|  |  |  |  |
| --- | --- | --- | --- |
| logo_udlap    **EDEI** | **RPCGEN** | | |
| **DEPARTAMENTO** | Computación, Electrónica y Mecatrónica | **MATERIA** | LIS- 3061 |
| **PROFESOR** | Dra. Ofelia Cervantes V | **PERIODO** | Primavera 2015 |
| **id** | 146247 | **Nombre** | Diego Rodriguez |

INTRODUCCION.

Es este documento se muestra los resultados obtenidos en la primer practica de laboratorio, la practica fue dividida en dos partes, en el documento se muestran todas las dificultades que se presentaron para completar la actividad.

OBJETIVO.

El objetivo principal de esta práctica es poner en funcionamiento dos servidores y conectarse a ellos para utilizar sus procesos.

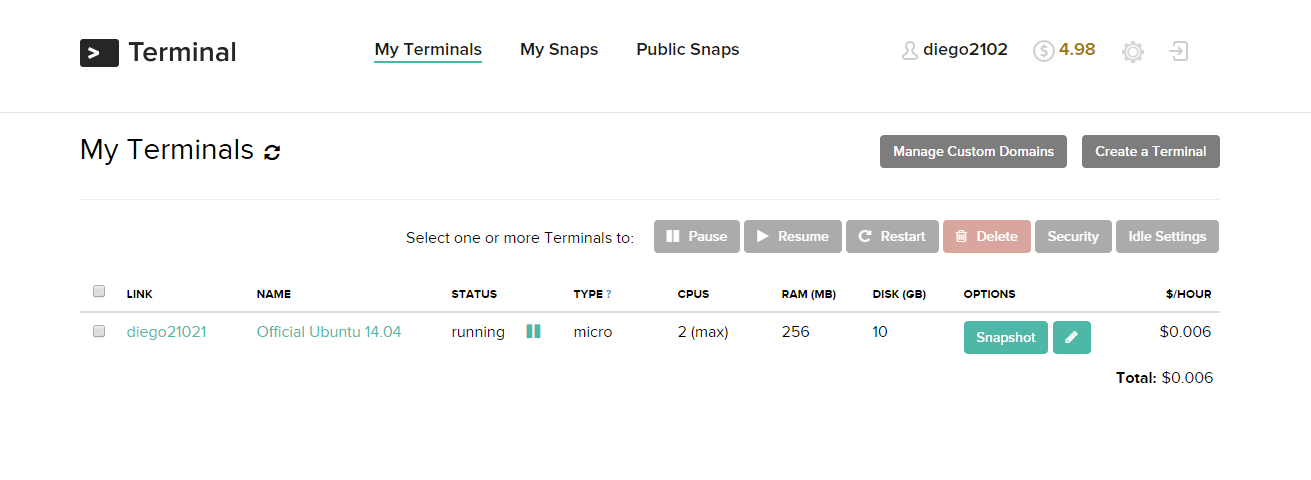
Uno de los objetivos intermedios es completar la parte 1 de la práctica, que comprende utilizar los comandos de Rpcgen para generar los archivos del servidor y del cliente, mediante ellos hacer funcionar el servidor, toda la practica

La segunda parte de la práctica es poner a prueba los conocimientos obtenidos en la parte uno, la aplicación debe de recibir un nombre, calcular la fecha y escribirla en un archivo de texto, otra función debe de recibir un nombre, buscar en el archivo de texto y mostrar la fecha.

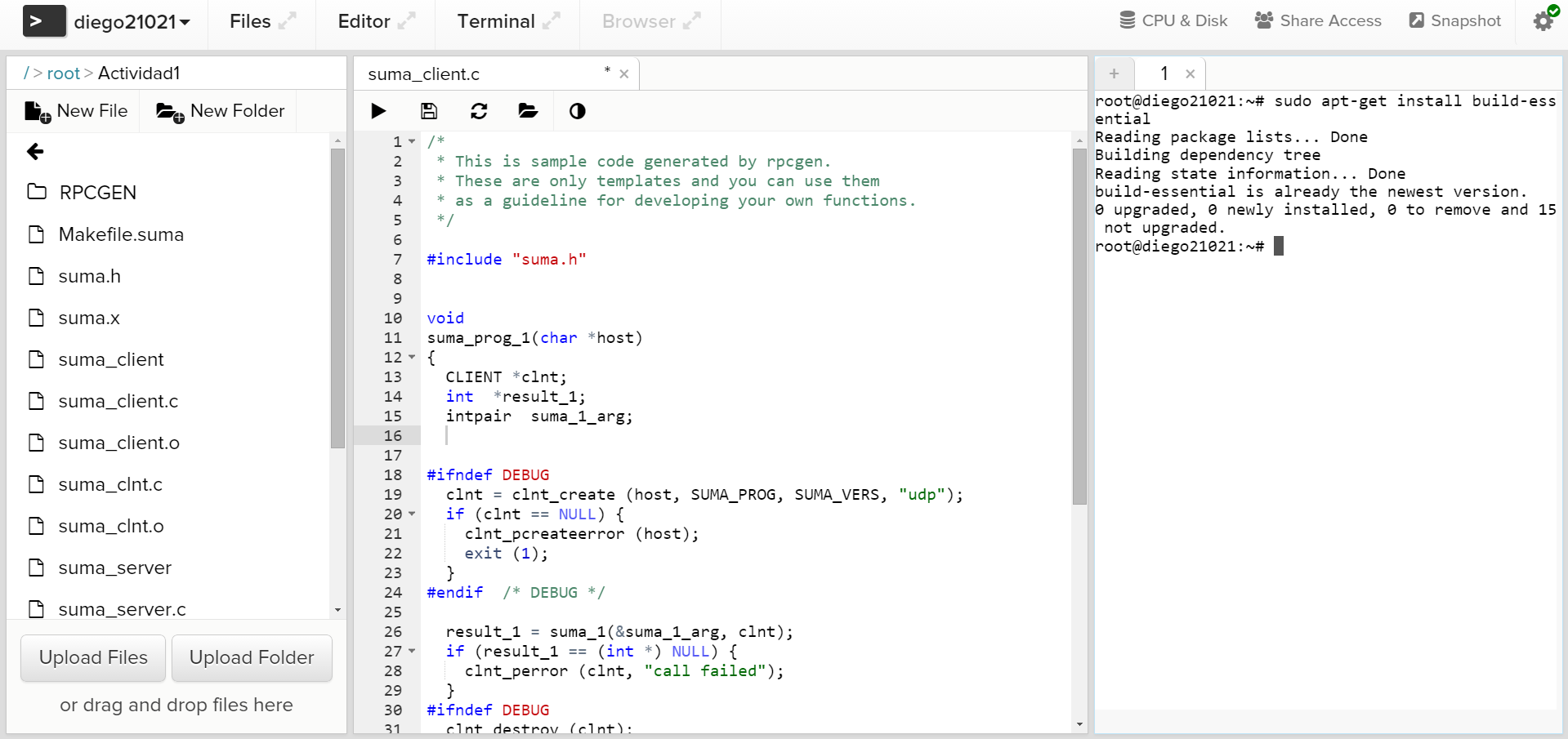
El último objetivo es entender e incrementar el conocimiento del lenguaje C.

CONTENIDO.

Toda la práctica se debe de realizar en sistema Linux, por lo que fue necesario virtualizar Ubuntu, otra alternativa posible para realizar la práctica fue el uso de la página de internet terminal.com, ofrece un servicio de servidores en diferentes sistemas, por lo que me permitió realizar la práctica en un servidor de tipo Linux, aunque es un servicio de paga la página ofrece 5 dólares de regalo para probar la aplicación. A continuación se muestra la página donde se da inicio a los servidores ligados a una cuenta personal relacionada con google+.



Una vez iniciada la terminal de Linux, la página ofrece una interfaz donde se puede iniciar el número de terminales que se requiera, un editor de archivos y un navegador, en este se puede organizar los archivos, como lo veremos en la próxima imagen, del lado derecho se muestran los archivos que se generaron durante la practica 1 parte 1, en el centro se muestra uno de los archivos que se editaron, y final mente del lado izquierdo aparece las terminales que se encuentran funcionando.



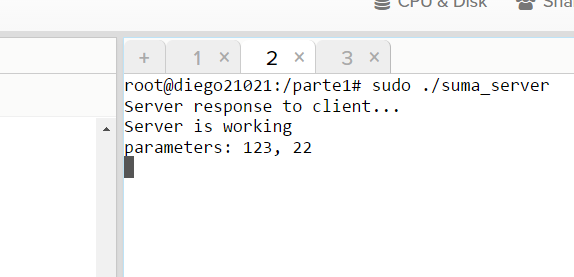
**1.** La primer parte la practica fue muy fácil de completar, solo consistió en copiar los comandos que se encontraban en la dirección de gitHub, <https://github.com/Innova4DLab/RPCGen>.

Mediante el archivo suma que contenía el siguiente código.

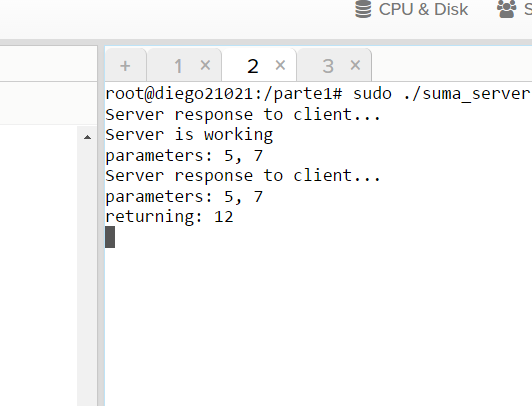
|  |
| --- |
| struct intpair { |
|  | int a; |
|  | int b; |
|  | }; |
|  |  |
|  | program SUMA\_PROG { |
|  | version SUMA\_VERS { |
|  | int SUMA(intpair) = 1; |
|  | } = 1; |
|  | } = 0x23451111; |

Este archivo genera un conjunto de archivos en leguaje C, que contiene las partes principales de la aplicación, el programador no tiene que hacer construir este código que permite la conexión entre cliente y el servidor, y única mente se debe concentrar en la construcción de funciones para la aplicación.

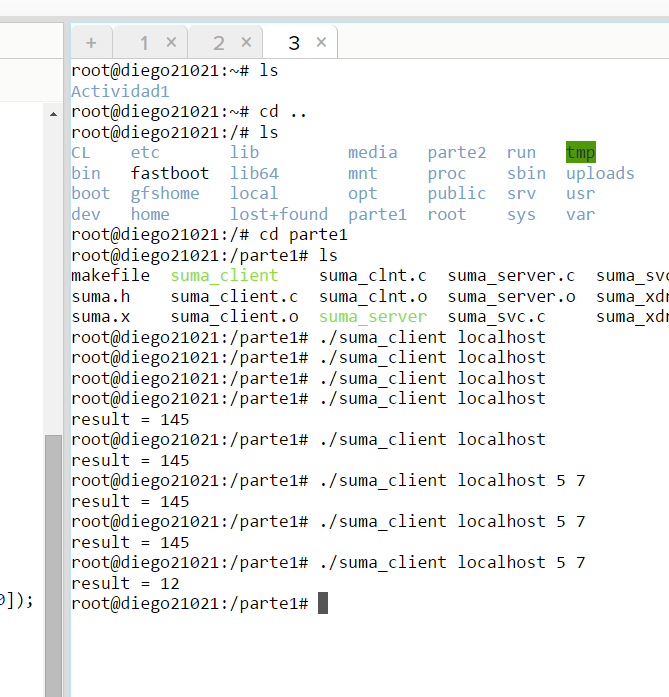
En la siguiente imagen podemos ver los resultados obtenidos después de enviar los parámetros al servidor, la imagen que se muestra abajo corresponde a la información que arroja el servidor



La siguiente imagen muestra el primer procesamiento que se desarrolla desde el servidor, este proceso no regresa nada al cliente, pero ya realiza una operación con datos que fueron enviados desde el cliente



Final mente en la siguiente imagen podemos ver como el cliente puede ver el resultado obtenido y que fue enviado desde el servidor, los datos que el cliente envía son introducido desde la terminal, a diferencia de las imágenes anteriores donde los datos están almacenadas en el código del programa. En esta imagen podemos ver las ejecuciones por parte del cliente a las que el servidor atendió, los resultados fueron mostrados en las imágenes anteriores.



Una vez cubierto estos puntos, la parte 1 de la práctica ha sido completada satisfactoriamente.

**Parte dos de la práctica.**

La parte dos de la practica consistió en hacer uso de los comandos y habilidades obtenidas en la práctica uno y aplicarlas a un nuevo problema, a continuación se muestran los objetivos de la práctica, que nuevamente se encontraron en gitHub.

* Generar un nueva especificación RPCGen (Como add.x) en donde se definan dos funciones:
  + La función "agregar" deberá:
    - Recibir un nombre y obtener la fecha con la librería [Date](http://goo.gl/tSrZ46) de C.
    - [Guardar](http://goo.gl/HQeG4H) en un archivo .txt la fecha y el nombre.
    - Retornar un mensaje si los datos se han escrito satisfactoriamente.
  + La función "buscar" deberá:
    - Recibir un nombre, mediante una [búsqueda](http://stackoverflow.com/a/13450846/1206078) obtener la fecha(s) relacionada(s) al nombre.
    - Retornar el la fecha relacionada para el nombre indicado y/o el número de registros con el mismo nombre.
    - Realizar el [paso 6 de la actividad 1](http://goo.gl/Qm2m59), en cuanto a limpieza del código y comentarios.

1.El archivo add.x debería contener el siguiente código.

program ADD\_PROG {

version ADD\_VERS {

string ADD(string) = 1;

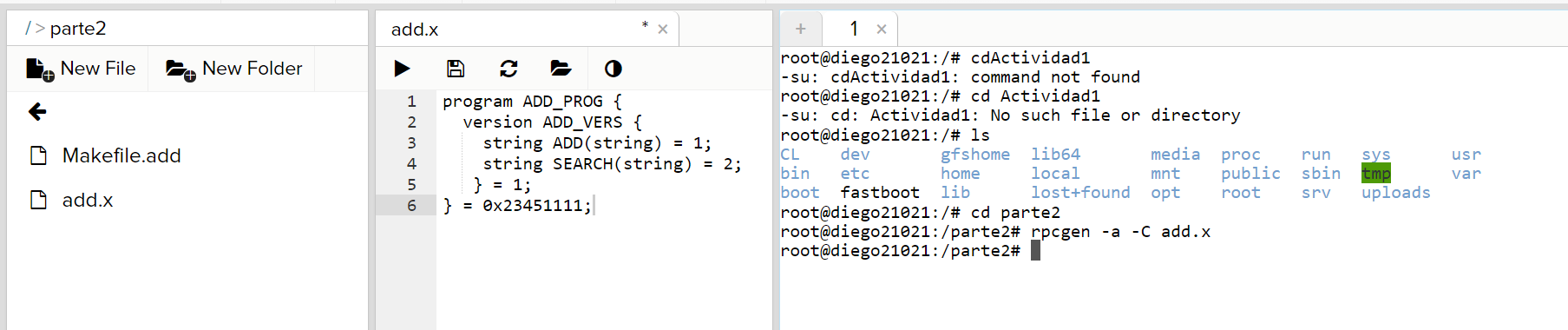
string SEARCH(string) = 2;

} = 1;

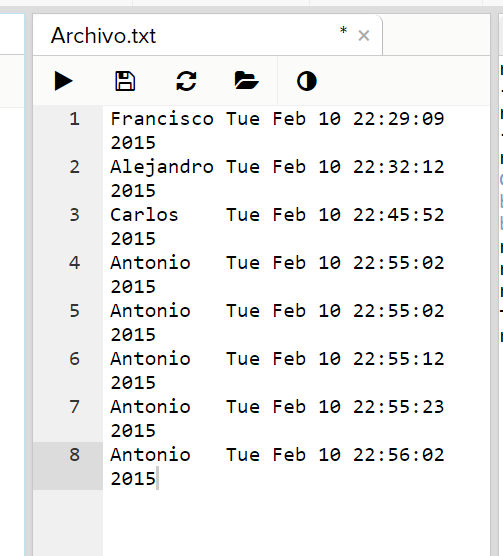
} = 0x23451111;

Con el commando rpcgen -a -C add.x

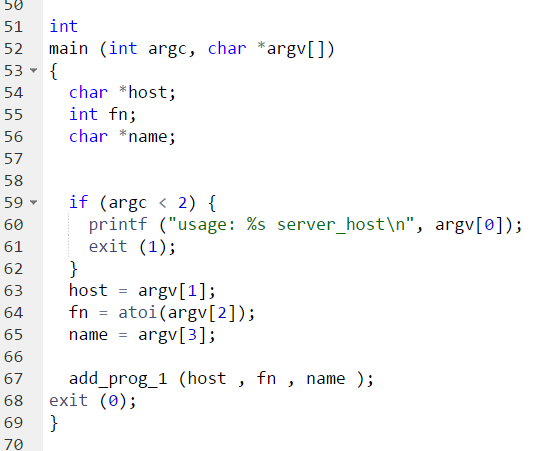
Con este comando se generaron todos los archivos necesarios para realizar la práctica.



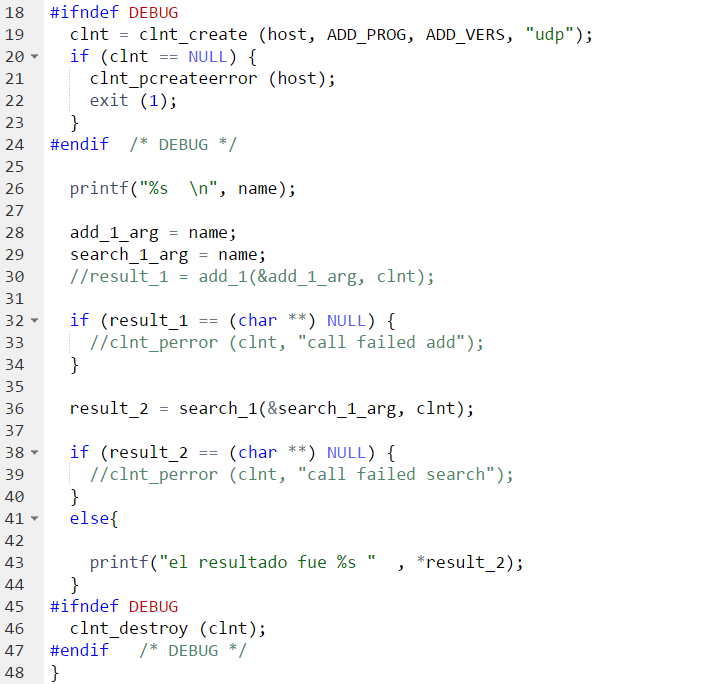
Archivo que genera la aplicación fue el siguiente.



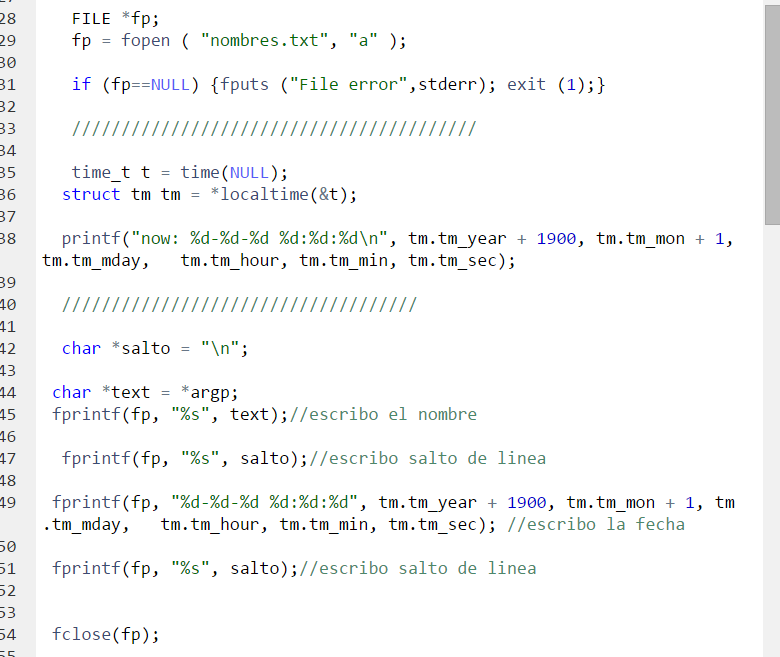
La primer parte a modificar del programa fue el main principal, localizado en el cliente, es el método que se ejecuta al momento de correr el programa, podemos observar que los datos obtenidos desde la terminal son enviados al meto del programa para que inicie el proceso.



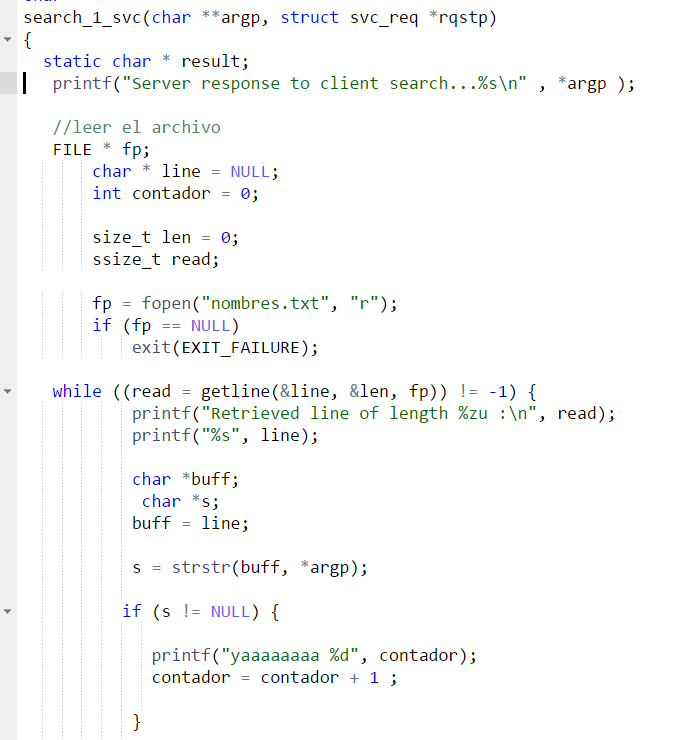
2.-Una vez que los datos llegan al método, automáticamente se genera el cliente y envía los datos que utilizara el servidor además de los datos del cliente para realizar la conexión, en nuestro caso será el local host.



El servidor de la aplicación realiza varias funciones, escribe datos y la fecha en un archivo de texto.

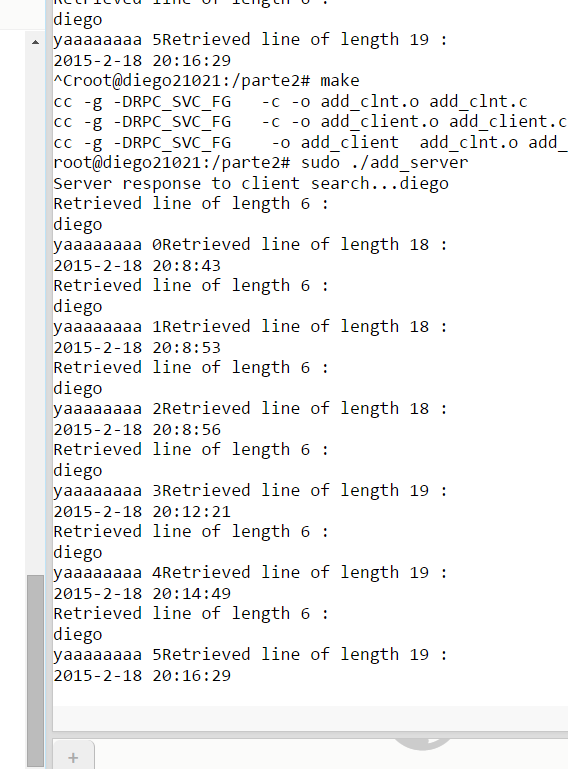


En su defecto busca un nombre en el archivo de texto.

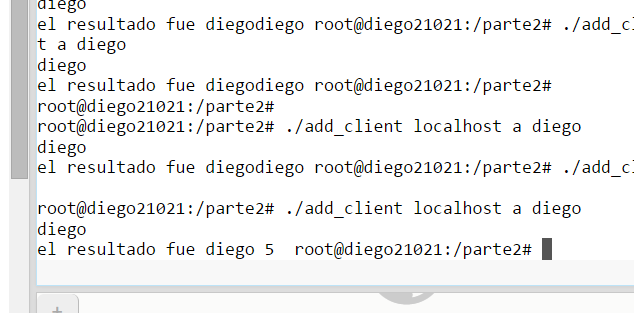


El cliente selecciona que función debe ejecutar el cliente.

3.-Finalmente los resultados son los siguientes, el servidor ejecuta la búsqueda dentro del archivo, y muestra la fecha cuando se realizó la inserción, he imprime los valores.



Y el cliente recibe el número total de veces que se encontró el nombre que envio al iniciar el programa, en este caso la respuesta es de 5.



El archivo que se genero fue el siguiente.



PREGUNTAS

* Identifica los archivos de cliente y servidor generados por RPCGen, comenta en el código tus observaciones. ¿Los nombres de variables son comprensibles?

Los nombres hacen referencia al contenido que se guarda en cada una de ellas

* Identifica la función main. ¿Qué hace? ¿Podrías hacerlo de otra manera? Comenta el código al respecto.

El método permite iniciar la ejecución del programa, el recibe los parámetros de entrada, no obstante durante la ejecución del programa se podrían pedir al usuario los datos.

* ¿Cómo maneja los errores RPCGen? ¿Qué pasa si hay un error y se siguen haciendo llamadas a través de RPC? Comenta en el código en los archivos correspondientes.

El codigo por si solo esta protegido, evita mandar posibles errores, aun asi el programa termina su ejecución si hay algún problema.

* Las líneas con **#ifdef DEBUG** deberían quitarse. ¿Qué puede hacerse para que el código se más legible?

La linead debe mantenerse debido a que le da claridad al código, para un posible mantenimiento en el futuro.

CLONCLUSIONES.

Como conclusión podemos decir que se completaron al 100% por ciento los objetivos, se completó en sus totalidad las dos partes de la practica 1, y se obtuvieron los resultados a adecuados, pero una falta mayor habilidad y trabajo en el lenguaje C, lo que hubiera permitido realizar la practica con mayor velocidad, no obstante con esta práctica se logró comprender el funcionamiento de RPCGEN, y aprender nuevas instrucciones en el lenguaje c.

Es muy importante mencionar que cuando se trabaja con un lenguaje que no está dominado, se debe ser muy cuidadoso con el código que se está desarrollando, debido a que se puede introducir errores de sintaxis o de omisión que puede retrasar el desarrollo.

Final mente la práctica fue completada con éxito, y los objetivos establecidos al inicio del documento fueron completados.

ANEXO

Server

/\*

\* This is sample code generated by rpcgen.

\* These are only templates and you can use them

\* as a guideline for developing your own functions.

\*/

#include "add.h"

#include <stdio.h>

#include <stdlib.h>

#include <stdio.h>

#include <time.h>

#include <stdlib.h>

#include <string.h>

#define atoa(x) #x

/\*

\* agrega un nombre nuevo al archivo

\*/

char \*\*

add\_1\_svc(char \*\*argp, struct svc\_req \*rqstp)

{

static char \* result;

/\*

\* insert server code here

\*/

printf("Server response to client add...%s\n" , \*argp );

/\*

\* crear el archivo nombre.txt si no existe

\*/

FILE \*fp;

fp = fopen ( "nombres.txt", "a" );

if (fp==NULL) {fputs ("File error",stderr); exit (1);}

/\*

\* se obtiene la fecha del equipo y se asigna a la variable tm

\*/

time\_t t = time(NULL);

struct tm tm = \*localtime(&t);

printf("now: %d-%d-%d %d:%d:%d\n", tm.tm\_year + 1900, tm.tm\_mon + 1, tm.tm\_mday, tm.tm\_hour, tm.tm\_min, tm.tm\_sec);

/\*

\* escribo el nombre en el archivo

\* reservo la primer linea despues del ultimo dato para escribir el nombre

\* agrego un salto de linea

\* escribo la fecha

\*/

char \*salto = "\n";

char \*text = \*argp;

fprintf(fp, "%s", text);//escribo el nombre

fprintf(fp, "%s", salto);//escribo salto de linea

fprintf(fp, "%d-%d-%d %d:%d:%d", tm.tm\_year + 1900, tm.tm\_mon + 1, tm.tm\_mday, tm.tm\_hour, tm .tm\_min, tm.tm\_sec); //escribo la fecha

fprintf(fp, "%s", salto);//escribo salto de linea

//ciero el archivo

fclose(fp);

//regreso el resultado

return &result;

}

/\*

\* recibe un nombre y lo busca en un archivo de texto, regresa el numero de repeticones en

\* el archivo

\*/

char \*\*

search\_1\_svc(char \*\*argp, struct svc\_req \*rqstp)

{

static char \* result;

printf("Server response to client search..........%s\n" , \*argp );

/\*

\* abre el archivo nombre.txt

\*/

FILE \* fp;

char \* line = NULL;

int contador = 0;

size\_t len = 0;

ssize\_t read;

fp = fopen("nombres.txt", "r");

if (fp == NULL)

exit(EXIT\_FAILURE);

/\*

\* compara cada linea del archivo con el nombre que se busca, aumentado el contador

\*/

while ((read = getline(&line, &len, fp)) != -1) {

printf("Retrieved line of length %zu :\n", read);

printf("%s", line);

char \*buff;

char \*s;

buff = line;

//me permite hacer la comparacion

s = strstr(buff, \*argp);

//compruebo el resultado de la comparacion

if (s != NULL) {

//aumento el contador

contador = contador + 1 ;

printf("yaaaaaaaa %d", contador);

}

}

fclose(fp);

//imprimo el numero de frecuecias del nombre buscado en el archivo

printf("yaaaaaaaa %d", contador);

result = \*argp , contador;

//regresa el resultado

return &result;

}

CLIENTE

/\*

\* This is sample code generated by rpcgen.

\* These are only templates and you can use them

\* as a guideline for developing your own functions.

\*/

#include "add.h"

void

add\_prog\_1(char \*host , int fn , char \*name ) {

CLIENT \*clnt;

char \* \*result\_1;

char \* add\_1\_arg;

char \* \*result\_2;

char \* search\_1\_arg;

#ifndef DEBUG

clnt = clnt\_create (host, ADD\_PROG, ADD\_VERS, "udp");

if (clnt == NULL) {

clnt\_pcreateerror (host);

exit (1);

}

#endif /\* DEBUG \*/

printf("%s \n", name);

//asigno el nombre a las variables.

add\_1\_arg = name;

search\_1\_arg = name;

//ejecuta la funcion en el servidor

result\_1 = add\_1(&add\_1\_arg, clnt);

if (result\_1 == (char \*\*) NULL) {

//clnt\_perror (clnt, "call failed add");

}

//ejecuta la funcion en el servidor

result\_2 = search\_1(&search\_1\_arg, clnt);

if (result\_2 == (char \*\*) NULL) {

//clnt\_perror (clnt, "call failed search");

}

else{

//imprime el resultado

printf("el resultado fue %s " , \*result\_2);

}

//termina con la ejecucion del servidor

#ifndef DEBUG

clnt\_destroy (clnt);

#endif /\* DEBUG \*/

}

/\*

\* funcion principal, recibe un entero y un arreglo de cadenas chars

\*/

int

main (int argc, char \*argv[])

{

char \*host;

int fn;

char \*name;

//confirma que exita la direccion del servidor

if (argc < 2) {

printf ("usage: %s server\_host\n", argv[0]);

exit (1);

}

host = argv[1];

fn = atoi(argv[2]);

name = argv[3];

//ejecuta el funcion del cliente

add\_prog\_1 (host , fn , name );

exit (0);

}