UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Arquitectura Cliente-servidor.

Grupo 01

Proyecto Final

“CLIENTE-SERVIDOR SSH”

Alumnos:

Cortés López Maricela.

Hernández Calderón Fernando.

Profesor: Ing. Carlos Alberto Román Zamitiz

Semestre: 2022-2

Índice

Objetivo.

El alumno creara un Cliente-Servidor que ejecute comandos remotamente, como ocurre con un Cliente-Servidor SSH comercial o gratuito el cual estará desarrollado en una arquitectura cliente-servidor, con sockets TCP/IP y con conexión remota desarrollado en un ambiente Linux.

Introducción.

Arquitectura Cliente servidor.

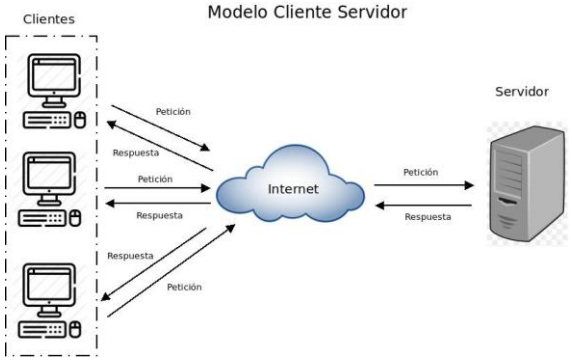
Esta arquitectura consiste básicamente en un cliente que realiza peticiones a otro programa (el servidor) que le da respuesta. Aunque esta idea se puede aplicar a programas que se ejecutan sobre una sola computadora es más ventajosa en un sistema operativo multiusuario distribuido a través de una red de computadoras. La interacción cliente-servidor es el soporte de la mayor parte de la comunicación por redes ya que ayuda a comprender las bases sobre las que están construidos los algoritmos distribuidos.

El servidor debe negociar con su Sistema Operativo un puerto donde esperar las solicitudes. El servidor espera pasivamente las peticiones en un puerto bien conocido que ha sido reservado para el servicio que ofrece. El cliente también solicita, a su sistema operativo, un puerto no usado desde el cual enviar su solicitud y esperar respuesta. Un cliente ubica un puerto arbitrario, no utilizado y no reservado, para su comunicación.

Diferencia entre cliente y servidor

El cliente es un computador pequeño con una estructura al igual a la que tenemos en nuestras oficinas u hogares la cual accede a un servidor o a los servicios del mismo a través de Internet o una red interna. Un claro ejemplo a este caso es la forma en que trabaja una empresa modelo con diferentes computadores donde cada uno de ellos se conectan a un servidor para poder obtener archivos de una base de datos o servicios ya sea correos electrónicos o aplicaciones.

El servidor al igual que el cliente, es una computadora, pero con diferencia de que tiene una gran capacidad que le permite almacenar gran cantidad de diversos de archivos, o correr varias aplicaciones en simultaneo para así nosotros los clientes poder acceder los servicios.



SSH (Secure SHell)

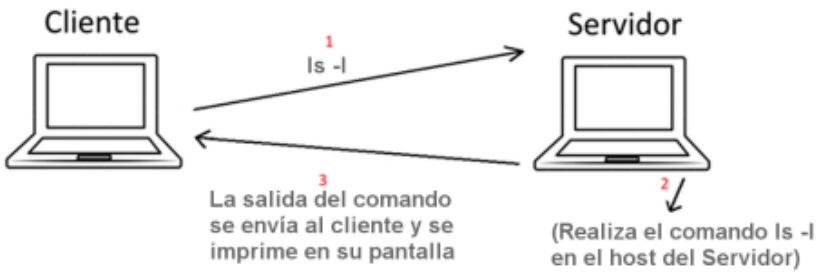
Es un protocolo que facilita las comunicaciones seguras entre dos sistemas usando una arquitectura cliente/servidor y que permite a los usuarios conectarse a un host remotamente.

SSH encripta la sesión de conexión, haciendo imposible que alguien pueda obtener contraseñas no encriptadas. SSH está diseñado para reemplazar los métodos más viejos y menos seguros para registrarse remotamente en otro sistema a través de la shell de comando, tales como telnet o rsh.. El uso de métodos seguros para registrarse remotamente a otros sistemas reduce los riesgos de seguridad tanto para el sistema cliente como para el sistema remoto

Desarrollo

Especificaciones

- La comunicación entre el Cliente y el Servidor debe ser remota vía sockets TCP (Sockets Internet)
- El programa Servidor debe iniciarse en el host servidor (en el puerto que decidas).
- El programa Cliente debe iniciarse en el host cliente (pasando el dominio o IP del servidor y el puerto desde línea de comandos).
- El Servidor acepta la conexión.
- Una vez aceptada la conexión, el Cliente escribe el comando y lo envía por el socket al Servidor (Paso # 1 de la imagen). En este ejemplo se muestra el comando ls –l pero puede ser cualquier comando de Linux o MacOS.
- El Servidor recibe el comando y lo ejecuta en sistema local (Paso # 2 de la imagen). Puedes usar las siguientes funciones: system(), fork() y la familia de funciones exec().
- El Servidor debe devolver la salida al cliente (Paso # 3 de la imagen).



- En la imagen anterior, el cliente ejecuta el comando ls –l pero no es una ejecución local en el cliente. El comando se aplica en el servidor y le envía el resultado al cliente.
- Por último, con el comando: salir (o exit) El cliente debe desconectarse del servidor.

Código Cliente

```
/*Proyecto Final de la Materia de Arquitectura Cliente-Servidor
Objetivo: Crear un Cliente-Servidor que ejecute comandos
remotamente como ocurre con un cliente-Servidor SSH comercial
Cortes Lopez Maricela
Hernandez Calderon Fernando
                                        Archivo Cliente*/

//Bibliotecas necesarias.
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <sys/wait.h>
#include <netdb.h>


// Puerto por el que se conectara el cliente
#define PORT 3490
// Numero maximo de bytes en el mensaje
#define MAXDATASIZE 300

int main(int argc, char *argv[]){
 int sockfd, numbytes;
 // Inicializamos el buffer con el tamaño maximo de bytes
 char bufEnvia[MAXDATASIZE];
 char bufRecibe[MAXDATASIZE];
 //Estructura para almacenar la informacion del host
 struct hostent *he;

 // Informacion de los conectores
 struct sockaddr_in their_addr;

 // Si no se ingresa un segundo argumento
 if(argc != 2){
  fprintf(stderr, "Client-Usa: %s host_servidor\n", argv[0]);
  exit(1);
 }

 // Intentamos obtener la informacion del host
 if((he=gethostbyname(argv[1])) == NULL){
  perror("gethostbyname()");
  exit(1);
 }
```

```
else
  printf("Client-El host remoto es: %s\n", argv[1]);

// Intentamos abrir el socket para iniciar la comunicacion
if((sockfd = socket(AF_INET, SOCK_STREAM, 0)) == -1){
  perror("socket()");
  exit(1);
} else {
  printf("Client-Se logro crear socket...\n");
}

// Configuramos los detalles de la conexion
their_addr.sin_family = AF_INET;
printf("Server-Usando %s con el puerto %d...\n", argv[1], PORT);
their_addr.sin_port = htons(PORT);
their_addr.sin_addr = *((struct in_addr *)he->h_addr);

//Ponemos los demas parametros de la estructura en 0
memset(&(their_addr.sin_zero), '\0', 8);

//Probamos si podemos conectarnos al servidor
if(connect(sockfd, (struct sockaddr *)&their_addr, sizeof(struct sockaddr)) == -1){
  perror("connect()");
  exit(1);
} else{
  printf("Client-Conectado...\n");
}

// Comenzamos a comunicarnos con el servidor
while (strcmp(bufEnvia,"exit\n") != 0) {
  //Limpiamos el vector bufEnvia y bufRecibe
  memset(bufEnvia,0,MAXDATASIZE);
  memset(bufRecibe,0,MAXDATASIZE);

  // Escribimos el comando a ejecutar
        printf("client> ");
        fgets(bufEnvia,MAXDATASIZE,stdin);

  //Intentamos enviar el mensaje al servidor
        if(send(sockfd, bufEnvia, MAXDATASIZE, 0) == -1){
        perror("Server-send() error lol!");
        } else{
        //printf("Servidor- Recibio %s\n", bufEnvia);
          }

  //Intentamos recibir la respuesta del servidor
  //if((numbytes = recv(sockfd, bufRecibe, MAXDATASIZE-1, 0)) == -1){
  //  perror("recv()");
  //  exit(1);
  //}else {
```

```
    //Ponemos fin de cadena al mensaje
  //  bufRecibe[numbytes] = '\0';
  //  printf("Servidor- Envia %s", bufRecibe);
  //}


  // Limpiamos la entrada y salida
  fflush(stdout);
  fflush(stdin);

  // Recibimos todos los mensajes del servidor hasta que envie un "termine"
  while (strcmp(bufRecibe,"termine\n") != 0){
       if (strlen(bufRecibe) >= 3){
         printf("%s", bufRecibe);
       }
       // Limpiamos la entrada y salida
       fflush(stdout);
       fflush(stdin);
       // Limpiamos el vector bufRecibe
       memset(bufRecibe,0,MAXDATASIZE);
       recv(sockfd, bufRecibe, MAXDATASIZE, 0);
      }
      printf("\n");
 }
 // Cerramos el socket del cliente
 printf("Client-Closing sockfd\n");
 close(sockfd);
 return 0;
}
```

Código Servidor.

```
/*Proyecto Final de la Materia de Arquitectura Cliente-Servidor
Objetivo: Crear un Cliente-Servidor que ejecute comandos
remotamente como ocurre con un cliente-Servidor SSH comercial
Cortes Lopez Maricela
Hernandez Calderon Fernando
                                                    Archivo Servidor*/

//Bibliotecas necesarias.
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <sys/wait.h>
#include <signal.h>

//Puerto al que nos queremos conectar
#define ELPUERTO 3490
//Establecemos un maximo de bytes que puede recibir a la vez
#define MAXDATASIZE 300
//El tamaño de cola de escucha
#define BACKLOG 5

//Variable global para cerrar el servidor al terminar
int flag = 0;

int conexion(int new_fd){
        int numbytes;
        // buf sera la entrada del usuario
        char buf[MAXDATASIZE];
        // salida sera el resultado del comando
        char salida[MAXDATASIZE];
        // comando es el apuntador al resultado del shell
        FILE *comando;

        while( strcmp (buf,"exit\n") != 0){
                memset(buf,0,MAXDATASIZE);
                //Verificamos que el mensaje recibido no tiene algun error
                if((numbytes = recv(new_fd, buf, MAXDATASIZE, 0)) == -1){
                        perror("recv()");
                        exit(1);
                }

                // Ponemos fin de cadena al mensaje
```

```
                        buf[numbytes] = '\0';
                        printf("Servidor- Recibe: %s \n", buf);

                        if(system(buf)!=0){
                                send(new_fd,"Comando desconocido\n", MAXDATASIZE, 0);
                                send(new_fd, "termine\n", MAXDATASIZE, 0);
                        }else{
                                //Mandamos a llamar al shell para ejecutar el comando
                                fflush(stdout);
                                if((comando = popen(buf,"r")) == NULL ){
                                        perror("popen error");
                                }

                                // Enviamos resultado al cliente
                                while (fgets(salida, MAXDATASIZE, comando) != NULL){
                                printf("Servidor- Envia: %s", salida);
                                send(new_fd, salida, MAXDATASIZE, 0);
                        }
                        // Limpiamos despues de haber enviado resultado
                        fflush(stdout);
                        fflush(stdin);
                        // Enviamos mensaje que termino el resultado
                        send(new_fd, "termine\n", MAXDATASIZE, 0);
                        // Cerramos el FILE comando
                                pclose(comando);
                        }

                }
                send(new_fd, "Conexion cerrada\n", MAXDATASIZE, 0);
                //Cerramos el socket de coneccion con el cliente
                close(new_fd);
                printf("Server-new socket, new_fd closed successfully...\n");
}

void manejador_senales(int sig){
 if( sig == SIGINT ){
                printf("\nCerrando servidor, adios\n");
                close(flag);
                exit(1);
 }
}

int main(int argc, char *argv[ ]){
        //Variables para la escucha del socket, nuevas conexiones y num de bytes
        int sockfd, new_fd, numbytes;
        //Estructura donde se guarda la info de nuestra direccion
        struct sockaddr_in my_addr;
        // Informacion de la direccion de los conectores
        struct sockaddr_in their_addr;
        int sin_size;
```

```
	//Comprobamos si hay un error con el descriptor de fichero
	if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) == -1){
		perror("Server-socket() error lol!");
		exit(1);
	} else{
		printf("Servidor-El socket funciona\n");
	}

	//Configuramos la estructura de la informacion del server
	my_addr.sin_family = AF_INET; //conjunto de direcciones a las que se puede
conectar
	my_addr.sin_port = htons(ELPUERTO); //Puerto por el que se conectara
	my_addr.sin_addr.s_addr = INADDR_ANY; //Recibe de cualquer de nuestras
interfaces
	//Ponemos en 0 o "null" el resto de la estructura
	memset(&(my_addr.sin_zero), '\0', 8);
	printf("El servidor esta en %s Con el puerto %d...\n", inet_ntoa(my_addr.sin_addr),
ELPUERTO);

	// Le intentamos asignar al socket sockfd una direccion de socket local
	if(bind(sockfd, (struct sockaddr *)&my_addr, sizeof(struct sockaddr)) == -1){
		perror("Server-bind() error");
		exit(1);
	} else{
		printf("Servidor-Se enlazo correctamente\n");
	}


	//Comprobamos si el servidor puede escuchar el socket
	if(listen(sockfd, BACKLOG) == -1){
		perror("Server-listen() error");
		exit(1);
	}else {
		printf("Servidor-Escucha...\n");
	}

	// obtenemos el tamaño de la estructura
	sin_size = sizeof(struct sockaddr_in);

	if (signal(SIGINT,manejador_senales) == SIG_ERR){
		printf("no puedo cachar SIGINT\n");
	perror("signal");
	}

	flag = sockfd;
	// Inicia el servidor
	while(1){
		// Intentamos crear la conexion con el cliente
```

```
                if((new_fd = accept(sockfd, (struct sockaddr *)&their_addr, &sin_size)) == -
1){
                        perror("Server-accept() error");
                        exit(1);
                }else{
                        printf("Servidor-Conexion exitosa...\n");
                        printf("Servidor: Conexion entrante de:  %s\n",
inet_ntoa(their_addr.sin_addr));
                        //Funcion de conexion
                        conexion(new_fd);
                }
        }
        printf("Cerrando servidor, adios\n");
        //Cerramos el descriptor de archivo de la conexion
        close(sockfd);
        return 0;
}
```

Pruebas

Ejecución:

Cliente

```
usuario@debian-10:~/Descargas/Proyecto/Nue
vo$ ./Cliente localhost
Client-El host remoto es: localhost
Client-Se logro crear socket...
Server-Usando localhost con el puerto 3490
...
Client-Conectado...
client>
```

Servidor

```
usuario@debian-10:~/Descargas/Proyecto/Nue
vo$ ./Servidor localhost
Servidor-El socket funciona
El servidor esta en 0.0.0.0 Con el puerto
3490...
Servidor-Se enlazo correctamente
Servidor-Escucha...
Servidor-Conexion exitosa...
Servidor: Conexion entrante de:  127.0.0.1
```

Comandos:

Clear

Cliente

```
usuario@debian-10:~/Descargas/Proyecto/Nue
vo$ ./Cliente localhost
Client-El host remoto es: localhost
Client-Se logro crear socket...
Server-Usando localhost con el puerto 3490
...
Client-Conectado...
client> clear
```

```
client>
```

Servidor

```
usuario@debian-10:~/Descargas/Proyecto/Nue
vo$ ./Servidor localhost
Servidor-El socket funciona
El servidor esta en 0.0.0.0 Con el puerto
3490...
Servidor-Se enlazo correctamente
Servidor-Escucha...
Servidor-Conexion exitosa...
Servidor: Conexion entrante de:  127.0.0.1
```

ls

Cliente

```
client> ls
ACS-Cliente.c
ACS-Servidor.c
Cliente
Servidor

client>
```

Servidor

```
Servidor- Recibe: ls

ACS-Cliente.c   Cliente
ACS-Servidor.c  Servidor
Servidor- Envia: ACS-Cliente.c
Servidor- Envia: ACS-Servidor.c
Servidor- Envia: Cliente
Servidor- Envia: Servidor
```

pwd

Cliente

```
client> pwd
/home/usuario/Descargas/Proyecto/Nuevo

client>
```

Servidor

```
Servidor- Recibe: pwd

/home/usuario/Descargas/Proyecto/Nuevo
Servidor- Envia: /home/usuario/Descargas/Proy
ecto/Nuevo
```

whoami

Cliente

```
client> whoami
usuario

client>
```

Servidor

```
Servidor- Recibe: whoami

usuario
Servidor- Envia: usuario
```

ls -l

Cliente

```
client> ls -l
total 52
-rw-rw-r-- 1 usuario usuario  3668 jun  8
 2022 ACS-Cliente.c
-rw-rw-r-- 1 usuario usuario  4672 jun  8
 2022 ACS-Servidor.c
-rwxr-xr-x 1 usuario usuario 17608 jun  8
01:22 Cliente
-rwxr-xr-x 1 usuario usuario 17816 jun  8
01:11 Servidor

client>
```

Servidor

```
Servidor- Recibe: ls -l

total 52
-rw-rw-r-- 1 usuario usuario  3668 jun  8
 2022 ACS-Cliente.c
-rw-rw-r-- 1 usuario usuario  4672 jun  8
 2022 ACS-Servidor.c
-rwxr-xr-x 1 usuario usuario 17608 jun  8
01:22 Cliente
-rwxr-xr-x 1 usuario usuario 17816 jun  8
01:11 Servidor
Servidor- Envia: total 52
Servidor- Envia: -rw-rw-r-- 1 usuario usua
rio  3668 jun  8  2022 ACS-Cliente.c
Servidor- Envia: -rw-rw-r-- 1 usuario usua
rio  4672 jun  8  2022 ACS-Servidor.c
Servidor- Envia: -rwxr-xr-x 1 usuario usua
rio 17608 jun  8 01:22 Cliente
Servidor- Envia: -rwxr-xr-x 1 usuario usua
rio 17816 jun  8 01:11 Servidor
```

Ejemplo de un comando erróneo

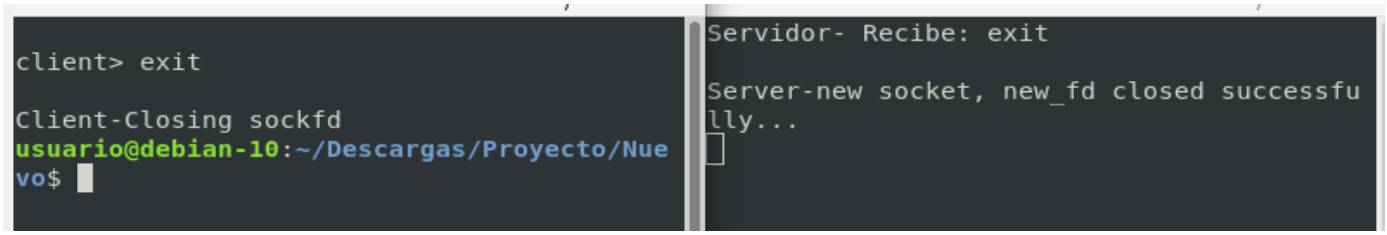
Cliente

Servidor

Exit

Cliente

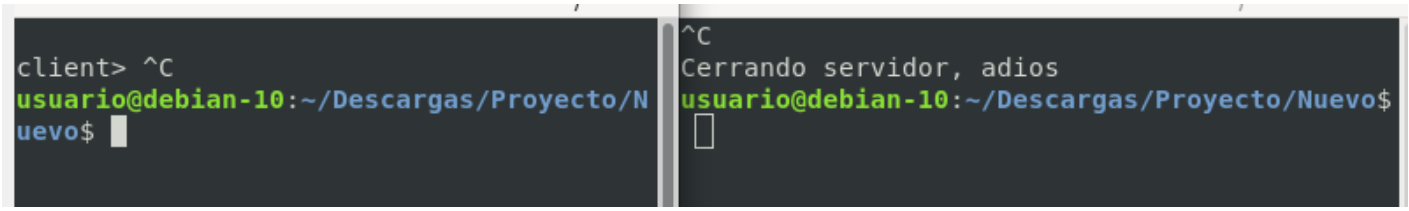
Servidor



Control + c

Cliente

Servidor



Conclusión

Con base a este proyecto logramos implementar nuestros conocimientos adquiridos a lo largo de semestre enfocándonos especialmente en los conocimientos adquiridos en el tema 4 en donde logramos crear de manera exitosa un Cliente-Servidor que ejecute comandos remotamente como ocurre con un cliente-Servidor SSH comercial.

Bibliografía.

[1] Deyimar A. (May 27, 2022) ¿Cómo funciona el SSH?, consultado de: https://www.hostinger.es/tutoriales/que-es-ssh#%C2%BFComo_funciona_SSH

[2] Red Hat Enterprise Linux 4: Manual de referencia, (Sin fecha), Protocolo SSH, consultado de: https://web.mit.edu/rhel-doc/4/RH-DOCS/rhel-rg-es-4/ch-ssh.html

[3] EcuRed (Sin fecha), Arquitectura Cliente Servidor, consultado de: https://www.ecured.cu/Arquitectura_Cliente_Servidor

[4] Andrés Schiaffarino. (12 Mar 2019). Modelo cliente servidor, consultado de: https://blog.infranetworking.com/modelo-cliente-servidor/