Instituto Politécnico Nacional

Escuela Superior de Cómputo

Ingeniería en Sistemas Computacionales

Aplicaciones para Comunicaciones en Red

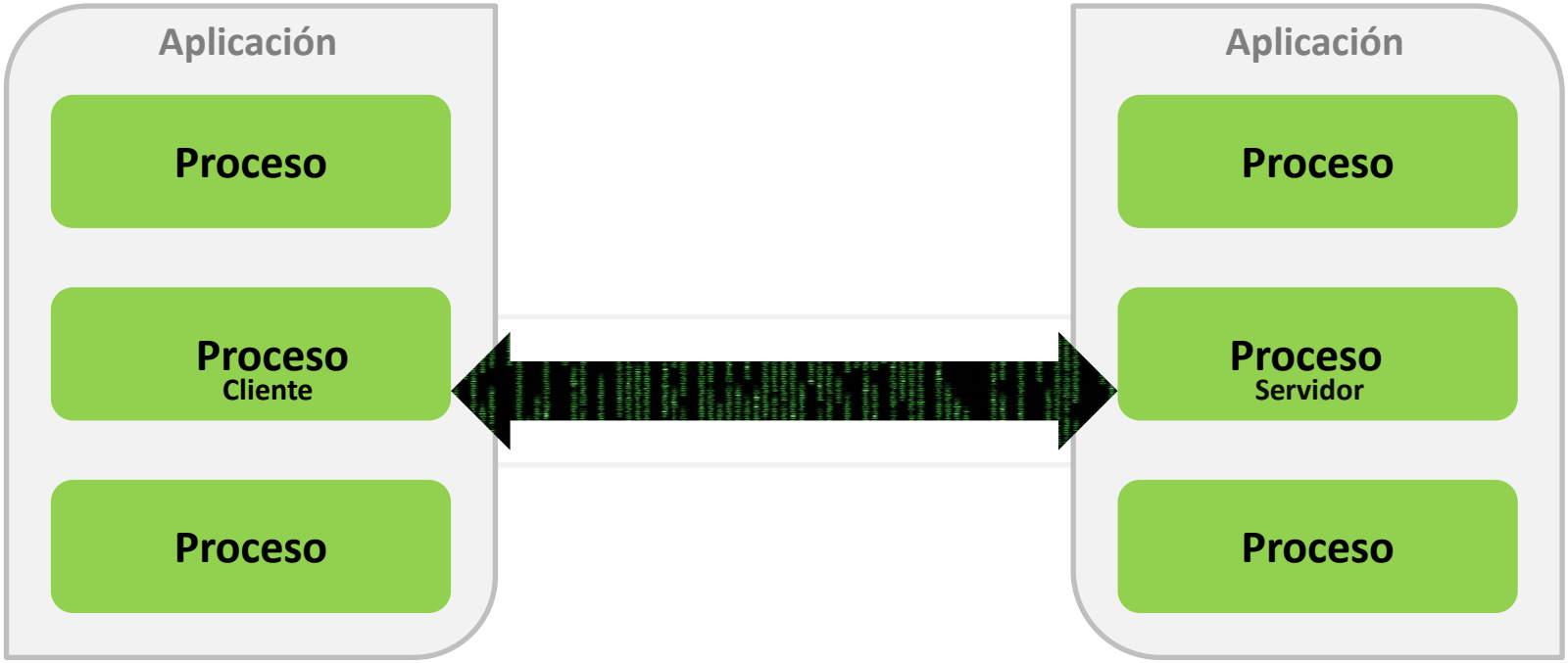
Sockets de flujo bloqueantes

M. en C. Sandra Ivette Bautista Rosales

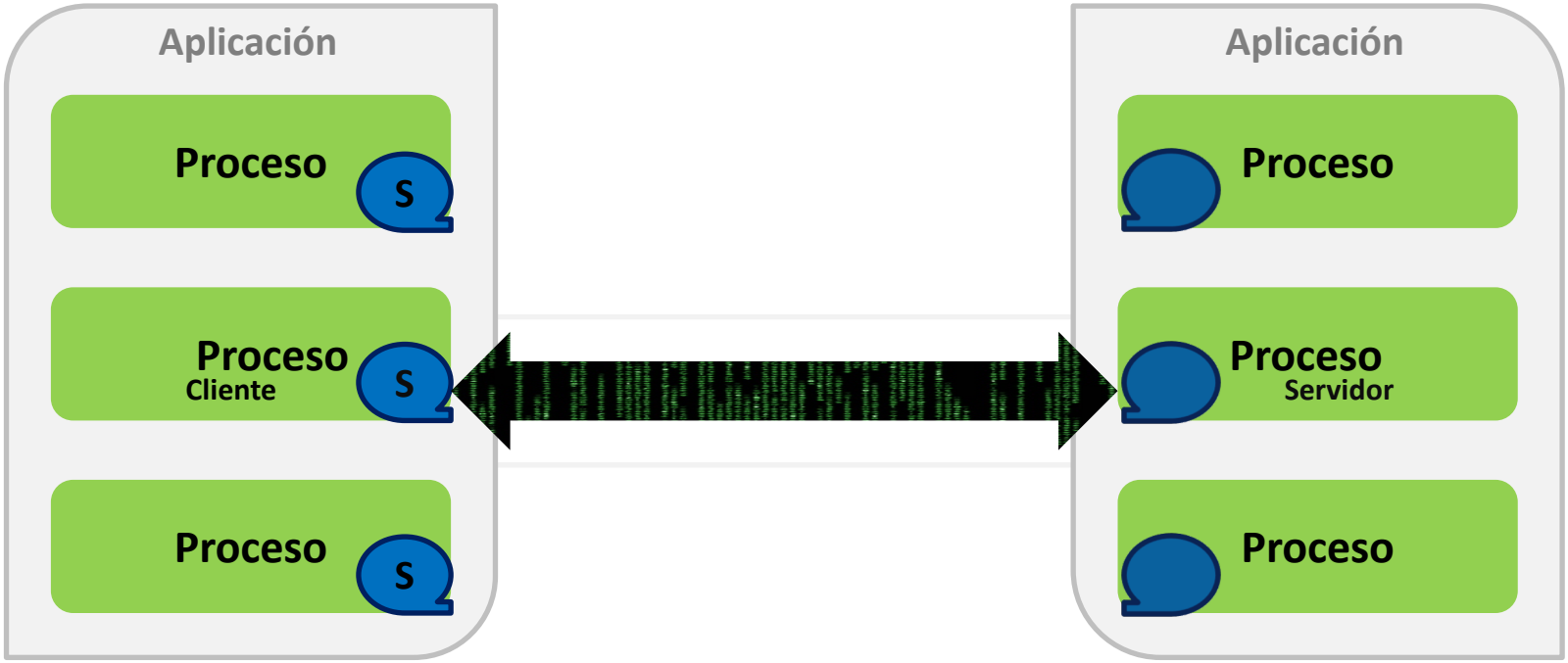
¿Qué es un socket?

¿Qué es un socket?

¿Qué es un socket?

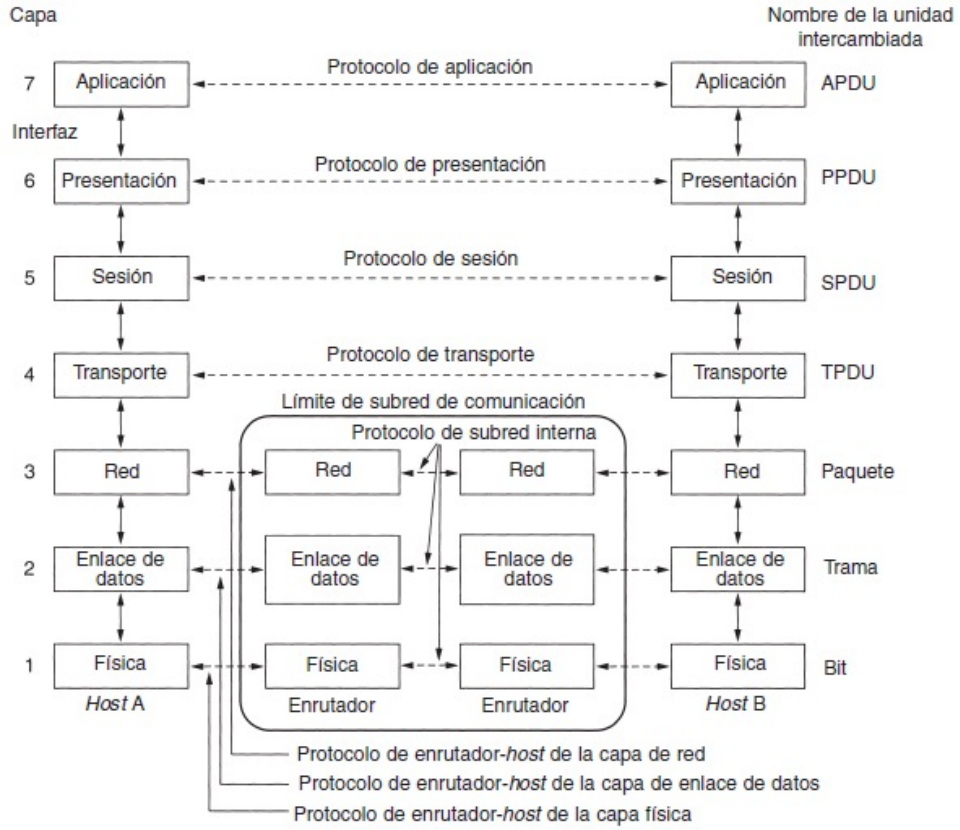


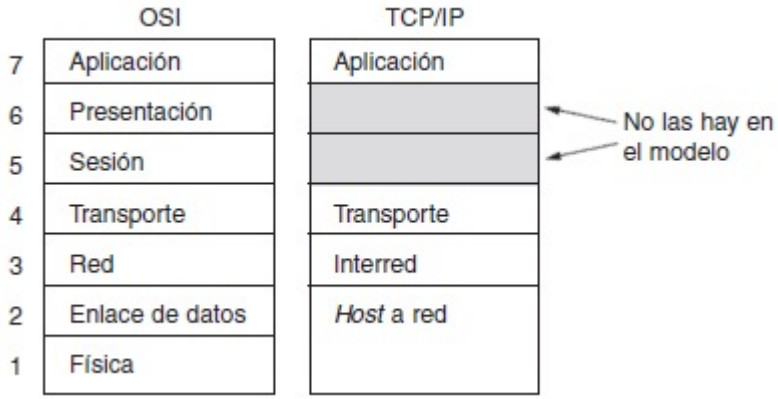
¿Qué es un socket?



¿Qué es un socket?

- Aparecieron a principios de los 80's con el sistema UNIX de Berkeley
  - Con el fin de proporcionar un medio de comunicación a los procesos.
- Es un punto de comunicación por el cual un proceso puede emitir o recibir información
- En el interior de un proceso, un socket se identifica por un **descriptor** de la misma naturaleza que los que identifican a los archivos
  - SD: Es un número entero asociado a un fichero (archivo) abierto (conexión de red, una terminal, etc.)

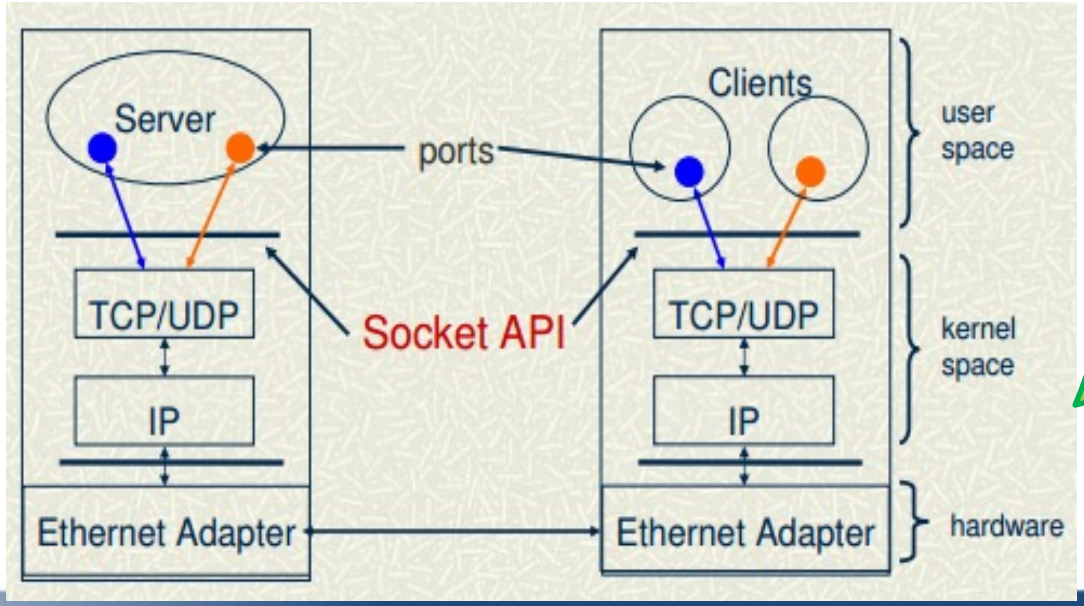

Capa
Nombre de la unidad intercambiada
7
Aplicación
Protocolo de aplicación
Aplicación
APDU
Interfaz
6
Presentación
Protocolo de presentación
Presentación
PPDU
5
Sesión
Protocolo de sesión
Sesión
SPDU
4
Transporte
Protocolo de transporte
Transporte
TPDU
Límite de subred de comunicación
Protocolo de subred interna
3
Red
Red
Red
Red
Paquete
2
Enlace de datos
Enlace de datos
Enlace de datos
Enlace de datos
Trama
1
Física
Física
Física
Física
Bit
Host A
Enrutador
Enrutador
Host B
Protocolo de enrutador-host de la capa de red
Protocolo de enrutador-host de la capa de enlace de datos
Protocolo de enrutador-host de la capa física


OSI
TCP/IP
7
Aplicación
Aplicación
6
Presentación
5
Sesión
No las hay en el modelo
4
Transporte
Transporte
3
Red
Interred
2
Enlace de datos
Host a red
1
Física

Interfaz de Sockets

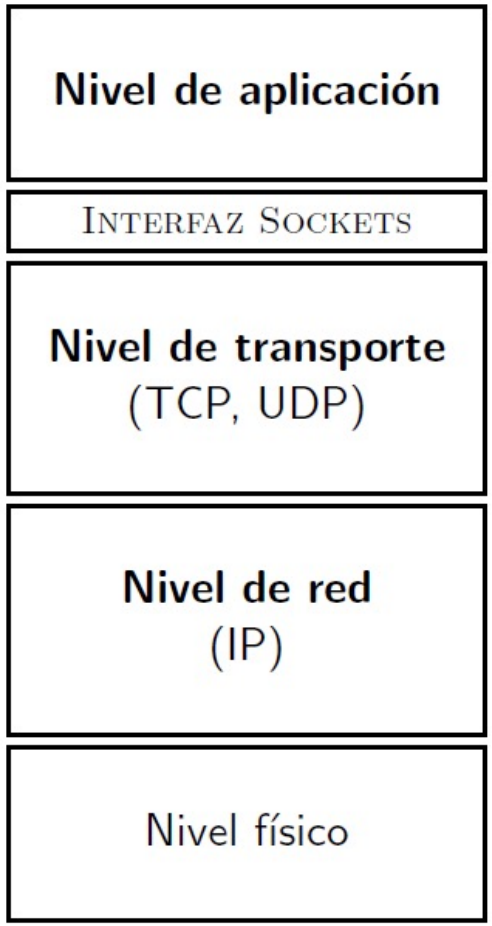
- Socket: Interfaz de programación (API) sobre el nivel de transporte
  - Abstracción que facilita al programador el acceso a los servicios y recursos del nivel de transporte
  - Conjunto de subrutinas, funciones y procedimientos (o métodos) que ofrece cierta biblioteca para ser utilizado por otro software
  - Ofrece un servicio punto a punto entre emisor y receptor

Los procesos de las aplicaciones residen en el espacio de usuario



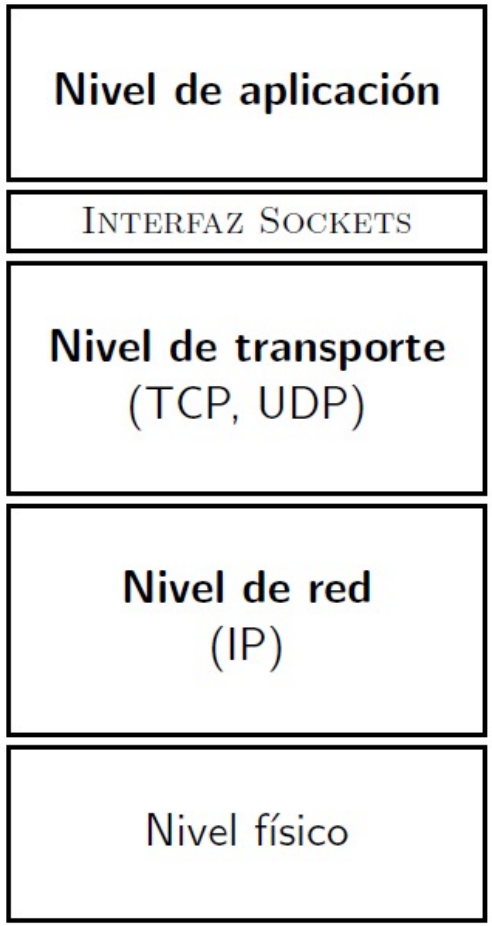
Los procesos de los protocolos de transporte forman parte del S.O.

Pila de protocolos TCP/IP



- Nivel de transporte:
  - Servicio de envío de datos extremo a extremo
  - Hace uso de servicios del nivel de red (IP)
- Protocolo TCP:
  - Servicio orientado a conexión, fiable, ordenado, con control de flujo y errores
    - Requiere establecimiento previo de una conexión entre ambos extremos
    - Ofrece flujo permanente entre los extremos en ambas direcciones
    - Controla la recepción en orden, completa y sin errores, gestionando el reenvío de paquetes perdidos

Pila de protocolos TCP/IP



- Protocolo UDP:
  - servicio no orientado a conexión, no fiable y sin control de flujo y errores
  - Cada mensaje UDP (datagrama) es independiente y se trata de forma aislada
  - No se garantiza la entrega de datagramas enviados ni que estos lleguen en orden

Primitivas para la utilización de sockets

- Secuencia de primitivas para la utilización de sockets que el cliente y el servidor tienen que usar para ambos tipos de servicio:
  - Orientado a conexión
  - No orientado a conexión

Diagrama de primitivas de sockets

Dominio de un socket

- Representa una familia de protocolos
- Una familia o dominio de la conexión, agrupa todos aquellos sockets que comparten características comunes y con los cuáles se puede establecer una comunicación.
- Un socket está asociado a un dominio desde su creación.
- Existen diferentes dominios de comunicación, algunos de los formatos reconocidos actualmente son:

Dominio de un socket

- AF_UNIX, AF_LOCAL
- AF_INET
- AF_INET6
- AF_IPX
- AF_NETLINK
- AF_X25
- AF_AX25
- AF_ATMPVC
- AF_APPLETALK
- AF_PACKET
- AF_ALG

Los servicios de sockets son independientes del dominio

Tipos de sockets

- Define las propiedades de las comunicaciones en las que se ve envuelto un socket, esto es, el tipo de comunicación que se puede dar entre cliente y servidor.
- Estas pueden ser:
  - Fiabilidad de la transmisión: Ningún dato transmitido se pierde
  - Conservación del orden de los datos. Los datos llegan en el orden en que han sido emitidos.
  - No duplicación de datos. Solo llega a destino un ejemplar de cada dato emitido

Tipos de sockets

- El «modo conectado» en la comunicación
- Envío de mensajes urgentes

Ejemplos:

- SOCK_STREAM
- SOCK_DGRAM
- SOCK_RAW
- SOCK_SEQPACKET
- SOCK_RDM
- SOCK_PACKET

Interfaz de Sockets

- **Puertos:** identificadores usados para asociar los datos entrantes a un proceso concreto de la máquina.
  - Usados tanto en TCP como en UDP
  - Números de 16 bits
  - 0-1023: reservados por convenio ("puertos bien conocidos")
    - Asignados a los servidores de servicios básicos

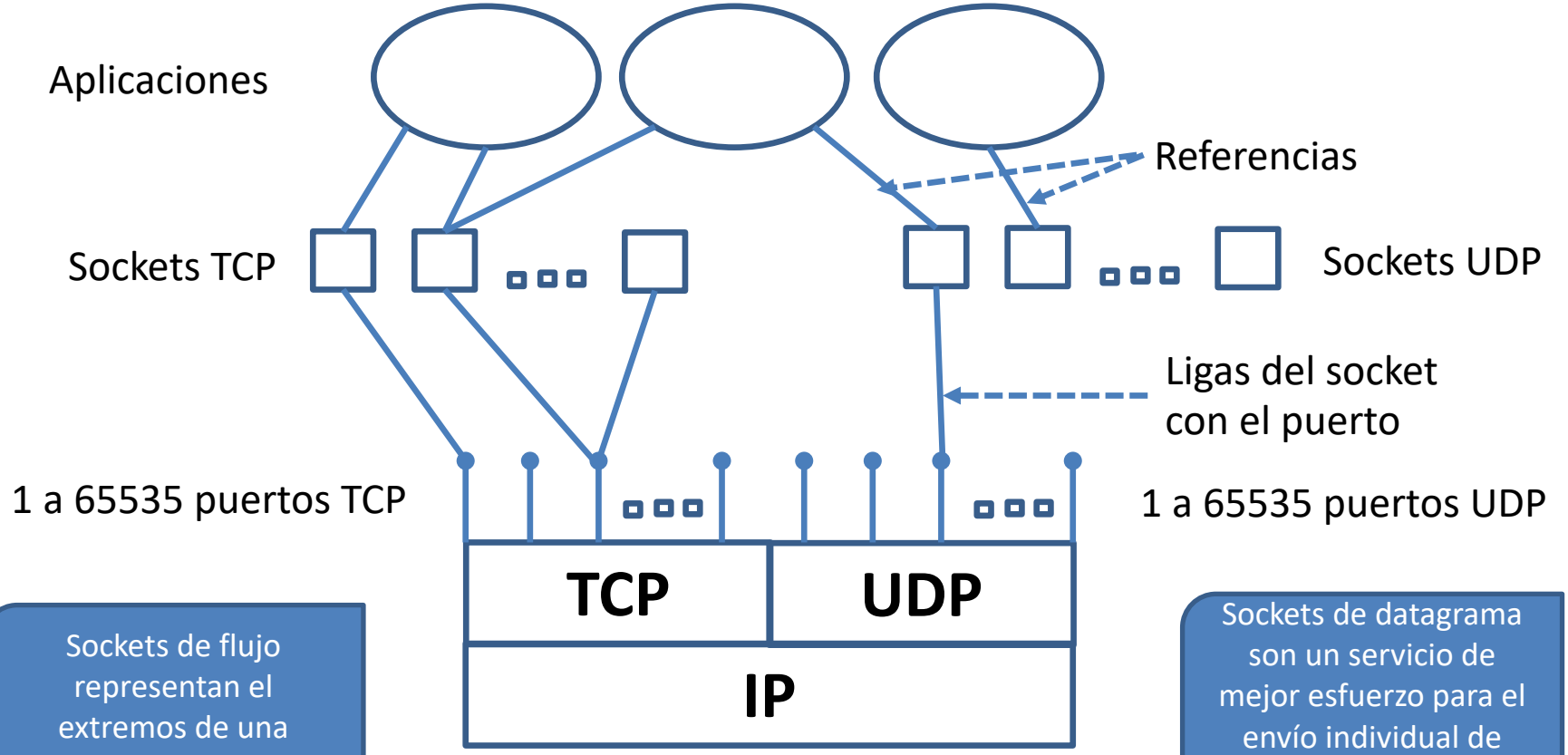
Servicio	Puerto	Servicio	Puerto
ftp	21	dns	53
telnet	22	http	80
ssh	23	pop3	110
smtp	25	https	443

  - 1024-65535: uso libre
    - Son los usados con los clientes al establecer conexiones
    - Suelen asignarse de forma aleatoria

Interfaz de Sockets

- **Direccionamiento:** Para comunicarse con otro proceso usando sockets debe conocerse:
  1. *Dirección IP* (32 bits) de la máquina donde se ejecuta el proceso
     - Alternativamente, su nombre para consultar servicio de nombre DNS (traducción dominio IP)
  2. *No. De puerto* (TCP o UDP) que utiliza el proceso en su máquina
- Por lo tanto, la interfaz de sockets no ofrece transparencia de localización

Relación lógica entre aplicaciones sockets, protocolos y puertos



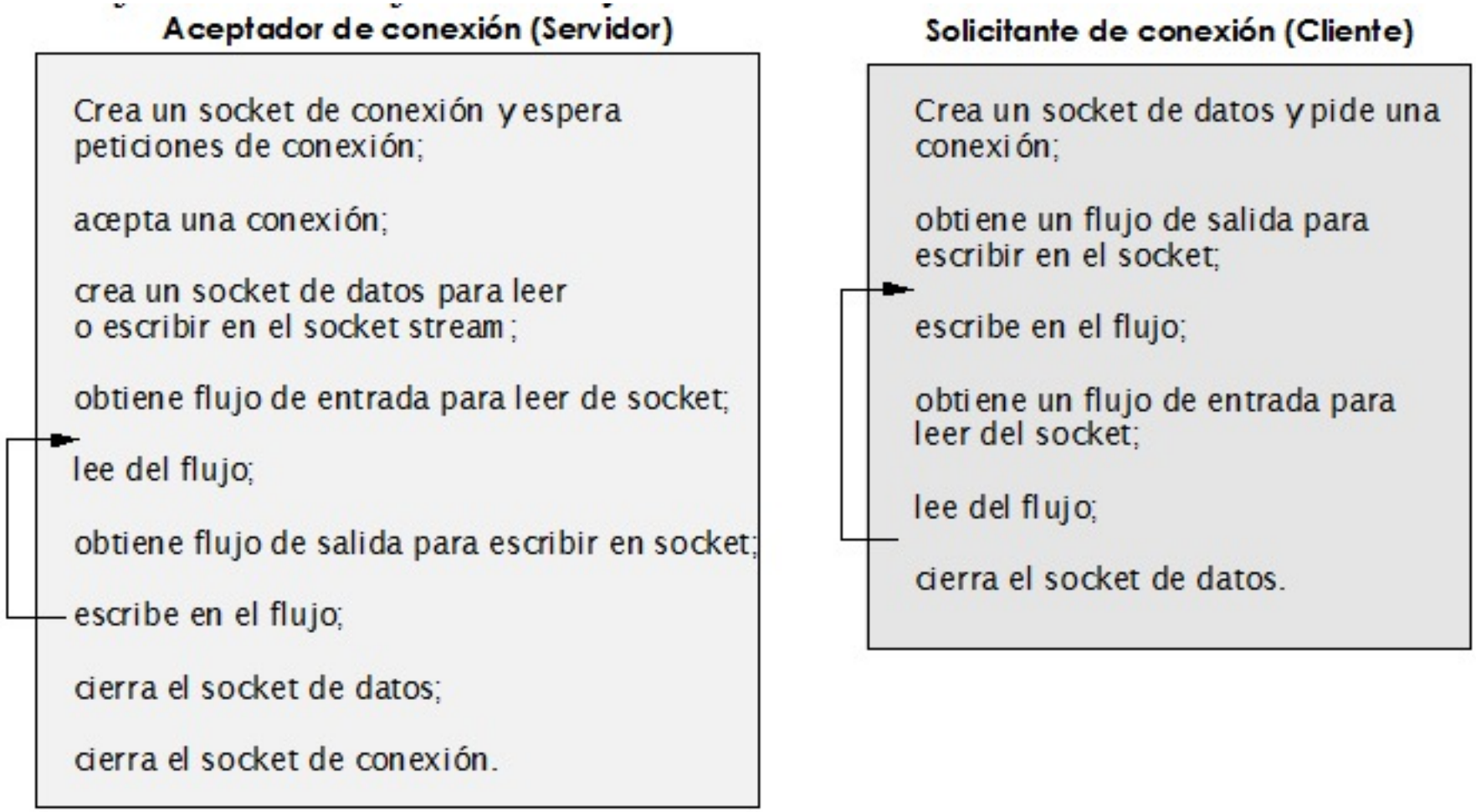
Sockets bloqueantes y no bloqueantes

- Realmente no se trata de sockets diferentes, son sólo opciones para las formas en las que trabajan.
- Los sockets bloqueantes son aquellos que se quedan esperando sin poder continuar con la ejecución hasta que existe información para establecer una conexión, leer o escribir un mensaje.
- Los sockets no bloqueantes interrogan si hay datos para procesar y en caso de que no se así, continúan con la ejecución del código.
- El socket no bloqueante se define modificando sus opciones (en el caso de C) o utilizando algún mecanismo, por ejemplo, hilos.

Socket orientado a flujo bloqueantes

- Es el tipo de socket que utiliza el protocolo TCP y por tanto tiene todas las características relacionadas

Implementación de sockets



Clase Socket

- Implementa un socket de flujo del lado del cliente
- Se le llama simplemente socket
- El trabajo del Socket lo desempeña principalmente una instancia de la clase **SocketImpl**
- Se encuentra en **java.net**

Constructores principales de Socket

- **Socket();** crea un socket de flujo desconectado usando el tipo por defecto de SocketImpl.
- **Socket(InetAddress address, int port);** Crea un socket de flujo y lo conecta a un número de puerto en una IP definida
- **Socket(InetAddress address, int port, InetAddress localAddress, int localPort);** Crea un socket de flujo, ligado a una dirección y puerto local y lo conecta a un número de puerto en una IP definida remota

Métodos principales de Socket

- **bind(SocketAddress bindport);** liga al socket con algún puerto local.
- **close();** Cierra el socket.
- **connect(SocketAddress endpoint);** conecta al socket con el servidor
- **connect(SocketAddress endpoint, int timeout);** conecta al socket con el servidor definiendo un tiempo máximo para la conexión

Clase ServerSocket

- Implementa un socket de servidor de flujo
- Una instancia de esta clase espera por solicitudes de conexión en la red
- El trabajo del **ServerSocket** lo desempeña principalmente una instancia de la clase **SocketImpl**
- Se encuentra en **java.net**

Constructores principales de ServerSocket()

- **ServerSocket();** crea un socket de servidor.
- **ServerSocket(int port);** crea un socket de servidor ligado a un puerto.
- **ServerSocket(int port, int backlog);** crea un socket de servidor ligado a un puerto con una cola de conexiones específica.
- **ServerSocket(int port, int backlog, InetAddress bindAddr);** crea un socket de servidor ligado a un puerto con una cola de conexiones específica y una dirección IP local.

Métodos principales de ServerSocket

- **accept();** escucha esperando una conexión en la red.
- **bind(SocketAddress endpoint);** liga al **ServerSocket** a una dirección IP y número de puerto específico.
- **close();** Cierra el socket.

ServerSocket() y Bind()

```
ServerSocket s = new ServerSocket();

InetSocketAddress dir = new InetSocketAddres(1234);

s.bind(dir);
```

Ó

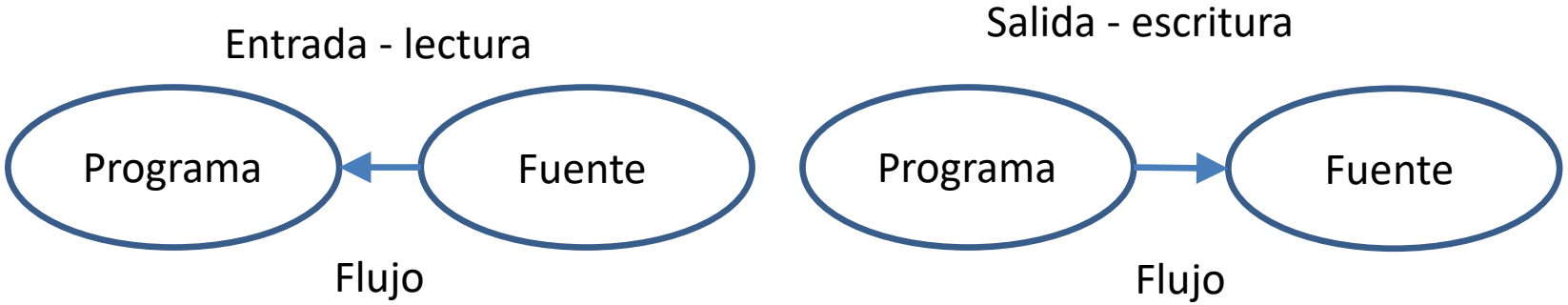
```
ServerSocket s = new ServerSocket(1234);
```

Flujos en java

Volcan Merapi, isla de Java

Flujos en java

- Paquete **java.io**



- Flujos binarios (bytes) y de carácter
- Propósito:
  - Entrada: InputStream y Reader
  - Salida: OutputStream, Writer
  - Lectura/Escritura: RandomAccessFile
  - Transformación de los datos
    - Realizan algún tipo de procesamiento sobre los datos (buffering, conversiones, filtrados: BufferedReader, BufferedWriter

Flujos en java

- Acceso
  - Secuencial
  - Aleatorio (RandomAccessFile)

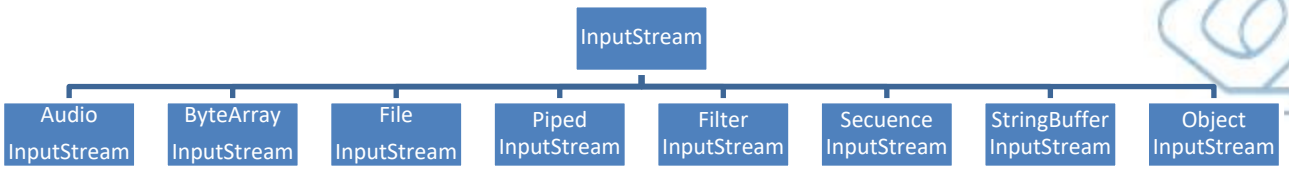
Flujos binarios o de bytes

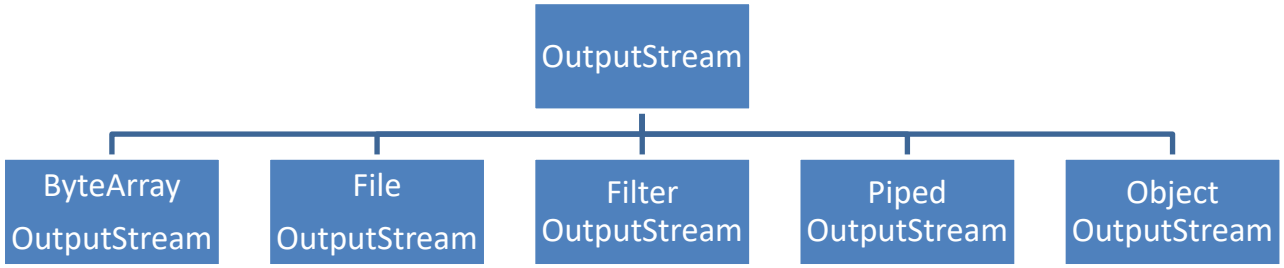
- Byte (8bits)
- Más primitivos y portables
- Los demás flujos lo usan
- Flujo de bajo nivel
- **Clases InputStream y OutputStream**

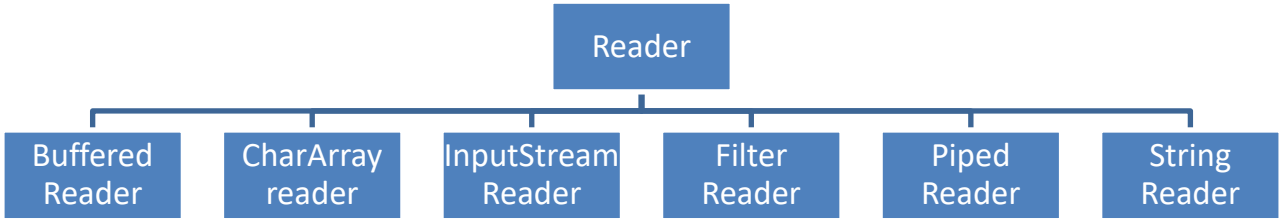
Flujos de caracteres

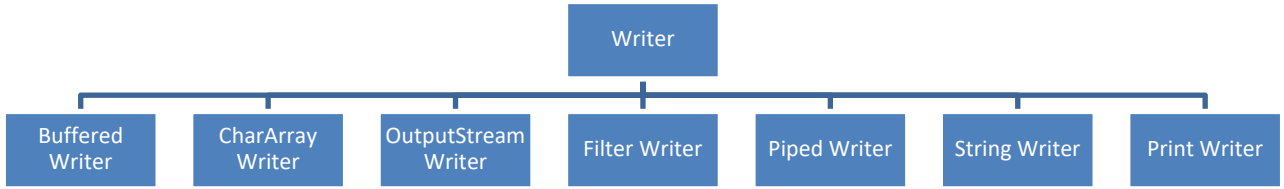
- char (16 bits)
- Codificación unicode
- Ideal para texto plano
- **Clases Reader y Writer**
- Se puede pasar de un flujo de bytes a uno de caracteres con InputStreamReader y OutputStreamWriter

Diagrama de clases principales



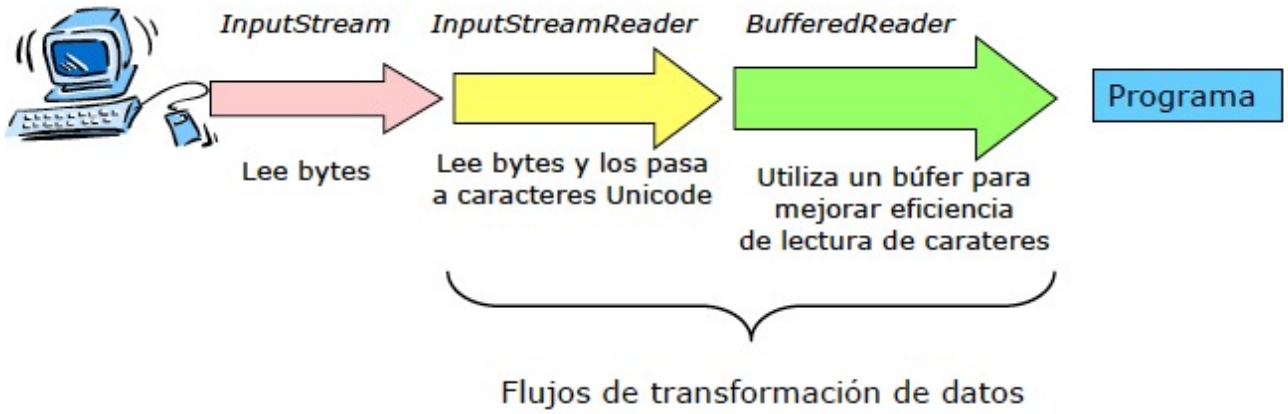






Combinación de flujos

- Los flujos se pueden combinar para obtener la funcionalidad deseada



Lectura y escritura

- Abrir
- Leer o escribir
- Cerrar

Lectura, ImputStream

- `int read();` Lee el próximo byte del flujo representado en un entero. Devuelve -1 si no quedan mas datos que leer.
- `int read(byte[] b);` Lee un arreglo de bytes del flujo.
- `int read(byte[] b, int off, int len);` Lee un arreglo de bytes del flujo, desde y hasta la posición indicada

Lectura, Reader

- int **read() –** Lee el próximo carácter del flujo representado en un entero. Devuelve -1 si no quedan mas datos que leer.
- int **read(***char[] cbuf***) –** Lee un arreglo de caracteres del flujo.
- int **read(***char[] cbuf, int off, int len***) –** Lee un arreglo de caracteres del flujo, desde y hasta la posición indicada.

Escritura, OutputStream

- **void write(int b);** Escribe un solo byte en el flujo.
- **void write(byte[] b);** Escribe un arreglo de bytes en el flujo.
- **void write(byte[] b, int off, int len);** Escribe una porción de un arreglo de bytes en el flujo.

Escritura, Writer

- **void write(int c);** Escribe un solo carácter en el flujo.
- **void write(char[] cbuf);** Escribe un arreglo de caracteres en el flujo.
- **void write(char[] cbuf, int off, int len);** Escribe una porción de un arreglo de caracteres en el flujo

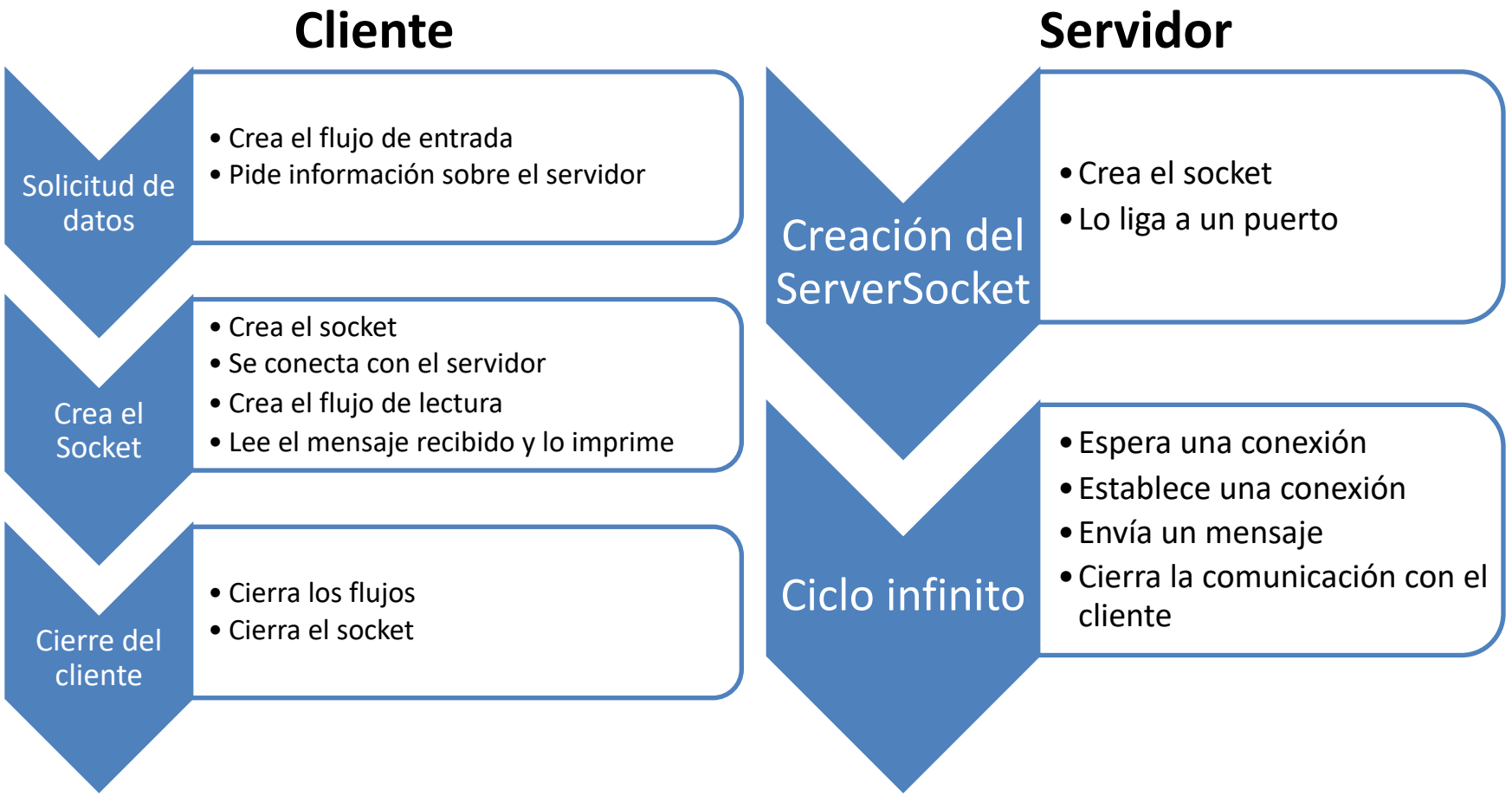
Entrada y salida estándar

- Clase **System** dentro de **java.lang**
- **InputStream in (InputStream);** Flujo de entrada estándar. Típicamente corresponde al teclado.
- **PrintStream out (OutputStream);** Flujo de salida estándar. Típicamente corresponde a la pantalla.
- **PrintStream err (OutputStream);** Flujo de salida estándar de errores. Típicamente corresponde a la pantalla.
- Pueden ser redirigidos

Ejemplo

- Realizar una aplicación con una arquitectura cliente/servidor en java con sockets bloqueantes
- El cliente se conecta con el servidor y recibe un mensaje

Programa de eco



Servidor (1/3)

- Primero, necesitamos las bibliotecas

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

- Ahora, se define la clase y creamos el socket. Lo ligamos al puerto 1234

```
public class SHola {
  public static void main(String args[]){
    try{
      ServerSocket s = new ServerSocket(1234);
      System.out.println("Esperando cliente ...");
```

La creación de sockets siembre debe de hacerse dentro de un bloque try-catch

Servidor (2/3)

- El servidor espera dentro de un ciclo infinito la solicitud de conexión de un cliente

```
for(;;){
        Socket cl = s.accept();
        System.out.println("Conexión establecida desde " +
                                        cl.getInetAddress()+":"+cl.getPort());
```

- Definimos el mensaje a enviar y ligamos un PrintWriter a un flujo de salida de carácter

```
String mensaje ="Hola mundo";
PrintWriter pw = new PrintWriter(new OutputStreamWriter(cl.getOutputStream()));
pw.println(mensaje);
pw.flush();
```

Servidor (3/3)

- Cerramos el flujo, cerramos el socket, termínanos el ciclo for y el bloque try-catch y cerramos la clase

```
            pw.close();
            cl.close();
          }//for
      }catch(Exception e){
          e.printStackTrace();
        }//catch
    }//main
}
```

Cliente (1/3)

- Agregamos las bibliotecas

```
import java.net.*;

import java.io.*;
```

- Definimos la clase y creamos la función main()

```
public class Cliente {

  public static void main(String args[]){
```

- Creamos el bloque try-catch y definimos un flujo de lectura de la entrada estándar

```
try{

    BufferedReader br1 = new BufferedReader(new InputStreamReader(System.in));
```

Cliente (2/3)

- A partir del flujo de lectura obtenemos del usuario la dirección y puerto del servidor

```
System.out.printf("Escriba la dirección del servidor: ");

String host = br1.readLine();

System.out.printf("\n\nEscriba el puerto:");

int pto = Integer.parseInt(br1.readLine());
```

- Creamos el socket

```
Socket cl = new Socket(host,pto);
```

- Creamos un flujo de carácter ligado al socket para recibir el mensaje

```
BufferedReader br2 = new BufferedReader(new InputStreamReader(cl.getInputStream()));
```

Cliente (3/3)

- Leemos el mensaje recibido

```
String mensaje = br2.readLine();

System.out.println("Recibimos un mensaje desde el servidor");

System.out.println("Mensaje:"+mensaje);
```

- Cerramos los flujos, el socket y terminamos el programa

```
            br1.close();

            br2.close();

            cl.close();

          }catch(Exception e){

            e.printStackTrace();

          }

        }

      }
```

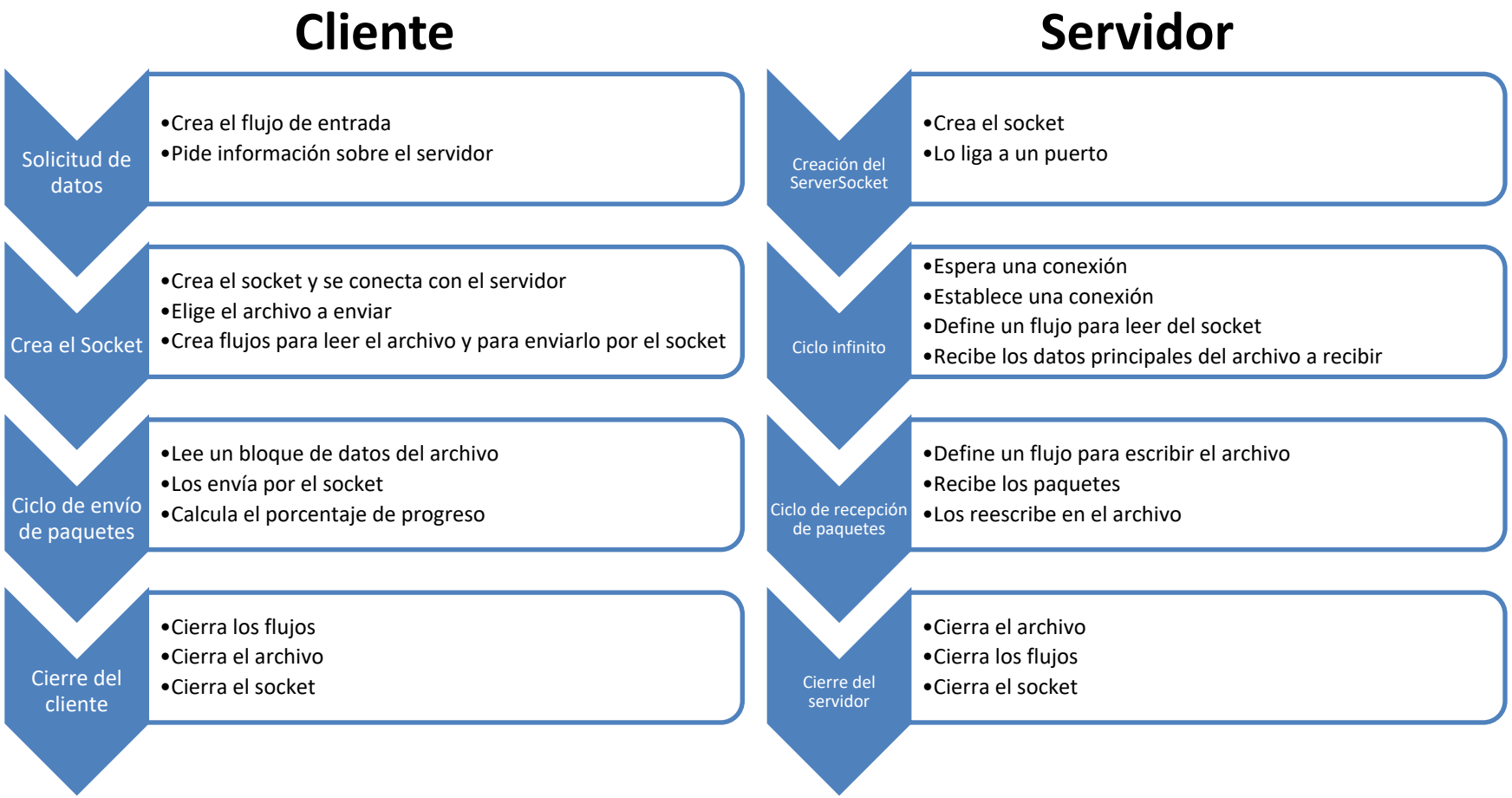
Ejercicio

- Modificar el ejemplo 1 para que el cliente regrese al servidor el mismo mensaje que le fue enviado y lograr la funcionalidad tipo “Eco”

Ejemplo 2: envío de archivos

- Crear una aplicación para el envío de un archivo desde el cliente al servidor
- Se usará un socket orientado a conexión bloqueante
- Los archivos podrán ser de texto o binarios

Envío de archivos



Cliente (1/4)

- Se agregan las bibliotecas pertinentes, se define la clase, la función main() y se inicia el bloque try-catch

```
import javax.swing.JFileChooser;
import java.net.*;
import java.io.*;
public class ClienteArchivo {
  public static void main(String[] args){
    try{
```

- Se define un flujo de entrada para obtener los datos del servidor

```
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
System.out.printf("Escriba la dirección del servidor:");
String host = br.readLine();
System.out.printf("\n\nEscriba el puerto:");
int pto = Integer.parseInt(br.readLine());
```

- Se define el socket

```
Socket cl = new Socket(host, pto);
```

Cliente (2/4)

- Se usa un **JFileChooser()** para elegir el archivo a enviar

```
JFileChooser jf = new JFileChooser();
int r = jf.showOpenDialog(null);
```

- Una vez seleccionado, se obtienen sus datos principales

```
if (r==JFileChooser.APPROVE_OPTION){
  File f = jf.getSelectedFile();  //Manejador
  String archivo = f.getAbsolutePath(); //Dirección
  String nombre = f.getName(); //Nombre
  long tam = f.length();  //Tamaño
```

- Se definen dos flujos orientados a bytes, uno se usa para leer el archivo y el otro para enviar los datos por el socket

```
DataOutputStream dos = new DataOutputStream(cl.getOutputStream());
DataInputStream dis = new DataInputStream(new FileInputStream(archivo));
```

Cliente (3/4)

- Enviamos los datos generales del archivo por el socket

```
dos.writeUTF(nombre);
dos.flush();
dos.writeLong(tam);
dos.flush();
```

- Leemos los datos contenidos en el archivo en paquetes de 1024 y los enviamos por el socket

```
byte[] b = new byte[1024];
long enviados = 0;
int porcentaje, n;
while (enviados < tam){
   n = dis.read(b);
   dos.write(b,0,n);
   dos.flush();
   enviados = enviados+n;
   porcentaje = (int)(enviados*100/tam);
   System.out.print("Enviado: "+porcentaje+"%\r");
 }//While
```

Cliente (4/4)

- Cerramos los flujos, el socket, terminamos bloques y cerramos la clase

```
            System.out.print("\n\nArchivo enviado");
            dos.close();
            dis.close();
            cl.close();
          }//if
        }catch(Exception e){
          e.printStackTrace();
        }
      }
    }
```

Servidor (1/3)

- Agregamos bibliotecas, definimos la clase , el main(), iniciamos el bloque try y definimos el socket

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

public class ServidorArchivo {
  public static void main(String[] args){
    try{
     ServerSocket s = new ServerSocket(7000);
```

- Iniciamos el ciclo infinito y esperamos una conexión

```
for(;;){
  Socket cl = s.accept();
  System.out.println("Conexión establecida desde"+cl.getInetAddress()+":"+cl.getPort());
```

- Definimos un flujo de nivel de bits de entrada ligado al socket

```
DataInputStream dis = new DataInputStream(cl.getInputStream());
```

Servidor (2/3)

- Leemos los datos principales del archivo y creamos un flujo para escribir el archivo de salida

```
byte[] b = new byte[1024];
String nombre = dis.readUTF();
System.out.println("Recibimos el archivo:"+nombre);
long tam = dis.readLong();
DataOutputStream dos = new DataOutputStream(new FileOutputStream(nombre));
```

- Preparamos los datos para recibir los paquetes de datos del archivo

```
long recibidos=0;
int n, porcentaje;
```

Servidor (3/3)

- Definimos el ciclo donde estaremos recibiendo los datos enviados por el cliente

```
while(recibidos < tam){
  n = dis.read(b);
  dos.write(b,0,n);
  dos.flush();
  recibidos = recibidos + n;
  porcentaje = (int)(recibidos*100/tam);
  System.out.print("\n\nArchivo recibido.");
}//While
```

- Cerramos los flujos, el socket y el resto del programa

```
            dos.close();
            dis.close();
            cl.close();
          }
        }catch(Exception e){
              e.printStackTrace();
        }//catch
      }
    }
```

Ejercicio

- Modificar el archivo anterior para que permita el envío de múltiples archivos.

Sockets en C

Sockets bloqueantes

Cliente de eco

- Crea un socket de flujo usando **socket()**
- Establece la conexión usando **connect()**
- Envía un mensaje empleado **send()**
- Recibe una respuesta con **recv()**
- Cierra la conexión usando **close()**

Cliente de eco
Biblioteca MensajeError.h

```
#include <stdio.h>

#include <stdlib.h>


void mensajeFinalError(const char *mensaje){

    fputs(mensaje,stderr);

    fputc('\n',stderr);

    exit(1);

}

void mensajeFinalSistema(const char *mensaje){

    perror(mensaje);

    exit(1);

}
```

Cliente de eco (1/5)
Segmento #include y análisis inicial

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include "MensajeError.h"
#define TAMBUFER 2000
```

```
int main(int argc, char** argv) {
    if (argc < 3|| argc >4)
        mensajeFinalError("Uso: EcoTCPCliente <Dirección del
servidor> <Palabra de eco> [<Puerto>]");
    char *servIP = argv[1];
    char *cadenaEco = argv[2];
    //Argumento opcional, se agrega por defecto
    in_port_t puerto = (argc == 4) ? atoi(argv[3]) : 7;
```

Cliente de eco (2/5)
Creación del socket de flujo

```
    //Crea el socket del cliente TCP
int s = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
if(s < 0){
      mensajeFinalError("Error de apertura del conector");
}
```

Prototipo socket()

```
#include <sys/types.h>

#include <sys/socket.h>


Int socket(int af, int type, int protocol)
```

Familia de direcciones (1/2)

Familia	Descripción
AF_LOCAL	Es otro nombre para AF_UNIX
AF_INET	Protocolo internet DARPA (TCP/IP)
AF_INET6	Protocolo internet versión 6
AF_PUP	Antigua red Xerox
AF_CHAOS	Red Chaos del MIT
AF_NS	Arquitectura Xerox Network System
AF_ISO	Protocolos OSI
AF_ECMA	Red European Computer Manufactures
AF_DATAKIT	Red Datakit de AT&T
AF_CCITT	Protocolos del CCITT, por ejemplo X.25
AF_SNA	System Network Architecture (SNA) de IBM
AF_DECnet	Red DEC

Familia de direcciones (2/2)

Familia	Descripción
AF_IMPLINK	Antigua interfaz de enlace 1822 Interface Message Processor
AF_DLI	Interfaz directa de enlace
AF_LAT	Interfaz de teminales de red de área local
AF_HYLINK	Network Syste, Córporation Hyperchannel
AF_APPLETALK	Red AppleTalk
AF_ROUTE	Comunicación con la capa de encaminamiento del núcleo
AF_LINK	Acceso a la capa de enlace
AF_XTP	eXpress Transfer Protocol
AF_COIP	Connection-oriented IP (ST II)
AF_CNT	Computer Network Tecnology
AF_IPX	Protocolo Internet de Novell

Tipos de semántica de la comunicación

- SOCK_STREAM, sockets de flujo
- SOCK_DGRAM, sockets de datagrama
- SOCK_RAW, sockets crudos
- SOCK_SEQPACKET, conector no orientado a conexión pero fiable de longitud fija (solo en AF_NS)
- SOCK_RDM, conector no orientado a conexión pero fiable y secuencial (no implementado pero se puede simular a nivel de capa de usuario)

Parámetro protocolo

- Especifica el protocolo particular
- Normalmente cada tipo de protocolo tiene relacionado un solo protocolo, en caso contrario, aquí se especificaría.
- Un valor de 0 deja que el sistema decida el protocolo.

Cliente de eco (3/5)
Preparando la dirección y conectando

```
//Creamos la dirección del servidor de entrada
   struct sockaddr_in dirServ;
   memset(&dirServ,0,sizeof(dirServ));
   dirServ.sin_family = AF_INET;
   int valRet = inet_pton(AF_INET, servIP,&dirServ.sin_addr.s_addr);
   if(valRet == 0)
        mensajeFinalError("Dirección del servidor erronea");
   else if(valRet < 0)
        mensajeFinalError("Error en el inet_pton()");
   dirServ.sin_port = htons(puerto);
   //Establecemos la comunicación con el servidor de eco
   if(connect(s, (struct sockaddr*) &dirServ,sizeof(dirServ))<0)
        mensajeFinalError("Error en la conexión");
   size_t longCadenaEco = strlen(cadenaEco);
```

Formas de direcciones de la familia AF_INET (direcciones de Internet)

```
struct in_addr{
        u_long s_addr // 32 bits con la dirección IP
};
Struct sockaddr_in{
        short sin_family; //AF_INET
        u_short sin_port; //16 bits para el puerto
        struct in_addr sin_addr;
        char sin_zero [8]; //8 bytes no usados
};
```

Prototipo connect()

```
#include <sys/socket.h>

#include <sys/types.h>

#include <netinet/in.h> //Familia AF_INET


int connect(int sfd, const void *addr, int addrlen);
```

Parámetros de connect()

- *sdf* es el descriptor del socket que da acceso al canal.
- *addr* puntero a la estructura de la dirección del conector.
- *addrlen* es el tamaño de la dirección en bytes.

connect()

- La estructura de la dirección depende de la familia de conectores
- Si es SOCK_DGRAM, especifica la dirección del conector pero no se conecta.
- Si es SOCK_STREAM, intenta conectarse con el ordenador remoto.

Cliente de eco (4/5)
Enviando cadena al servidor

```
//Envia el mensaje al servidor
    ssize_t numBytes = send(s,cadenaEco,longCadenaEco, 0);
    if(numBytes< 0)
        mensajeFinalError("Fallo el envio");
    else if(numBytes != longCadenaEco)
        mensajeFinalError("Número de bytes enviados erroneo");
```

Envío de un mensaje a un conector

- **writev(), send(), sendto() y sendmsg()**
- **writev()** es una generalidad de **write()** y se puede usar para escribir en un archivo o en un socket
- El resto solo se usan para sockets

writev()

#include <sys/uio.h>

ssize_t writev(int fildes, cons struct iovec *iov, size_t iovcnt)

- *fildes* es el descriptor del fichero o socket
- *iov* es el vector donde están los datos
- *iovcnt* nos dice cuantos datos se van a escribir.

send(), sendto() y sendmsg()

```
#include <sys/socket.h>

int send(int sfd, void buf, int len, int flags)

int sendto(int sfd, void buf, int len, int flags, void *to, int tolen);

Int sendmsg (int sfd, struct msghdr msg[], int flag);
```

- Los posibles valores de *flag* son 0 ó MSG_OOB, mensaje urgente.
- *msg[],* son un arreglo de datos struc msghdr y permite enviar un mensaje que está en varias secciones de memoria.

Cliente de eco (5/5)
Recibiendo respuesta del servidor

```
//Recibimos de vuelta la cadena desde el servidor
    unsigned int totalBytesRec = 0;
    while(totalBytesRec < longCadenaEco){
        char bufer[TAMBUFER];
        memset(bufer,0,TAMBUFER);
        numBytes = recv(s,bufer,TAMBUFER, 0);
        if(numBytes<0)
            mensajeFinalError("Recepción fallida");
        else if(numBytes==0)
            mensajeFinalError("Conexión cerrada prematuramente");
        totalBytesRec += numBytes;
        printf("Recibido: %s\n",bufer);
    }
    close(s);
    return 0;
}
```

Lectura de un mensaje a un socket

- **readv(), recv(), recvto() y recvmsg()**
- **readv()** es una generalidad de **read()** y se puede usar para escribir en un archivo o en un socket
- El resto solo se usan para sockets y son un espejo de las funciones anteriores.

Servidor de eco en C (1/6)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include "./MensajeError.h"

#define MAXLISTA 5
#define TAMBUFER 1024
```

```
void manejadorTCPCliente(int);

int main(int argc, char **argv){
    if(argc != 2) //Revisamos el número de argumentos
        mensajeFinalError("Uso: EcoTCPServidor
[<puerto>]");
    in_port_t prtoServ = atoi(argv[1]);
```

Servidor de eco en C (2/6)

```
//Creamos el socket de entrada
    int sockServ;
    if((sockServ = socket(AF_INET,SOCK_STREAM,IPPROTO_TCP)) < 0)
        mensajeFinalError("Fallo la apertura del socket");
    //Se construye la estructura de la dirección
    struct sockaddr_in dirServ;              //Estructura para la dirección local
    memset(&dirServ, 0 , sizeof(dirServ));       //Limpiamos la estructura
    dirServ.sin_family = AF_INET;             //Familia de direcciones IPv4
    dirServ.sin_addr.s_addr = htons(INADDR_ANY);   //Cualquier interfaz de entrada
    dirServ.sin_port = htons(prtoServ);         //Número de puerto

    //Se enlaza a la dirección local
    if(bind(sockServ, (struct sockaddr*)&dirServ, sizeof(dirServ))< 0)
        mensajeFinalError("Error al enlazar");
```

Función bind()

```
#include <sys/socket.h>

#include <netinet/in.h> //Solo AF_INET


Int bind(int sfd, const void *addr, int addrlen);
```

Servidor de eco en C (3/6)

```
//Marcamos el socket para que pueda escuchar conexiones
    if(listen(sockServ, MAXLISTA) < 0)


    for(;;){//Lazo infinito
        struct sockaddr_in dirCliente;      //Dirección del cliente
        //Obtenemos el tamaño de la estructura
        socklen_t dirClienteTam = sizeof(dirCliente);
```

Servidor de eco en C (4/6)

```
        //Esperamos que se conecte un cliente
        int sockCliente = accept(sockServ,(struct sockaddr *)&dirCliente, &dirClienteTam);
        if (sockCliente < 0)
            mensajeFinalError("Fallo la conexión ");
        //Se conecto un cliente
        char nombreCliente[INET_ADDRSTRLEN];
        if(inet_ntop(AF_INET, &dirCliente.sin_addr.s_addr, nombreCliente, sizeof(nombreCliente)) != NULL)
            printf("Cliente conectado: %s/%d\n",nombreCliente,ntohs(dirCliente.sin_port));
        else
            puts("Inposible conectar el cliente");
        manejadorTCPCliente(sockCliente);
    }
}
```

Función listen()

```
#include <sys/socket.h>

#include <netinet/in.h> //Solo AF_INET


int listen(int sfd, int backlog);
```

- *sfd* es el descriptor de socket
- *backlog* el número de conexiones permitidas

Función accept()

#include <sys/socket.h>

int accept (int sfd, void *addr, int *addrlen)

- *sfd* es el descriptor de socket
- *addr* apuntador a la estructura local del socket remoto
- *addrlen* es el tamaño de dicha estructura

Servidor de eco en C (5/6)

```
void manejadorTCPCliente(int sockCliente){
    char bufer[TAMBUFER];

    //Recibe mensaje del cliente
    ssize_t numBytesRecibidos = recv(sockCliente, bufer, TAMBUFER, 0);
    if(numBytesRecibidos < 0)
        mensajeFinalError("Error en la lectura de datos recibidos");
    //Envia los datos recibidos
    while(numBytesRecibidos > 0){
```

Servidor de eco en C (6/6)

```
//Eco del mensaje
        ssize_t numBytesEnviados = send(sockCliente, bufer, numBytesRecibidos, 0);
        if(numBytesEnviados < 0)
            mensajeFinalError("Error en el envio");
        else if(numBytesEnviados == 0)
            mensajeFinalError("Número de bytes enviado erroneo");
        //Revisamos si hay mas datos a recibir
        numBytesRecibidos = recv(sockCliente,bufer, TAMBUFER,0);
        if(numBytesRecibidos < 0)
            mensajeFinalError("Error en la lectura de datos recibidos");
    }
    close(sockCliente);
}
```