Trabajo Práctico N° 1
Capa de Aplicación.

Fecha de entrega: 6 de Junio.

Programación Cliente / Servidor utilizando la interfaz sockets.

Integrantes:

-Julian Gelvez

-Macarena Belen Garcia Arcija

-Laura Santander

Código base.

```
/*
PrimerServidorTCP.c
   Servicio: Las cadenas de texto recibidas de un Cliente son
             enviadas a la salida estándar.
   Nota: Por simplicidad del código no se realiza ningún tipo de
         control de errores. No obstante el servidor es totalmente
         funcional.
*/
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <string.h>
#include <stdlib.h>
#include <stdio.h>

#define PORTNUMBER  12345

int main(void){
    char buf[10];
    int s, n, ns, len;
    struct sockaddr_in direcc;

    s = socket(AF_INET, SOCK_STREAM, 0);

    bzero((char *) &direcc, sizeof(direcc));
    direcc.sin_family = AF_INET;
    direcc.sin_port = htons(PORTNUMBER);
    direcc.sin_addr.s_addr = htonl(INADDR_ANY);

    len = sizeof(struct sockaddr_in);
    bind(s, (struct sockaddr *) &direcc, len);
    listen(s, 5);
    ns = accept(s, (struct sockaddr *) &direcc, &len);

    while ((n = recv(ns, buf, sizeof(buf), 0)) > 0)
        write(1, buf, n);
```

```
    close(ns); close(s);
    exit(0);
}
```

Tareas a Realizar.

Abra una terminal, compile el servidor (*#> gcc -o server Server.c*) y luego ejecutelo (*#> ./server*). Abra otra terminal y conectese con el servidor usando el cliente *telnet* (*#> telnet localhost 12345*). Verifique el funcionamiento del mismo y luego cierre la sesión de *telnet*.

1) **¿Que sucede con el Servidor?¿Por qué?**

El servidor finaliza su ejecución inmediatamente después de que el cliente cierra la sesión, ya que al no ser iterativo, ni concurrente, este(el servidor) no permanecerá escuchando y atendiendo a aquellos clientes que deseen conectarse luego de que el primero lo haya hecho o haya finalizado su conexión.

2) Modifique el servidor para que no finalice y guardelo como *Server1.c.*
3) Modifique *Server1.c* para que el servicio que presta se implemente en una función separada del cuerpo principal del programa (*main()*) y guardelo como *Server2.c*
4) Modifique y *Server2.c* para que preste el servicio de ***echo*** (vea RFC862) y guardelo como *Server3.c*
5) Modifique *Server3.c* para que preste el servicio de ***character generator*** (vea RFC864) y guardelo como *Server4.c*
6) Desarrolle un cliente que se comunique con los *Server3.c y Server4.c* ; y llamelo `Cliente1.c`.
7) Transformar el *Server3.c*, en un servidor concurrente usando fork() y guardelo como *Server5.c.*
8) Modificar el *Server5.c* para que espere por la finalización de sus hijos para evitar los procesos zombies y guardelo como *Server6.c.*
9) Modificar el *Server6.c* para que cambie el servicio por uno que devuelva la suma de dos parámetros enteros o una leyenda de error en parámetros si estos no son enteros y guardelo como *Server7.c*. Tips: Usar la función sscanf() y snprintf(). Cree el cliente correspondiente y guardelo como `Cliente2.c`.. No acepta números mayores a 2147483647.
10) Modificar el *Server7.c* para que utilice estructuras en vez de texto plano y guardelo como *Server8.c*. Cree el cliente correspondiente y guardelo como `Cliente3.c`.}
11) *Transformar el Server3.c, en un servidor iterativo de Eco con UDP y guardelo como Server9.c. Cree el cliente correspondiente y guardelo como Cliente4.c.*
12) Dado el siguiente cliente en Java, compilarlo y usarlo para conectarse con el *Server3.c*. ¿Por qué pueden interactuar sin problemas?

```
import java.net.*;
import java.io.*;

public class ClienteTCP {
   public static void main(String args[])  {
   // args proporciona el Nombre/IP de Server Destino y el mensaje
         if (args.length != 2)  {
      System.out.println("2 argumentos: servidor y mensaje");
```

```
            System.exit(1);
         }
         try {
            int puertoServicio = 12345;
            Socket s = new Socket(args[0], puertoServicio);
            DataInputStream entrada = new DataInputStream(s.getInputStream());
            DataOutputStream salida = new DataOutputStream(s.getOutputStream());
            salida.writeUTF(args[1]);
            String datos = entrada.readUTF();
            System.out.println("Recibido: " + datos);
            s.close();
         }catch( Exception e) {
            e.printStackTrace();
         }
   }
}
```

El cliente Java puede conectarse con el servidor C ya que si bien ambos utilizan una tecnología diferente en la redacción del código, ambos utilizan el mismo protocolo de comunicación

13. Desarrolle el *`Server3.c`* en Java. Indique, según su experiencia, cuales son las principales ventajas / desventajas del uso de las dos tecnologías para el desarrollo de Servidores.

	C	**JAVA**
VENTAJAS	Desarrollar el servidor se vuelve mas directo en sus sentencias.	El desarrollo del servidor es mas intuitivo, lo que permite entender mejor el funcionamiento del mismo.
	Posee una menor cantidad de líneas de codigo en comparacion con el desarrollo en Java.	Lleva un control riguroso con los errores, obligando el uso de las excepciones en sus metodos y desarrollo.
		Al ser desarrollado en java se obtiene(logra) la portabilidad del programa

DESVENTAJAS	Su funcionamiento no es tan intuitivo, logrando, asi, que se vuelva engorroso de entender.	El proceso de programarlo es mas complejo en comparación a c, provocando obtener una mayor cantidad de líneas de código.