Virtualización y Sistemas Operativos Avanzados

2020 - Trabajo Práctico No 1

Sockets - RPC

Integrantes:

Moreno Carla: morenocarla593@gmail.com

Berger Roberto: bergerroberto@gmail.com

Documentación:

-Implementación con Sockets:

-Compilación de archivos:

make

-Modo de ejecución del Servidor:

./servidorSCK

-Modo de ejecución del Cliente:

./clienteSCK <IP_Servidor> <Comando> <Parámetro_1>…<Parámetro_N>

-Cuestiones a considerar:

- No ejecuta comandos que impliquen modificar un archivo o directorio. Ejemplos: **cd**, **ssh**, **chmod.**

- Si el comando **no es válido** ya sea porque el comando no exista o uno de los argumentos no sea válido, no puede retornar el error de lo ocurrido al Cliente. Ejemplo: se ejecuta **ls /imágenes** y dicho directorio no existe, el error se muestra en la aplicación del Servidor, pero el Cliente no recibe información alguna. Se intentó colocar en el if(fp=NULL) alguna validación, pero el programa no alcanzaba dicha sentencia.

- Si se ejecuta un comando con una salida muy larga, el Cliente recibe parte de ella.
  Ejemplo: se ejecuta **man ls** la salida que recibe está incompleta, comparándola con la ejecución de dicho comando desde la terminal. Esto aplica tanto para Cliente como Servidor. Es probable que sea por la definición de popen().

-Implementación con RPC:

Compilación de archivos:

make

Modo de ejecución del Servidor:

./servidorRPC

Modo de ejecución del Cliente:

./clienteRPC <IP_Servidor> <Comando> <Parametro_1>…<Parametro_N>

Cuestiones a considerar:

- Si se ejecuta un comando con una salida muy larga, el Cliente recibe parte de ella.
  Ejemplo: se ejecuta **man ls** la salida que recibe está incompleta, comparándola con la ejecución de dicho comando desde la terminal. Esto aplica tanto para Cliente como Servidor. Es probable que sea por la definición de popen().

- No ejecuta comandos que impliquen modificar un archivo o directorio. Ejemplos: **cd**, **ssh**, **chmod.**

- Si el comando **no es válido** ya sea porque el comando no exista o uno de los argumentos no sea válido, no puede retornar el error de lo ocurrido al Cliente. Ejemplo: se ejecuta **ls /imágenes** y dicho directorio no existe, el error se muestra en la aplicación del Servidor, pero el Cliente no recibe información alguna. Al igual que en el programa anterior, se intentó utilizar una validación con if(fp==NULL) pero nunca llegaba a dicha sentencia.

Código Fuente - RPC:

- comando.x:

```
struct comando {
 int tam;
 string elementos<>;

};

program COMANDO_PRG {
 version COMANDO_1 {
        string ejecucionComando(struct comando) = 1;
} = 1;
} = 0x20000001;
```

- comandoServidor.c:

```
/*
 * 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 "comando.h"

char **
ejecucioncomando_1_svc(struct comando *argp, struct svc_req *rqstp)
{
        static char *result = NULL;
        struct comando argumento = *argp;

        /*
         * insert server code here
         */
   //Agrego logica del popen aqui
   FILE *fp;
   int status;
   char path[PATH_MAX] = {0};
   char comando[1024]={0};

        //printf("Valor comando: %s %d\n", argumento.elementos, strlen(argumento.elementos));
   strcpy(comando, argumento.elementos);

   printf("Comando a ejecutar: %s %d\n ", comando, strlen(comando));
   fp=popen(comando, "r");

        if (fp == NULL) {
      //Coloco result en null
     //result=NULL;
     return NULL;
    }
```

```
            result = malloc(PATH_MAX+1); //El 1 es para el null
            strcpy(result, "");


    while (fgets(path, PATH_MAX, fp) != NULL && (strlen(result)+strlen(path))<PATH_MAX) {
       strcat(result, path);
    }

    printf("Resultado: %s\n", result);

            status = pclose(fp);
    if (status == -1) {
          /* Error reported by pclose() */
    } else {
          /* Use macros described under wait() to inspect `status' in ord$
           to determine success/failure of command executed by popen() */
    }

            return &result;
}
```

- **comandoCliente.c:**

```
/*
 * 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 "comando.h"


void
comando_prg_1(char *host, char *argumento)
{
        CLIENT *clnt;
        char * *result_1;
        struct comando  ejecucioncomando_1_arg;

#ifndef DEBUG
        clnt = clnt_create (host, COMANDO_PRG, COMANDO_1, "udp");
        if (clnt == NULL) {
                clnt_pcreateerror (host);
                exit (1);
        }
#endif  /* DEBUG */

        strcpy(ejecucioncomando_1_arg.elementos,argumento);
        printf("Comando: %s\n",ejecucioncomando_1_arg.elementos);

        /*result_1 = malloc(PATH_MAX+1);*/
        result_1 = ejecucioncomando_1(&ejecucioncomando_1_arg, clnt);
```

```
	if (result_1 == (char **) NULL) {
		clnt_perror (clnt, "call failed");
	}
	printf("Resultados: \n %s \n", *result_1);

#ifndef DEBUG
	clnt_destroy (clnt);
#endif	 /* DEBUG */
}


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

	char *host;
	char comando[1024] = {0};

	if (argc < 3) {
		printf ("usage: %s server_host command\n", argv[0]);
		exit (1);
	}

	//printf("Entro al for\n");

	for(int i=2; i<argc; i++){

		//printf("Concateno comando con arg\n");
		strcat(comando, argv[i]);
		strcat(comando," ");
	}

	host = argv[1];
	printf("Llamada al procedimiento\n");
	comando_prg_1 (host, comando);

	exit (0);

}
```

Código Fuente - Sockets:

- server_sck.c:

```
// Server side C/C++ program to demonstrate Socket programming
#include <unistd.h>
#include <stdio.h>
#include <sys/socket.h>
#include <stdlib.h>
#include <netinet/in.h>
#include <string.h>
#include <errno.h>

struct argumentos {
   char *a;
   char *b;
};
#define PORT 8080

#define PATH_MAX 30

int main(int argc, char const *argv[])
{

   //
   // Creating socket file descriptor
   //int sockfd = socket(domain, type, protocol)->Retorna un entero que es el socket descriptor
   //AF_INET -> IPv4, SOCK_STREAM: Orientado a la conexión, confiable
   //0: IP
   int server_fd;
   if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0)
   {
      perror("socket failed");
      exit(EXIT_FAILURE);
   }


   //Manipula opciones para el socket referido por el file descriptor
   //server_fd.
   //int setsockopt(int sockfd, int level, int optname, const void *optval, socklen_t optlen);
   //level: SOL_SOCKET

   // Forcefully attaching socket to the port 8080
   //Se setea SO_REUSEADDR y el otro en 1 para que pueda ser reutilizada
   //la dirección ip y puerto en otras conexiones
   int opt = 1;
   if (setsockopt(server_fd, SOL_SOCKET, SO_REUSEADDR | SO_REUSEPORT,
                                    &opt, sizeof(opt)))
   {
      perror("setsockopt");
      exit(EXIT_FAILURE);
   }


   //Define estructura del socket: dirección IP, puerto (PORT definido arriba), etc
   struct sockaddr_in address;
   address.sin_family = AF_INET;
   //Esto es para configurar la dirección IP del socket.
   //Con INADDR_ANY, el bind va a tomar todas las interfaces, no solo
   //Localhost.
   address.sin_addr.s_addr = INADDR_ANY;
   address.sin_port = htons( PORT );
```

```
// Forcefully attaching socket to the port 8080
if (bind(server_fd, (struct sockaddr *)&address,
                         sizeof(address))<0)
{
    perror("bind failed");
    exit(EXIT_FAILURE);
}

//Si pudo ejecutarse el bind, el servidor intenta ejecutar el listen
//que pone al servidor a escuchar peticiones
//El 3 indica la longitud maxima a la que la cola pendiente de conexiones puede
//crecer. Si llega un request cuando esta llena (son más de 3 peticiones), el cliente recibe un error

if (listen(server_fd, 3) < 0)
{
    perror("listen");
    exit(EXIT_FAILURE);
}


//Revisa la cola de conexiones pendientes, y si tiene elementos, toma
//el primero, crea un nuevo socket para realizar la conexión, y new_socket
//contiene el nuevo fd del socket que referencia. Se establece conexión
//entre cliente y servidor

//El servidor se queda escuchando peticiones

while(1){

    int new_socket;
    int addrlen = sizeof(address); //Longitud de la dirección
    if ((new_socket = accept(server_fd, (struct sockaddr *)&address,
                    (socklen_t*)&addrlen))<0)
    {
                      perror("accept");
          exit(EXIT_FAILURE);
    }


    //Declaro la estructura donde voy a almacenar lo que envió el cliente
    char buffer[1024] = {0};
    //Almacena el file descriptor del buffer
    //Lee lo que vino desde cliente y lo coloca en el buffer definido
    int valread;
    valread = read( new_socket , buffer, 1024);

    printf("Comando a Ejecutar: %s\n", buffer);


    FILE *fp;
    int status;
    char path[PATH_MAX];

    fp = popen(buffer, "r");
    if (fp == NULL) {

        send(new_socket , "Hubo un error" , strlen("Hubo un error") , 0);

    }

    //Se inicializa en 0 los elementos, porque si no almacena basura
    //Aparacen simbolos raros en la salida del cliente.
    char mensaje[1024]={0};

    while (fgets(path, PATH_MAX, fp) != NULL ) {
```

```
            if((strlen(mensaje)+strlen(path))>1024){

                send(new_socket , mensaje , strlen(mensaje) , 0);
                valread = read( new_socket , buffer, 1024);
                //printf("%s\n", buffer);

                //Inicializo en cero el array:
                for (int i = 0; i < strlen(mensaje) ; i++){
                    mensaje[i] = 0;
                }

            }

            strcat(mensaje, path);

        }

        send(new_socket , mensaje, strlen(mensaje), 0);
        send(new_socket , "RECIBIDO" , sizeof("RECIBIDO") , 0);
        //printf("Salgo del while\n");
        status = pclose(fp);
        if (status == -1) {
            /* Error reported by pclose() */
        } else {
            /* Use macros described under wait() to inspect `status' in order
            to determine success/failure of command executed by popen() */
        }

    }

    return 0;
}
```

- client_sck.c:

```
// Client side C/C++ program to demonstrate Socket programming
#include <stdio.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <string.h>
#define PORT 8080

int main(int argc, char const *argv[])
{

//Lógica de Sockets

    //Crea socket del cliente con el que se comunicará con servidor
    //Tiene mismo protocolo, domain, y type
    int sock = 0;
    if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0)
    {
        printf("\n Socket creation error \n");
        return -1;
    }
```

```
    //Se define una estructura con la dirección del servidor.
    //Donde especifico IP?
    struct sockaddr_in serv_addr;
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_port = htons(PORT);

    // Convert IPv4 and IPv6 addresses from text to binary form
    //Esto convierte el "127.0.0.1" en AF_INET para que pueda ser usado
    //en la estructura de serv_addr
    //AQUI ES DONDE VA ARGV[1]
    if(inet_pton(AF_INET, argv[1], &serv_addr.sin_addr)<=0)
    {
        printf("\nInvalid address/ Address not supported \n");
        return -1;
    }

    //Conecta el socket del cliente (de fd = sock) con el socket del servidor

    if (connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0)
    {
        printf("\nConnection Failed \n");
        return -1;
    }

//Lógica de la conexión. Envío de mensajes

    //Defino variable que almacenará el fd de la conexion?
    int valread;
    char buffer[1024] = {0};
    char comando[1024] = {0};

    //Armo el comando que quiere ejecutar
    int i=2;
    while(i<argc){

        strcat(comando, argv[i]);
        strcat(comando, " ");

        i++;
    }

    send(sock , comando , strlen(comando) , 0 );

    //Imprime la respuesta del servidor
    valread = read( sock, buffer, 1024);

    while(valread>0){


        if(strstr(buffer, "RECIBIDO")){

            if(strlen(buffer) == strlen("RECIBIDO")){

                send(sock, "Recibido", strlen("Recibido"), 0);
```

```
            return 0;

        }

        char aux[1024] = {0};
        strncpy(aux, buffer, strlen(buffer)-sizeof("RECIBIDO"));
        printf("%s\n",aux);
        send(sock, "Recibido", strlen("Recibido"), 0);
        return 0;

    }

    printf("%s\n",buffer );
    send(sock, "Recibido", strlen("Recibido"), 0);
    valread = read( sock, buffer, 1024);


}




return 0;
}
```