate the C source code for the RPC stubs and skeletons. We can then implement the server-side code for the PRINTER program using the generated source code.



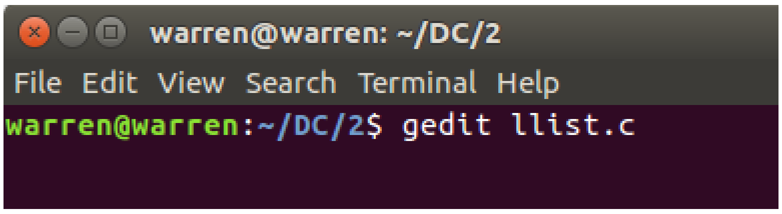
```
warren@warren:~/DC/2$ gedit llist.x
warren@warren:~/DC/2$ rpcgen llist.x
warren@warren:~/DC/2$
```

This will generate four files: llist.h, llist_clnt.c, llist_svc.c, and llist_xdr.c.



```
warren@warren:~/DC/2$ gedit llist.x
warren@warren:~/DC/2$ rpcgen llist.x
warren@warren:~/DC/2$ ls
llist_clnt.c  llist.h  llist_svc.c  llist.x  llist_xdr.c
warren@warren:~/DC/2$
```

We can use gedit text editor to create a file named llist.c for C programming



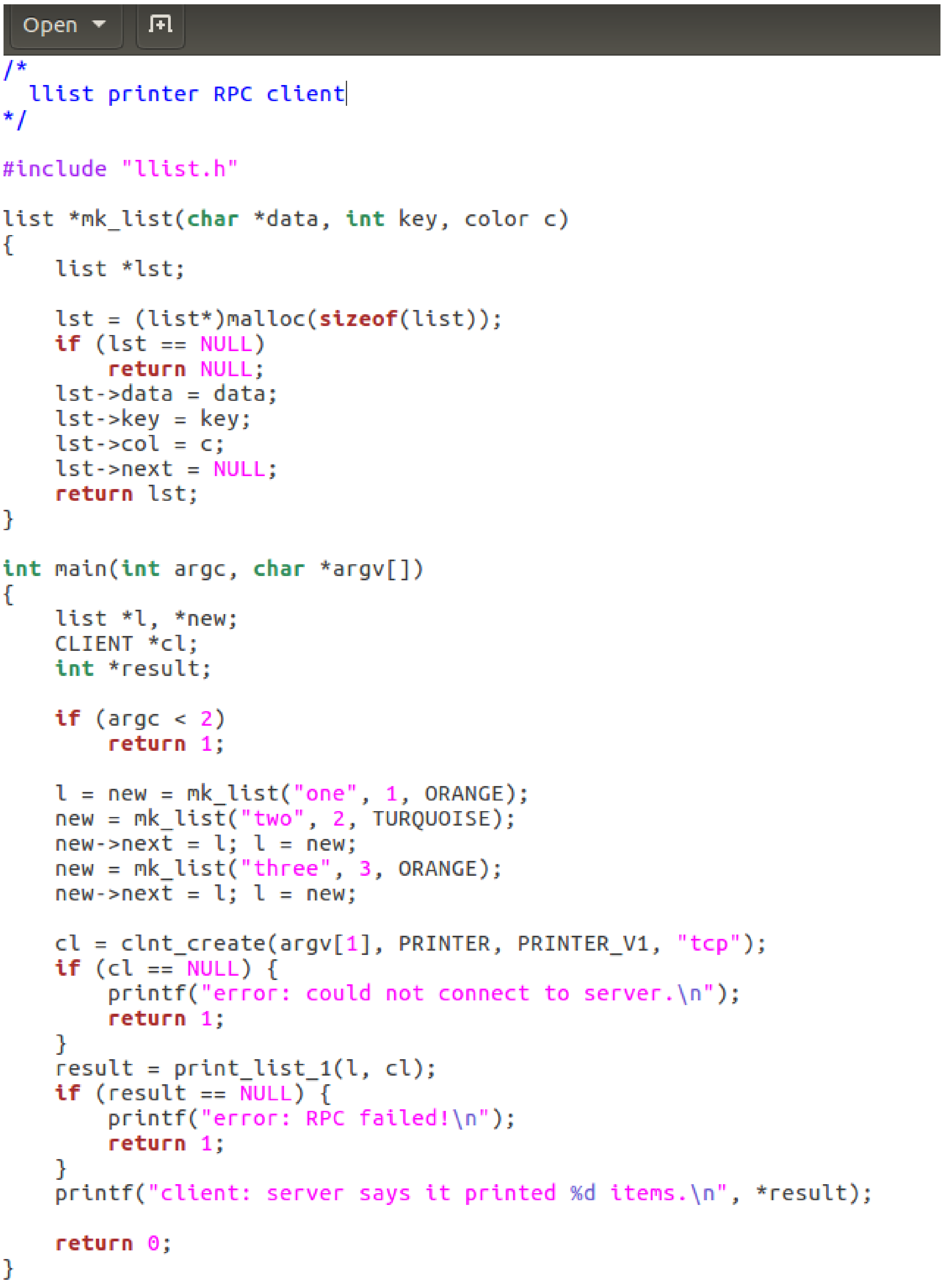
```
warren@warren:~/DC/2$ gedit llist.c
```

llist.c

To create a client program in C that uses Remote Procedure Calls (RPC) to connect to the server and print the contents of a linked list, we can start by defining the necessary data structures and functions. In this case, we also need to create a linked list using a function called mk_list.

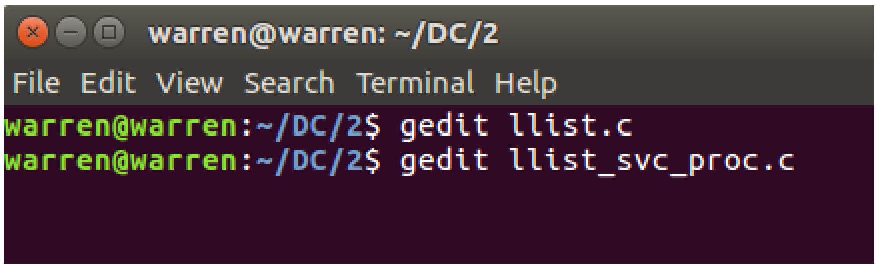
1. Define the necessary data structures for the client program, such as the llist and node structs, and the sum_t struct.
2. Use the clnt_create() function to connect to the RPC server. This function takes the hostname of the server and the program number (in this case, PRINTER) as arguments, and returns a client handle that can be used to call the server's procedures.
3. Implement the mk_list function to create a linked list. This function takes an array of integers and the length of the array as arguments, and returns an llist struct containing the linked list.

4. Call the server's print_list_1() function to print the contents of the linked list. This function takes the client handle and the llist struct as arguments.



```
/*
  llist printer RPC client
*/

#include "llist.h"

list *mk_list(char *data, int key, color c)
{
    list *lst;

    lst = (list*)malloc(sizeof(list));
    if (lst == NULL)
        return NULL;
    lst->data = data;
    lst->key = key;
    lst->col = c;
    lst->next = NULL;
    return lst;
}

int main(int argc, char *argv[])
{
    list *l, *new;
    CLIENT *cl;
    int *result;

    if (argc < 2)
        return 1;

    l = new = mk_list("one", 1, ORANGE);
    new = mk_list("two", 2, TURQUOISE);
    new->next = l; l = new;
    new = mk_list("three", 3, ORANGE);
    new->next = l; l = new;

    cl = clnt_create(argv[1], PRINTER, PRINTER_V1, "tcp");
    if (cl == NULL) {
        printf("error: could not connect to server.\n");
        return 1;
    }
    result = print_list_1(l, cl);
    if (result == NULL) {
        printf("error: RPC failed!\n");
        return 1;
    }
    printf("client: server says it printed %d items.\n", *result);

    return 0;
}
```

We can use gedit text editor to create a file named llist_svc_proc.c for C programming



```
warren@warren:~/DC/2$ gedit llist.c
warren@warren:~/DC/2$ gedit llist_svc_proc.c
```

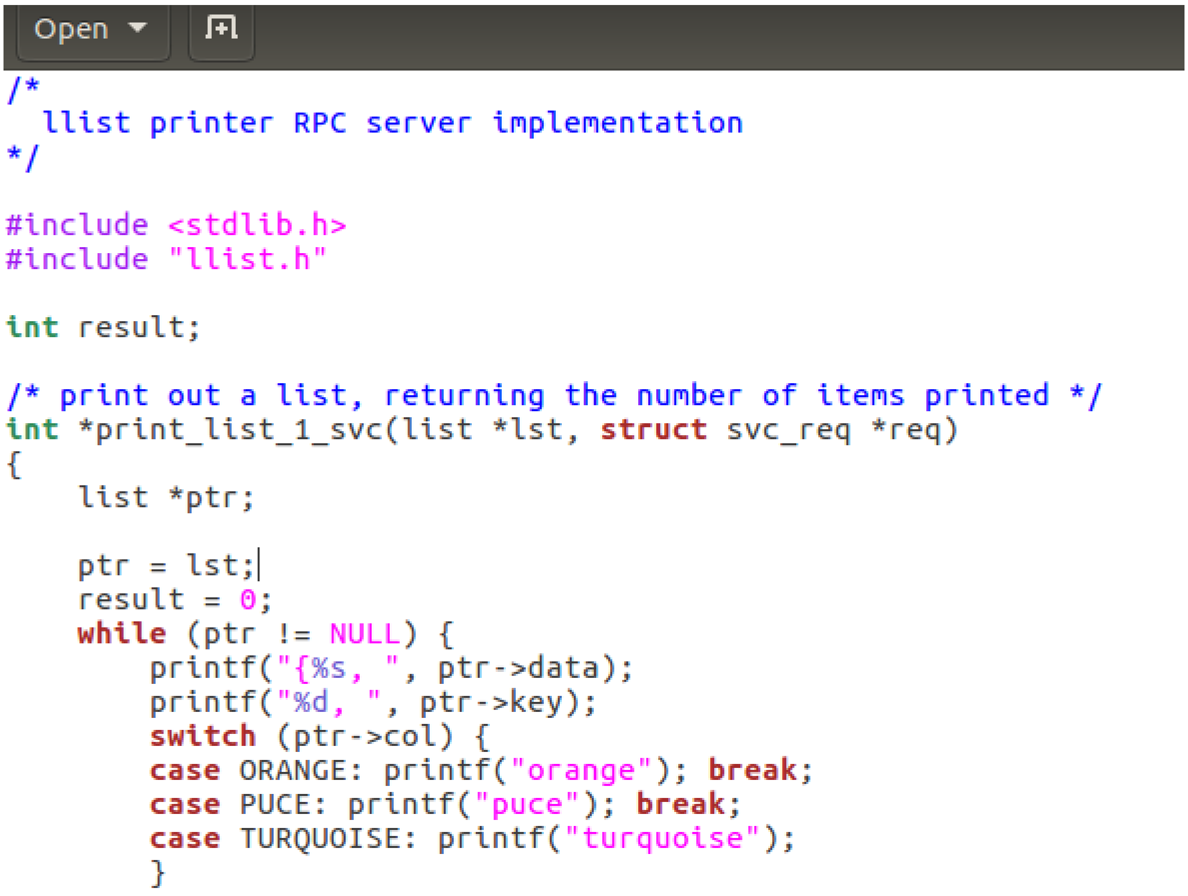
llist_svc_proc.c

To create a server program in C that can handle Remote Procedure Calls (RPC) for printing the contents of a linked list and computing the sum of its elements, we can start by including the header file llist.h that was created using rpcgen. This file contains the necessary data types and function prototypes for handling the RPC requests.

Next, we can implement two functions called print_list_1_svc() and sum_list_1_svc(), which correspond to the print_list and sum_list procedures defined in the IDL file.

The print_list_1_svc() function takes an argument of type llist and returns void. It prints the contents of the linked list to the server's standard output.

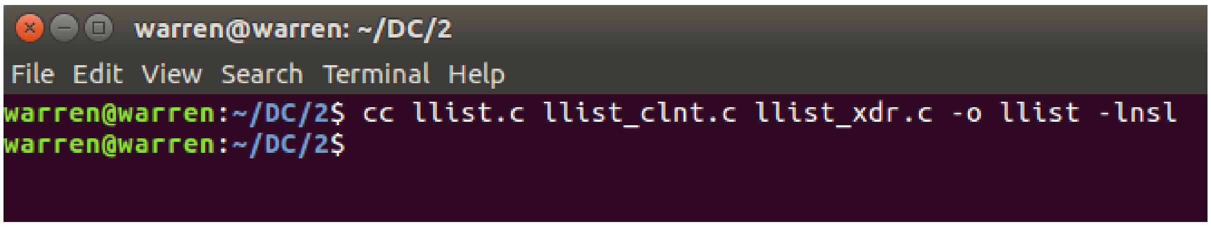
The sum_list_1_svc() function takes an argument of type llist and returns a pointer to a llist struct. It computes the sum of the elements in the linked list and returns the result in the llist struct.



```
/*
   llist printer RPC server implementation
*/

#include <stdlib.h>
#include "llist.h"

int result;

/* print out a list, returning the number of items printed */
int *print_list_1_svc(list *lst, struct svc_req *req)
{
    list *ptr;

    ptr = lst;
    result = 0;
    while (ptr != NULL) {
        printf("{%s, ", ptr->data);
        printf("%d, ", ptr->key);
        switch (ptr->col) {
        case ORANGE: printf("orange"); break;
        case PUCE: printf("puce"); break;
        case TURQUOISE: printf("turquoise");
        }
```

```
            printf("}\n");
            result++;
            ptr = ptr->next;
        }
        return &result;
}

int *sum_list_1_svc(list *lst, struct svc_req *req)
{
        list *ptr;

        ptr = lst;
        result = 0;
        while (ptr != NULL) {
            result += ptr->key;
            ptr = ptr->next;
        }
        return &result;
}
```

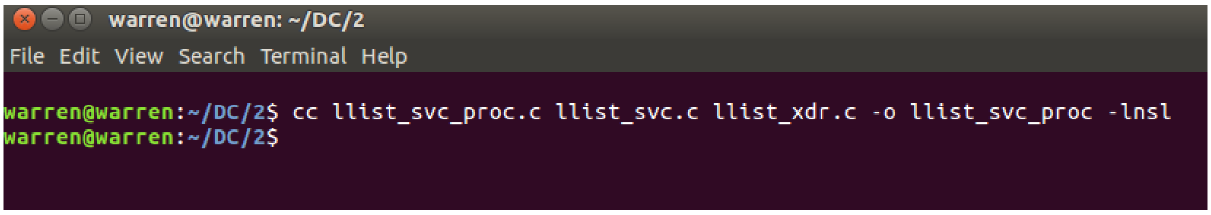
To compile the C file llist.c that contains client-side code for making Remote Procedure Calls (RPC) and uses the generated clnt and xdr code, we can use the following command:



```
warren@warren:~/DC/2$ cc llist.c llist_clnt.c llist_xdr.c -o llist -lnsl
warren@warren:~/DC/2$
```

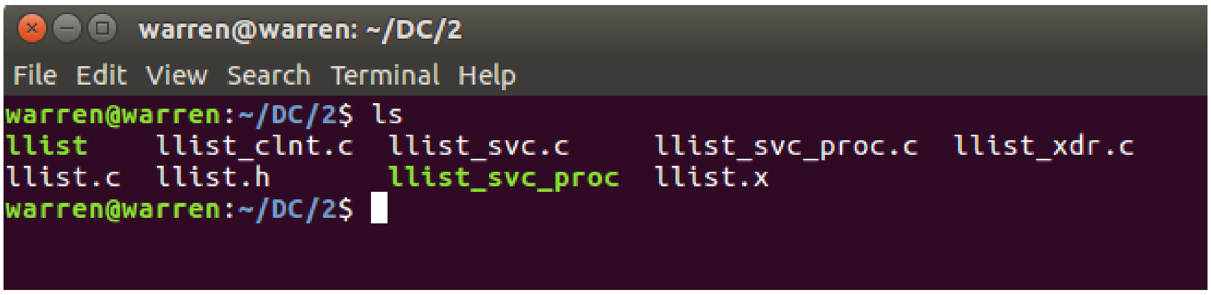
This command compiles the llist.c file along with the llist_clnt.c and llist_xdr.c files that were generated by rpcgen, and produces an executable file called llist.

To compile the C file llist_svc_proc.c that contains server-side code for making Remote Procedure Calls (RPC) and uses the generated svc and xdr code, we can use the following command:



```
warren@warren:~/DC/2$ cc llist_svc_proc.c llist_svc.c llist_xdr.c -o llist_svc_proc -lnsl
warren@warren:~/DC/2$
```

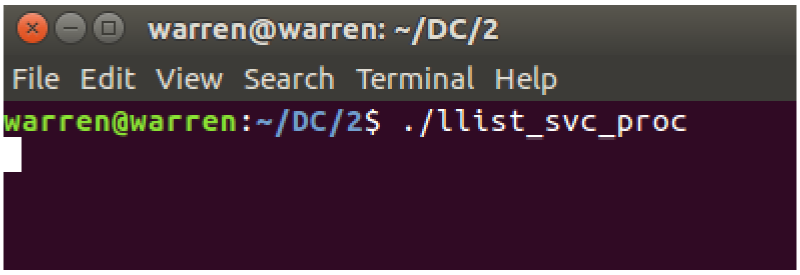
This command compiles the llist_svc_proc.c file along with the llist_svc.c and llist_xdr.c files that were generated by rpcgen, and produces an executable file called llist_svc_proc.



```
warren@warren:~/DC/2$ ls
llist      llist_clnt.c  llist_svc.c      llist_svc_proc.c  llist_xdr.c
llist.c    llist.h       llist_svc_proc   llist.x
warren@warren:~/DC/2$ 
```

To run the executable llist_svc_proc, which is the RPC server for the llist program, we can simply type the name of the executable in a terminal window and press Enter.

For example, if we navigate to the directory where the executable is located and type the following command:



```
warren@warren:~/DC/2$ ./llist_svc_proc
```

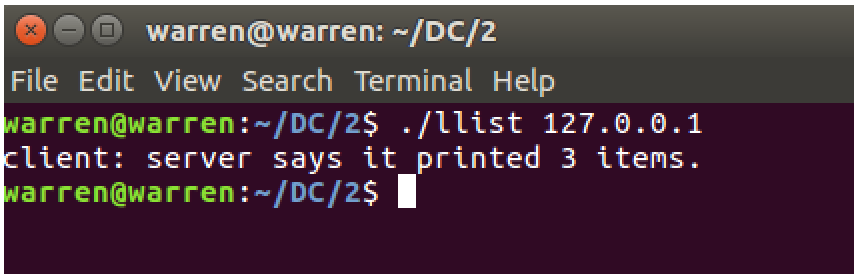
This command starts the llist_svc_proc executable with no additional arguments. By default, the server will listen for incoming RPC requests on port 5001.

Once the server is running, we can use a client program to make RPC calls to the server and perform operations on the linked list. The client program can be compiled and run on the same machine as the server, or on a different machine that has network access to the server.

To run the executable llist, which is the RPC client for the llist program, and connect to a server running on the same machine with IP address 127.0.0.1 (localhost), we need to open a terminal window and navigate to the directory where the executable is located. Then, we can simply type the name of the executable followed by the IP address of the server.

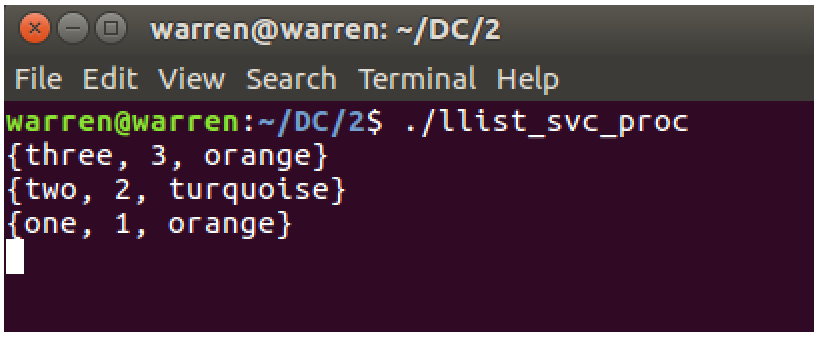
If the server is running on a different machine, we need to replace 127.0.0.1 with the IP address or hostname of the machine where the server is running. We also need to make sure that the server is running and listening on the correct port for incoming RPC requests.

For example, if we want to connect to a server running on the same machine with IP address 127.0.0.1, we can run the following command:



```
warren@warren:~/DC/2$ ./llist 127.0.0.1
client: server says it printed 3 items.
warren@warren:~/DC/2$ 
```

This command starts the llist_client executable and passes the IP address 127.0.0.1 as the argument to connect to the server. After the print_list_1 function is called, the message 'client: server says it printed 3 times' will be printed to the terminal window. The three colours will be printed on the server terminal.



```
warren@warren:~/DC/2$ ./llist_svc_proc
{three, 3, orange}
{two, 2, turquoise}
{one, 1, orange}
```

Conclusion:

In this RPC lab experiment, we have learned about the Remote Procedure Call (RPC) mechanism and how it is used to implement client-server communication over a network. We have created an RPC program using the C programming language and demonstrated how to create a server and a client program that can communicate with each other using RPC.

We have used the rpcgen tool to generate the client and server stubs for our RPC program, which includes the necessary code for establishing the RPC connection and invoking remote procedures. We have created a server program that implements two remote procedures: print_list_1 and sum_list_1. We have also created a client program that uses clnt_create() to connect to the RPC server and invokes the print_list_1 function.

We have learned how to compile the client and server programs using the necessary libraries and dependencies generated by rpcgen. We have run the server program on port 5001 and the client program on the same machine to establish a connection between them and test the RPC mechanism.