
# TRABAJO PRÁCTICO Nro. 2

## Virtualización y sistemas operativos avanzados - 2019

**Integrantes:**
Fleitas, Tomás Andrés - tomas_federal@hotmail.com
Re, Luis Santiago - lsantire2@gmail.com

## Comentarios

1. Para compilar o ejecutar el cliente y el servidor, de la implementación que se desee (RPC o Sockets), se debe ejecutar el comando “make” seguido del target deseado, desde la ruta correspondiente a la carpeta de la implementación que se quiera ejecutar. Los targets existentes, cuyo nombre creemos son suficientemente descriptivos, son:
    a. build_client
    b. build_server
    c. run_client
    d. run_server
2. Decidimos implementar las aplicaciones del cliente de una forma más interactiva a la planteada en el enunciado. En lugar de tener que ejecutarlo utilizando parámetros desde la línea de comandos, decidimos implementar un menú a través del cual el usuario puede interactuar con la aplicación.
3. Descripción del archivo de usuarios y contraseñas, llamado usuarios.txt:
    a. Línea 1: contiene un número con la cantidad de pares de usuario/contraseña cargados.
    b. Desde la línea 2: las líneas pares (2, 4, 6, …) tendrán un nombre de usuario y las líneas impares (3, 5, 7, …) tendrán la contraseña correspondiente al usuario de la línea anterior.

## RPC Client

```
#include "foo.h"

void getEstadisticas_client(char *host)
{
	CLIENT *clnt;
	estadisticas  *result_2;
	char *getestadisticas_1_arg;
#ifndef	DEBUG
	clnt = clnt_create (host, UIDPROG, UIDVERS, "udp");
	if (clnt == NULL) {
		clnt_pcreateerror (host);
		exit (1);
	}
#endif	/* DEBUG */
	result_2 = getestadisticas_1((void*)&getestadisticas_1_arg, clnt);
	if (result_2 == (estadisticas *) NULL) {
		printf("REVIENTA\n");
		clnt_perror (clnt, "call failed");
	}
	else{
		printf("\n\n\tRESULTADO");
		printf("\n\t----------------------------\n");
		printf("\tCantidad de autenticaciones correctas: %d\n",result_2->cantOk);
```

```
                printf("\tCantidad de autenticaciones incorrectas: %d\n",result_2->cantTotal-result_2->cantOk);
                printf("\t\tCantidad de fallos con usuario incorrecto: %d\n",result_2->cantUsuarioIncorrecto);
                printf("\t\tCantidad de fallos con password incorrecta: %d\n",result_2->cantPassIncorrecta);
                printf("\t\tCantidad de fallos inesperados:
%d\n",result_2->cantTotal-result_2->cantOk-result_2->cantUsuarioIncorrecto-result_2->cantPassIncorrecta);
        }
#ifndef   DEBUG
        clnt_destroy (clnt);
#endif    /* DEBUG */
}

void autenticar_client(char *host)
{
        CLIENT *clnt;
        int  *result_1;
        autenticacion  autenticar_1_arg;

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

        autenticar_1_arg.nombreUsuario = malloc(128);
        autenticar_1_arg.pass = malloc(128);

        printf("\nNombre de usuario: ");
        scanf("%s",autenticar_1_arg.nombreUsuario);
        printf("Password: ");
        scanf("%s",autenticar_1_arg.pass);

        result_1 = autenticar_1(&autenticar_1_arg, clnt);
        if (result_1 == (int *) NULL) {
                clnt_perror (clnt, "call failed");
        }
        else{
                printf("\n\n\tRESULTADO");
                printf("\n\t----------------------------\n");
                switch(*result_1)
                {
                case 0:
                        printf("\tUsuario y password correctos :-)\n");
                        break;
                case 1:
                        printf("\tUsuario y/o password incorrectos :-(\n");
                        break;
                case 2:
```

```
                        printf("\tOcurrio un error :-(\n");
                        break;
                default:
                        printf("\nERROR FATAL");
                        exit(0);
                }
        }
#ifndef   DEBUG
        clnt_destroy (clnt);
#endif     /* DEBUG */
}

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

        if (argc < 2) {
                printf ("usage: %s server_host\n", argv[0]);
                exit (1);
        }
        host = argv[1];
        do
        {
                printf("\n\tMENU");
                printf("\n\t----------------------------");
                printf("\n\n\t 1. AUTENTICAR");
                printf("\n\t 2. ESTADISTICAS");
                printf("\n\t 3. SALIR");
                printf("\n\n Enter Your Choice: ");
                scanf("%c",&choice);
                switch(choice)
                {
                case '1':
                        autenticar_client(host);
                        break;
                case '2':
                        getEstadisticas_client(host);
                   break;
                case '3':
                   exit(0);
                default:
                        printf("\nINVALID SELECTION...Please try again");
                }
                getchar();
                printf("\nPresione ENTER para continuar\n");
                getchar();
                //getc();
        }while(1);
```

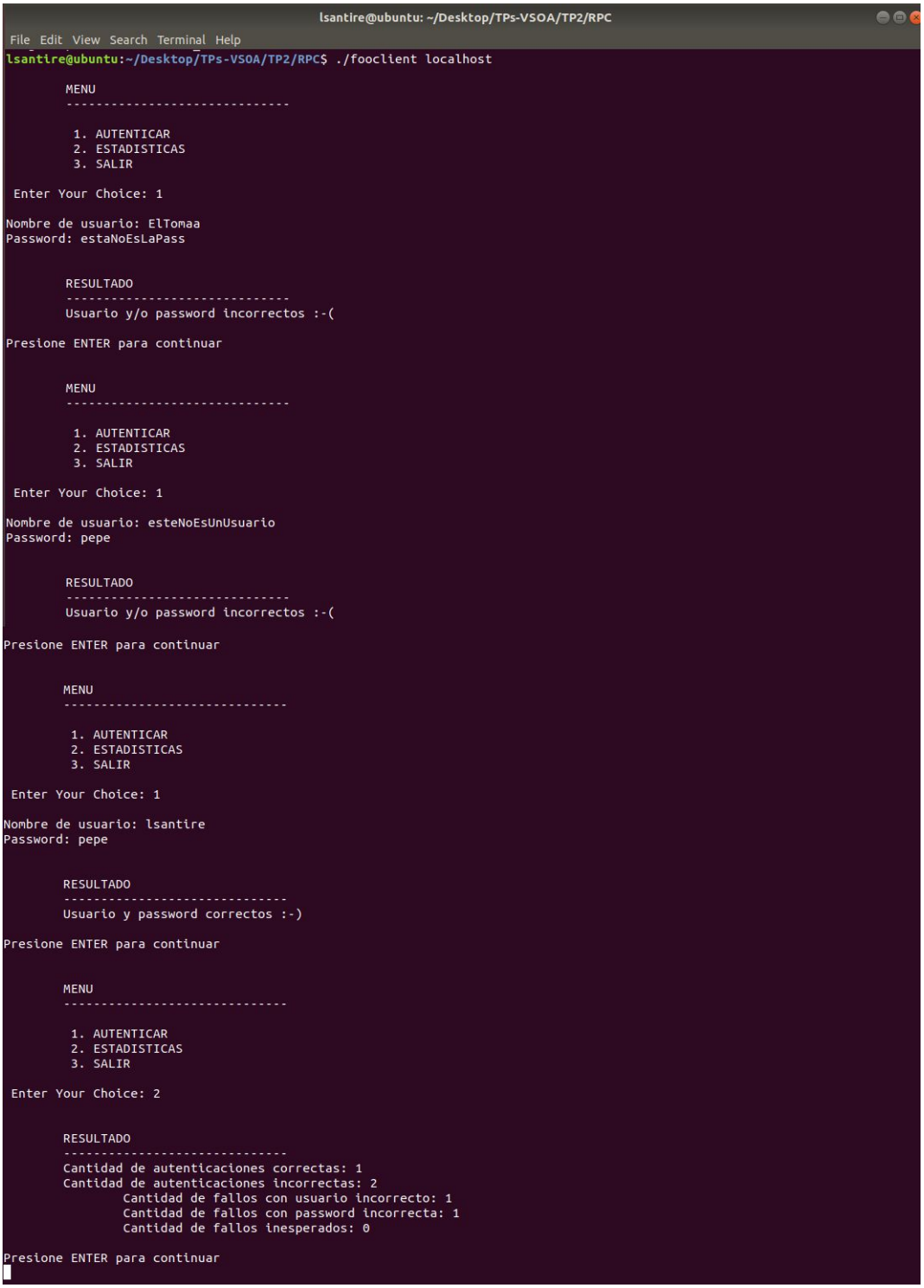
```
	return 0;
}
```

## RPC Server

```
#include "foo.h"

int cantOk;
int cantUsuarioIncorrecto;
int cantPassIncorrecta;
int cantTotal;

int * autenticar_1_svc(autenticacion *argp, struct svc_req *rqstp)
{
	static int result;
	int n;
	char *nombreUsuario,*pass;
	int usuarioOk=0,i=0;

	nombreUsuario = malloc(128);
	pass = malloc(128);

	cantTotal++;

	if(freopen("usuarios.txt","r",stdin)==NULL)
	{
		result=2;
		return &result;
	}
	scanf("%d",&n);
	while(i<n)
	{
		scanf("%s",nombreUsuario);
		scanf("%s",pass);
		if(strcmp(nombreUsuario,argp->nombreUsuario)==0)
		{
			usuarioOk=1;
			if(strcmp(pass,argp->pass)==0)
			{
				cantOk++;
				result=0;
				return &result;
			}
		}
		i++;
	}
	if(usuarioOk) cantPassIncorrecta++;
	else cantUsuarioIncorrecto++;
```

```
	result=1;
	return &result;
}

estadisticas * getestadisticas_1_svc(void *argp, struct svc_req *rqstp)
{
	static estadisticas  result;

	result.cantOk=cantOk;
	result.cantUsuarioIncorrecto=cantUsuarioIncorrecto;
	result.cantPassIncorrecta=cantPassIncorrecta;
	result.cantTotal=cantTotal;

	return &result;
}
```

## Salida RPC


lsantire@ubuntu: ~/Desktop/TPs-VSOA/TP2/RPC
File Edit View Search Terminal Help
lsantire@ubuntu:~/Desktop/TPs-VSOA/TP2/RPC$ ./fooclient localhost
MENU
1. AUTENTICAR
2. ESTADISTICAS
3. SALIR
Enter Your Choice: 1
Nombre de usuario: ElTomaa
Password: estaNoEsLaPass
RESULTADO
Usuario y/o password incorrectos :-(
Presione ENTER para continuar
MENU
1. AUTENTICAR
2. ESTADISTICAS
3. SALIR
Enter Your Choice: 1
Nombre de usuario: esteNoEsUnUsuario
Password: pepe
RESULTADO
Usuario y/o password incorrectos :-(
Presione ENTER para continuar
MENU
1. AUTENTICAR
2. ESTADISTICAS
3. SALIR
Enter Your Choice: 1
Nombre de usuario: lsantire
Password: pepe
RESULTADO
Usuario y password correctos :-)
Presione ENTER para continuar
MENU
1. AUTENTICAR
2. ESTADISTICAS
3. SALIR
Enter Your Choice: 2
RESULTADO
Cantidad de autenticaciones correctas: 1
Cantidad de autenticaciones incorrectas: 2
Cantidad de fallos con usuario incorrecto: 1
Cantidad de fallos con password incorrecta: 1
Cantidad de fallos inesperados: 0
Presione ENTER para continuar

## Socket Client

```
#define PORT 8080
void autenticar_client(int sock)
{
	char message[128]="1|";
	char nombreUsuario[128];
	char pass[128];
	int valread,result;
	char buffer[1024] = {0};

	printf("\nNombre de usuario: ");
	scanf("%s",nombreUsuario);
	printf("Password: ");
	scanf("%s",pass);

	strcat(message,nombreUsuario);
	strcat(message,"|");
	strcat(message,pass);
	send(sock , message , strlen(message) , 0 );
	valread = read( sock , buffer, 1024);
	result = atoi(buffer);
		printf("\n\n\tRESULTADO");
		printf("\n\t----------\n");
		switch(result)
		{
		case 0:
			printf("\tUsuario y password correctos :-)\n");
			break;
		case 1:
			printf("\tUsuario y/o password incorrectos :-(\n");
			break;
		case 2:
			printf("\tOcurrio un error :-(\n");
			break;
		default:
			printf("\nERROR FATAL");
			exit(0);
		}
}
void getEstadisticas_client(int sock)
{
	char message[128]="2";
	int valread;
	char buffer[1024] = {0};

	send(sock , message , strlen(message) , 0 );
	valread = read(sock, buffer, 1024);
```

```
        int cantOk;
        int cantUsuarioIncorrecto;
        int cantPassIncorrecta;
        int cantTotal;
        int pos = 0;
        int varActual = 0;
        char aux[128];
        for (int i = 0;buffer[i] != '\0'; ++i){
                if (buffer[i] == '|')
                {
                        aux[pos] = '\0';
                        switch(varActual){
                        case 0:
                                cantOk = atoi(aux);
                                break;
                        case 1:
                                cantUsuarioIncorrecto = atoi(aux);
                                break;
                        case 2:
                                cantPassIncorrecta = atoi(aux);
                                break;
                        }
                        varActual++;
                        pos = 0;
                }else{
                        aux[pos++] = buffer[i];
                }
        }
        aux[pos] = '\0';
        cantTotal = atoi(aux);
        printf("\n\n\tRESULTADO");
        printf("\n\t----------------------------\n");
        printf("\tCantidad de autenticaciones correctas: %d\n",cantOk);
        printf("\tCantidad de autenticaciones incorrectas: %d\n",cantTotal-cantOk);
        printf("\t\tCantidad de fallos con usuario incorrecto: %d\n",cantUsuarioIncorrecto);
        printf("\t\tCantidad de fallos con password incorrecta: %d\n",cantPassIncorrecta);
        printf("\t\tCantidad de fallos inesperados:
%d\n",cantTotal-cantUsuarioIncorrecto-cantPassIncorrecta-cantOk);
}

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

   sockaddr_in address,serv_addr;
   int sock = 0;
   char choice;
   if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0)
   {
      printf("\n Socket creation error \n");
      return -1;
```

```
    }
    memset(&serv_addr, '0', sizeof(serv_addr));
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_port = htons(PORT);
    // Convert IPv4 and IPv6 addresses from text to binary form
    if(inet_pton(AF_INET, "127.0.0.1", &serv_addr.sin_addr)<=0)
    {
        printf("\nInvalid address/ Address not supported \n");
        return -1;
    }
    if (connect(sock, (sockaddr *)&serv_addr, sizeof(serv_addr)) < 0)
    {
        printf("\nConnection Failed \n");
        return -1;
    }
//==================MENUUUUU=======================//
            do{
                    printf("\n\tMENU");
                    printf("\n\t----------------------------");
                    printf("\n\n\t 1. AUTENTICAR");
                    printf("\n\t 2. ESTADISTICAS");
                    printf("\n\t 3. SALIR");
                    printf("\n\n Enter Your Choice: ");
                    scanf("%c",&choice);
                    switch(choice)
                            {
                            case '1':
                                    autenticar_client(sock);
                                    break;
                            case '2':
                                    getEstadisticas_client(sock);
                                     break;
                            case '3':
                       exit(0);
                            default:
                            printf("\nINVALID SELECTION...Please try again");
                            }
                    getchar();
                    printf("\nPresione ENTER para continuar\n");
                    getchar();
                    //getc();
            }while(1);

    return 0;
}
```

## Socket Server

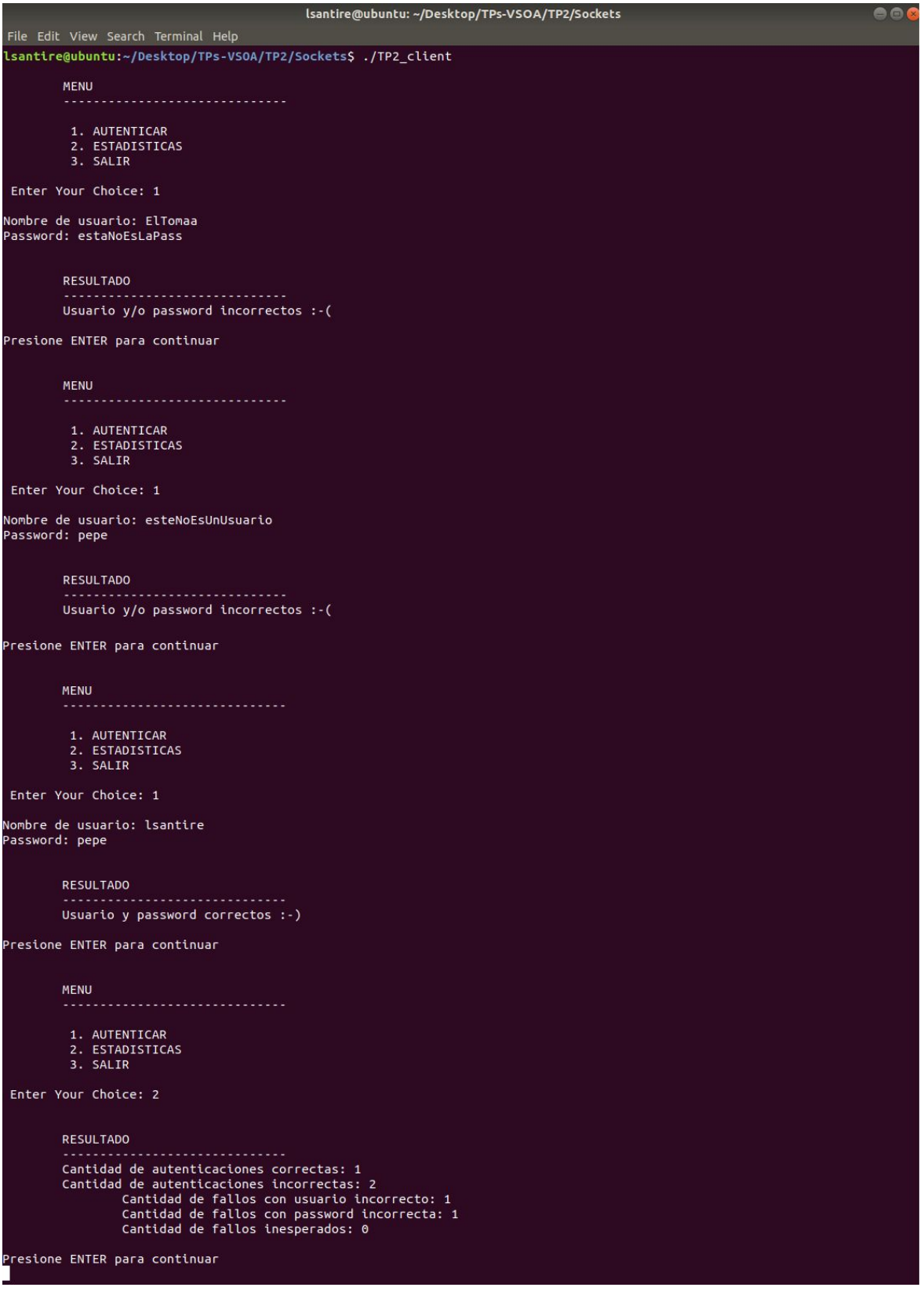
```
#define PORT 8080
int cantOk;
int cantUsuarioIncorrecto;
int cantPassIncorrecta;
int cantTotal;

void autenticar_1_svc(char buffer[1024], int new_socket)
{
   static int result;
   int n;
   char nombreUsuario[128],pass[128];
   int usuarioOk=0,i=0;
   char new_buffer[1024];

   cantTotal++;

   if(freopen("usuarios.txt","r",stdin)==NULL)
   {
      send(new_socket , "2" , 1 , 0 );
      return;
   }
   scanf("%d",&n);

   char nombreUsuarioAuth[128];
   char passAuth[128];

   int posNombreAuth = 0;
   int posPassAuth = -1;

   for (int i = 2;buffer[i] != '\0'; ++i)
   {
      if (buffer[i] == '|')
      {
         posPassAuth = 0;
      }else{
         if (posPassAuth != -1)
         {
            passAuth[posPassAuth++] = buffer[i];
         }else{
            nombreUsuarioAuth[posNombreAuth++] = buffer[i];
         }
      }

   }
```

```
    if (posPassAuth == -1)
    {
       posPassAuth++;
    }

    passAuth[posPassAuth] = '\0';
    nombreUsuarioAuth[posNombreAuth] = '\0';

    while(i<n)
    {
       scanf("%s",nombreUsuario);
       scanf("%s",pass);

       if(strcmp(nombreUsuario,nombreUsuarioAuth)==0)
       {
          usuarioOk=1;
          if(strcmp(pass,passAuth)==0)
          {
             cantOk++;
             send(new_socket , "0" , 1 , 0 );
             return;
          }
       }
       i++;
    }
    if(usuarioOk) cantPassIncorrecta++;
    else cantUsuarioIncorrecto++;
    send(new_socket , "1" , 1 , 0 );
}
int toString(int x, char str[]){
    if(x<10){
            str[0]=x+'0';
            return 1;
    }
    int aux = toString(x/10, str);
    str[aux] = x%10+'0';
}

void getestadisticas_1_svc(int new_socket)
{
    char aux[128];
    char new_buffer[1024]="";
    toString(cantOk,aux);
    strcat(new_buffer,aux);
    strcat(new_buffer,"|");
    toString(cantUsuarioIncorrecto,aux);
    strcat(new_buffer,aux);
    strcat(new_buffer,"|");
```

```
    toString(cantPassIncorrecta,aux);
    strcat(new_buffer,aux);
    strcat(new_buffer,"|");
    toString(cantTotal,aux);
    strcat(new_buffer,aux);
    send(new_socket , new_buffer , strlen(new_buffer) , 0 );
}


int main(){
            int server_fd,new_socket;
            int opt = 1;
            sockaddr_in address;
            int addrlen = sizeof(address);
   // Creating socket file descriptor
    if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0)
    {
       perror("socket failed");
       exit(EXIT_FAILURE);
    }

   // Forcefully attaching socket to the port 8080
    if (setsockopt(server_fd, SOL_SOCKET, SO_REUSEADDR | SO_REUSEPORT,
                                    &opt, sizeof(opt)))
    {
       perror("setsockopt");
       exit(EXIT_FAILURE);
    }
    address.sin_family = AF_INET;
    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);
    }
char buffer[1024] = {0};
                        if (listen(server_fd, 3) < 0)
            {
                        perror("listen");
            exit(EXIT_FAILURE);
                        }
                        if ((new_socket = accept(server_fd, (struct sockaddr *)&address,  (socklen_t*)&addrlen))<0)
              {
                  perror("accept");
                  exit(EXIT_FAILURE);
```

```
                }

            while(true)
            {

                int valread = read( new_socket , buffer, 1024);

        if (buffer[0] == '1')
        {
          autenticar_1_svc(buffer,new_socket);
        }else{
          getestadisticas_1_svc(new_socket);
        }

            }
}
```

## Salida socket:



Fleitas, Tomás Andrés
Re, Luis Santiago

## Conclusiones

Después de haber trabajado con ambas tecnologías (RPC y Sockets), no observamos diferencias en cuanto a la eficiencia de ambas. Esto puede ser porque realmente no hay diferencias, o porque no llegaron a notarse debido a que tanto cliente como servidor corrían en la misma máquina y las cargas de trabajo y mensajes eran bajas.

Pero sí podemos decir que RPC brinda algunas facilidades que al trabajar con Sockets no tuvimos. Por un lado, porque RPC genera de forma automática el código relacionado a las conexiones y comunicaciones. Y por otro, porque también brinda la posibilidad de usar estructuras personalizadas para el envío de datos. En cambio al utilizar Sockets, tuvimos que construir los mensajes por nuestra cuenta, diseñando algún formato en especial para luego poder parsearlo de forma correcta en el receptor.