SOCKETS

Introducción

- Permiten la comunicación entre dos o más procesos ejecutando en un mismo equipo o equipos independientes

- La comunicación que posibilitan es full-duplex (bi-direccional)

- Sigue el modelo Cliente – Servidor

- El Cliente debe conocer la existencia y dirección del Servidor
  El Servidor no precisa conocer la dirección del cliente

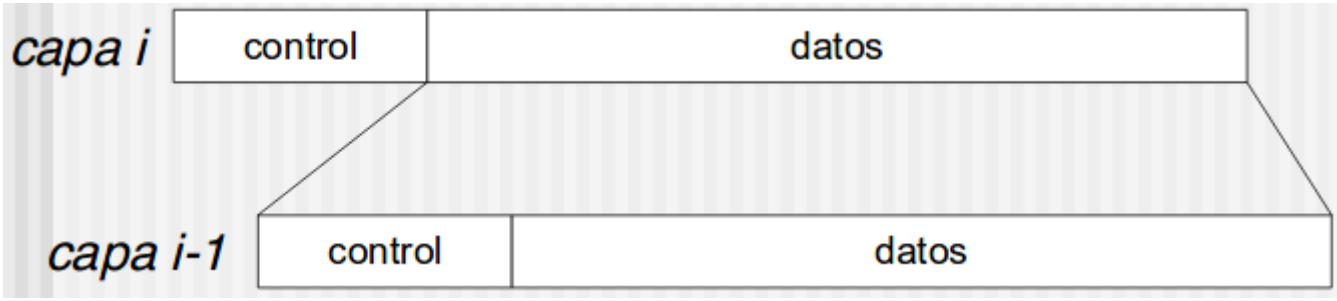
Instituto Politécnico Superior

Introducción

- En 1977 la Internacional Standards Organization (ISO) diseñó una arquitectura de comunicación.

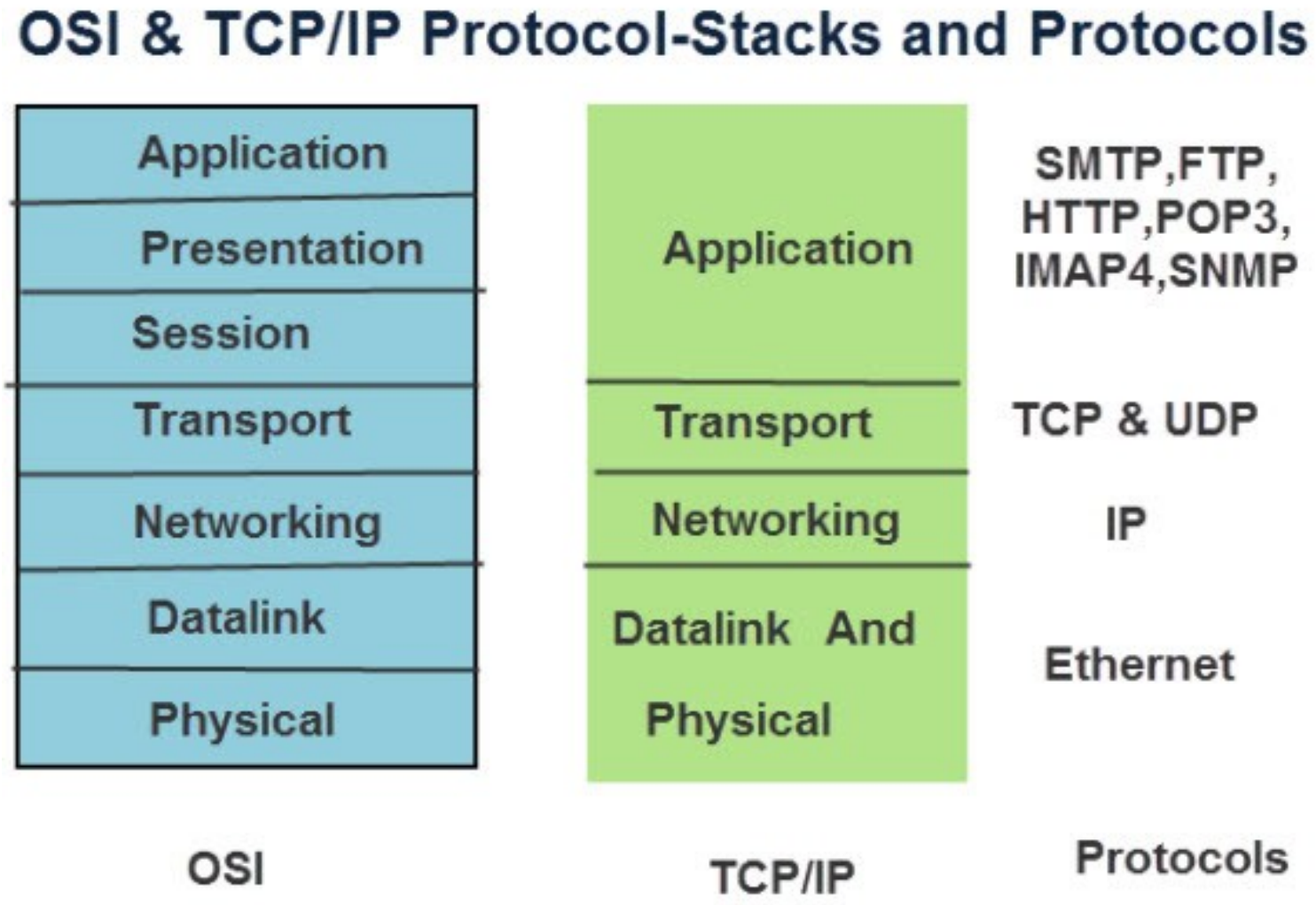
- Se logró el modelo de referencia para la Interconexión de Sistemas Abiertos (OSI) que define una arquitectura de siete niveles o capas

- Cada paquete enviado por una capa se compone de control + datos
  El conjunto control+datos de una capa viaja en los datos de la capa superior



Instituto Politécnico Superior

- Stacks de protocolos



Instituto Politécnico Superior

5 Application Layer

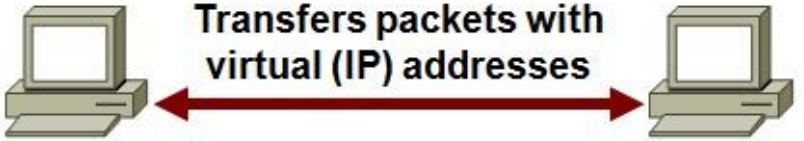
The Application layer is the group of applications requiring network communications.

4 Transport Layer (TCP/UDP)

The Transport layer establishes the connection between applications on different hosts.

3 Network Layer (IP)

The Network layer is responsible for creating the packets that move across the network.



2 Data Link Layer (MAC)

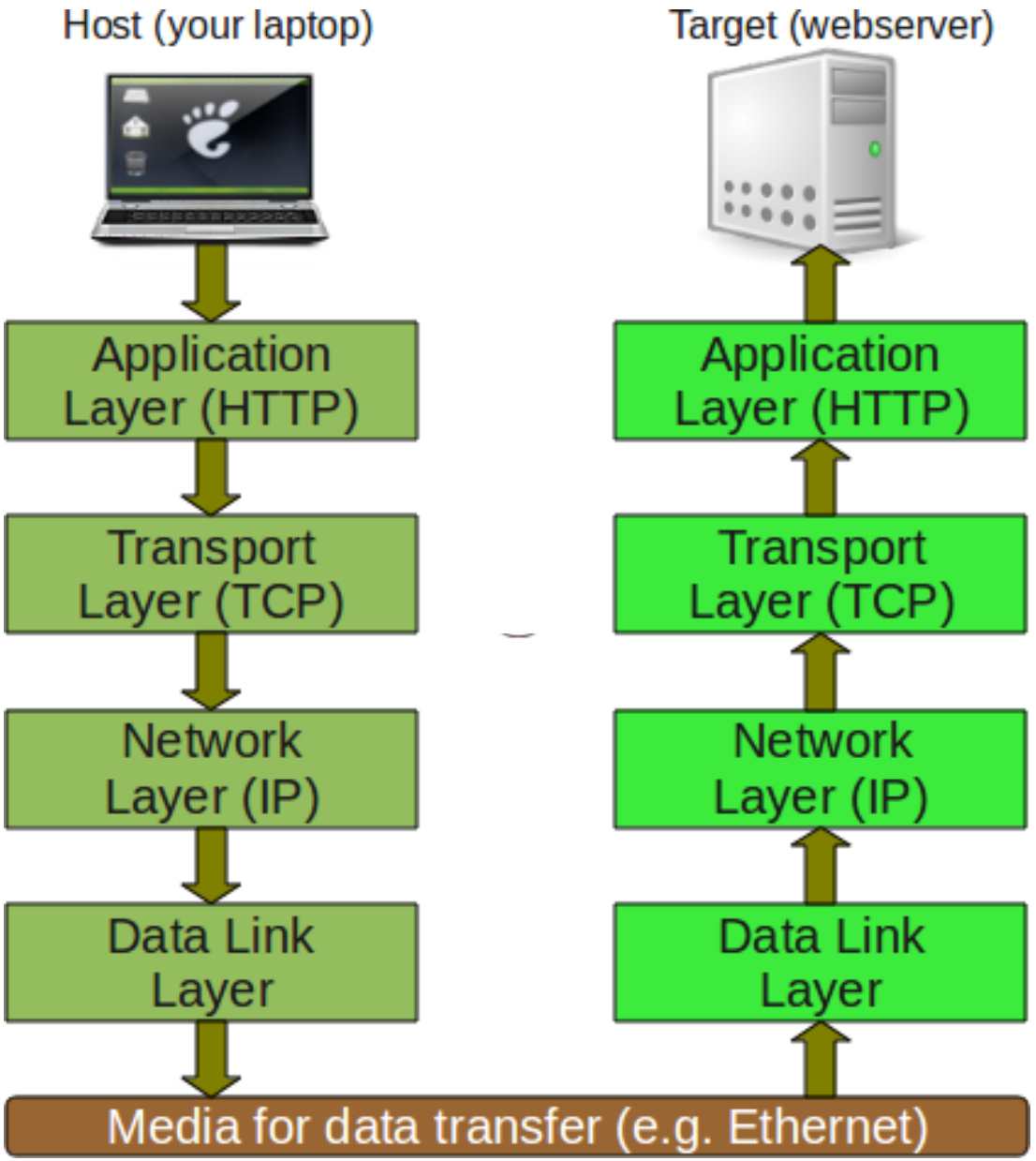
The Data Link layer is responsible for creating the frames that move across the network.

1 Physical Layer

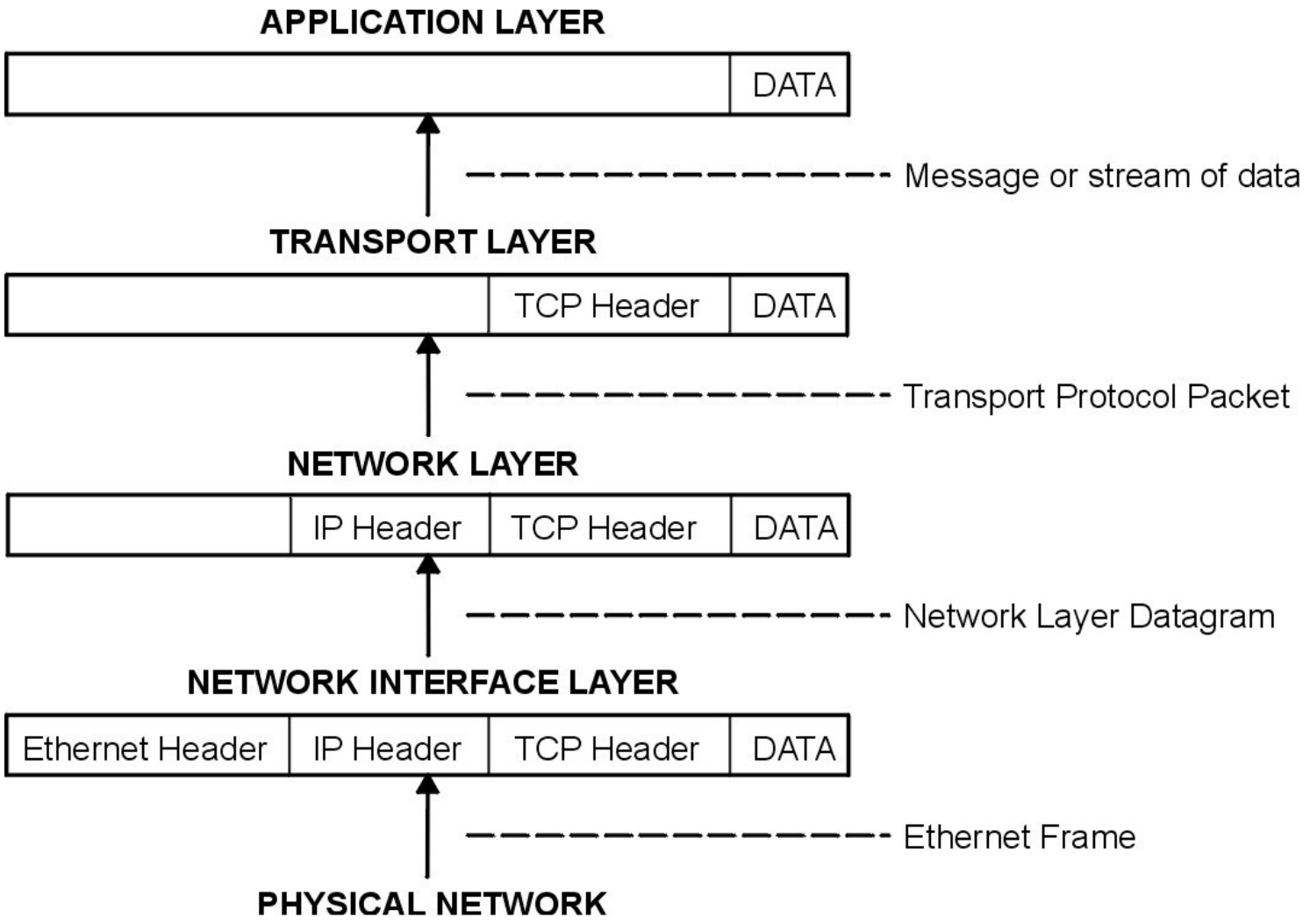
The Physical layer is the transceiver that drives the signals on the network.

Instituto Politécnico Superior

- Empaquetado



- Empaquetado



Instituto Politécnico Superior

Tipos de sockets

- Al crear un socket se debe especificar;
  . address domain (**dominio**)
  . socket type (**tipo**)

- Dos procesos pueden comunicase si sus sockets son del mismo **dominio** y **tipo**.

- **Dominios** (también llamados Address Family)
  - El ***unix domain***: dos procesos en el mismo equipo se comunican compartiendo un file system común.
    Formato de direcciones: string indicando un path en el file system
    Address Family UNIX (AF_UNIX o AF_LOCAL)

  - El ***Internet domain***, dos procesos se ejecutan en diferentes equipos y se comunican a través de Internet.
    Formato de direcciones: dirección IP y puerto
    Direccion IP(v4): 32 bits. Puerto (entero positivo, >2000)
    Address Family INET (AF_INET para IPv4)

- Tipos de Sockets

Stream sockets (SOCK_STREAM)

- Tratan la comunicación como un flujo (stream) continuo de caracteres
- Usan **TCP** (Transmission Control Protocol), protocolo confiable (restransmisiones en error y orden) y orientado a la conexión.

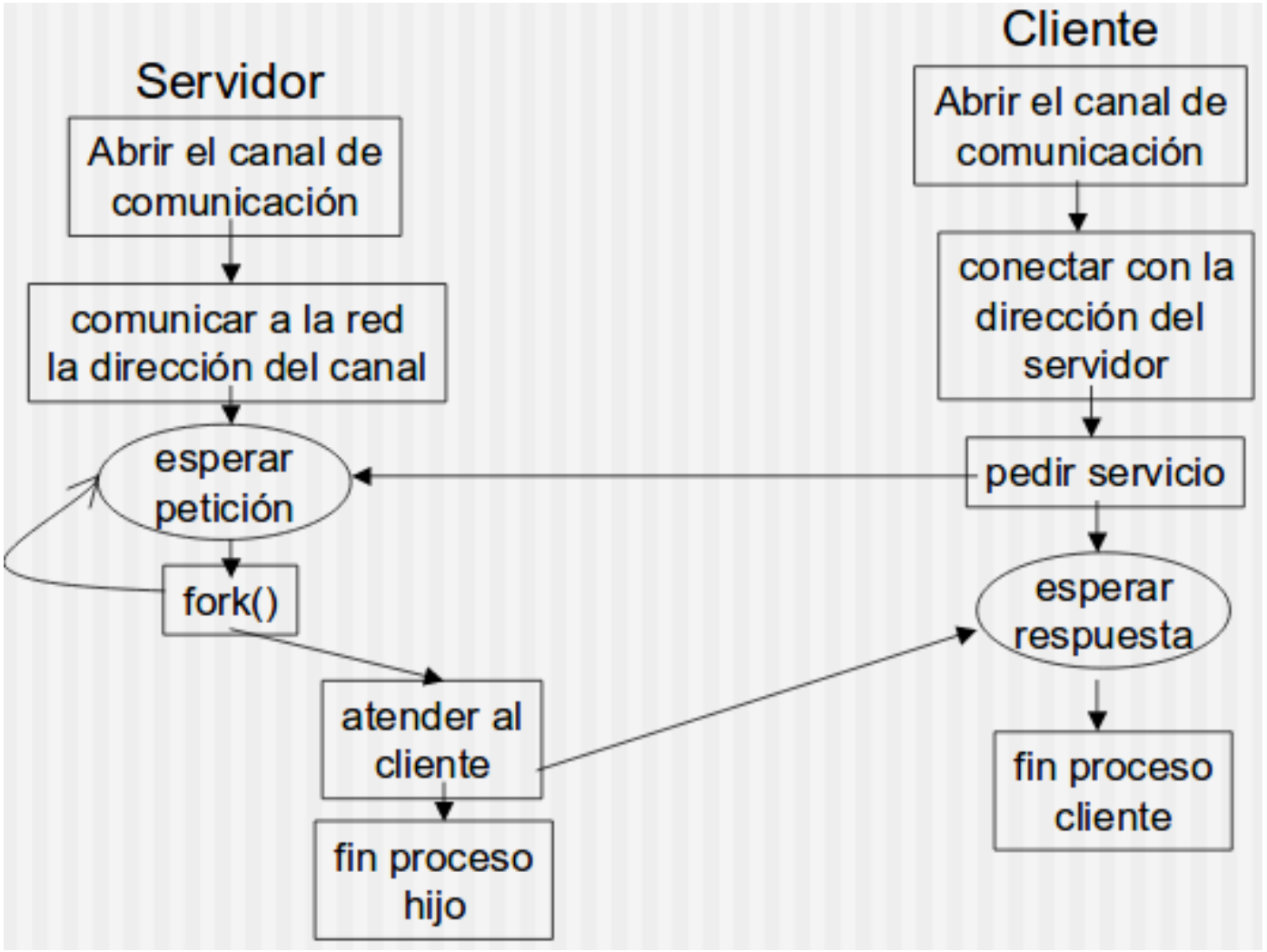
Datagram sockets (SOCK_DGRAM)

- Deben leer mensajes completos.
- Usan **UDP** (User Datagram Protocol), protocolo no confiable y no orientado a la conexión.

Que protocolos emplean cada TCP y UDP?
TCP: HTTP, HTTPS, FTP, SMTP
UDP: DNS, DHCP, VoIP, NTP, SNMP

Diferencias importantes
http://www.diffen.com/difference/TCP_vs_UDP

Instituto Politécnico Superior



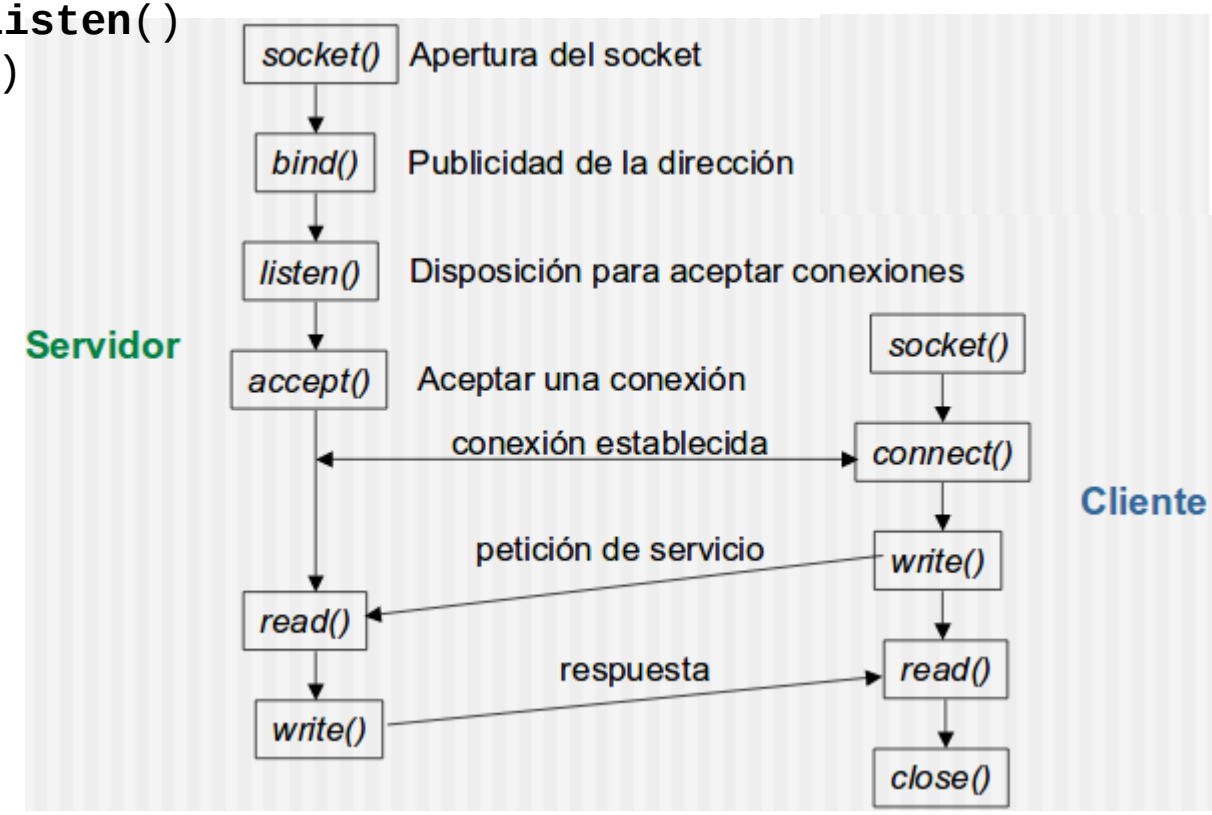
Instituto Politécnico Superior

- Pasos que realiza el **Cliente**

- Crea un socket con **`socket()`**
- Conecta el socket con la dirección del servidor usando **`connect()`**
- Envía y recibe data, como siempre lo trata como un archivo, y se puede emplear por ejemplo `read()` y `write()`

- Pasos que realiza el **Servidor**

- Crea un socket con **`socket()`**
- Vincula el socket a una dirección using **`bind()`**
- Escucha por conexiones entrantes con **`listen()`**
- Acepta dichas conexiones con `accept()`
- Envía y recibe data



Función socket

Crea un socket

```
# include <sys/types.h>
# include <sys/socket.h>
int socket (int addr_family, int type, int protocol)
```

addr_family: AF_INET o AF_UNIX
type: SOCK_STREAM o SOCK_DGRAM
protocol: (sub)protocolo a utilizar, 0(cero) escoge el sistema

Retorno
En éxito, retorna un file descriptor al nuevo socket.
Error, retorna -1 y se setea errno.

Función bind

Asocia un socket a una dirección determinada - “assigning a name to a socket”

```
# include <sys/types.h>
# include <sys/socket.h>
# include <sys/netinet.h>  //Solo para addr_family AF_INET
```

int bind (int sockfd, struct sockaddr *addr, int addrlen)

sockfd: Socket file descriptor (referencia al socket, retornado por socket(...))
addr: Dirección a donde hacer el *bind*
addrlen: se caclula con sizeof

```
struct sockaddr {
    sa_family_t sa_family;
    char        sa_data[14];
}
```

El propósito de esta esctructura es castear el puntero para evitar warnings en complicación.

Retorno
En éxito, retorna 0 (cero).
Error, retorna -1 y se setea errno.

Structs

`struct sockaddr` es una definición genérica.
Empleada por cualquier socket que requiere una dirección.

```
struct sockaddr {
   sa_family_t sa_family;
   char sa_data[14];
};
```

`struct sockaddr_un ("Unix sockets" address)`

```
struct sockaddr_un {
   short sun_family;      /*AF_UNIX*/
   char sun_PATH[108];    /*path name */
};
```

`struct sockaddr_in ("Internet socket" address)`

```
struct sockaddr_in {
   short sin_family;       /* AF_INET */
   u_short sin_port;       /* 16-bit port number */
   struct in_addr sin_addr;
   char sin_zero[8];       /* unused */
};
```

Función listen

Indica que el servidor está listo para reicibir peticiones marcando el socket como “pasivo”.
El socket debe de ser de tipo SOCK_STREAM
Habilita una cola asociada al socket donde alojar peticiones de conexión de los clientes, en caso de cola completa, el cliente recibe un error de conexión.

```
# include <sys/types.h>
# include <sys/socket.h>
int listen(int sockfd, int backlog)
```

sockfd: Socket file descriptor
backlog: tamaño de la cola (máxima longitud)
Debe ser mayor para servidores interactivos – muchos clientes

Retorno
En éxito, retorna 0 (cero).
Error, retorna -1 y se setea errno.

Función accept

Se usa solo en el servidor.
Extrae la primer conexión de la cola de conexiones pendientes en el socket **sockfd**.
El socket debe de ser de tipo SOCK_STREAM
Crea un nuevo socket (conectado) y retorna un nuevo file descriptor refiriendo a ese socket, el cual no está en estado “listen”. El socket original no es afectado por esta llamada.

```
#include <sys/types.h>
#include <sys/socket.h>
int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);
```

sockfd: Socket file descriptor (el socket original)
addr: puntero para completar la estructura con la dirección del “peer socket” (el socket remoto que pide la conexión)
addrlen: especifica en bytes el tamaño de la estructura apuntada por **addr**

Retorno
En éxito, retorna un file descriptor para que se usa para la comunicación con el cliente.
Error, retorna -1 y se setea errno.

Instituto Politécnico Superior

Función connect

Se usa solo en el cliente para conectar con el servidor.
- socket SOCK_DGRAM no se conecta en la llamada, se retorna inmediatamente (UDP)
- socket SOCK_STREAM la llamada se bloquea intentando conectar (TCP)

```
#include <sys/types.h>
#include <sys/socket.h>
int connect(int sockfd, const struct sockaddr *addr, socklen_t addrlen);
```

sockfd: Socket file descriptor
addr: puntero constante a estructura sockaddr
addrlen: especifica en bytes el tamaño de la estructura apuntada por **addr**

Retorno
En éxito, retorna 0 (cero).
Error, retorna -1 y se setea errno.

```c
#include <sys/types.h>
#include <sys/socket.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#define MAX_CLIENTS 5

void error(char *msg) {
    perror(msg);
    exit(1);
}

int main(int argc, char *argv[]) {
  int sockfd;
  int msgsock;
  int rval;
  struct sockaddr server;
  char buf[1024];

  sockfd = socket(AF_UNIX, SOCK_STREAM, 0);
  if (sockfd == -1) {
    error("Falla socket");
  }

  server.sa_family = AF_UNIX;
  strcpy(server.sa_data, argv[1]);
  unlink(server.sa_data);       //Para desvincular cualquier socket previo...

  if(bind(sockfd, &server, sizeof(server))) {
    error("Falla bind");
  }

  listen(sockfd, MAX_CLIENTS);

  for (;;) {
    msgsock = accept(sockfd, 0, 0);
    if (msgsock == -1) {
          error("Falla accept");
    }
    do {
        bzero(buf, sizeof(buf));  // Inicializa el buffer
        if ((rval = read(msgsock, buf, 1024)) < 0) // lee del socket
            error("Falla lectura");
        else if (rval == 0) {
              printf("Conexion finalizada.\n");
              exit(0);
            } else {  //Lee datos ok
                printf("-->%s\n", buf);
            }
    } while (rval > 0);
    close(msgsock);
  } // for
  close(sockfd);

  return 0;
}
```

SERVER
AF_UNIX
$ ./server my-socket

```c
#include <sys/types.h>
#include <sys/socket.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>

void error(char *msg) {
    perror(msg);
    exit(1);
}

int  main(int argc, char *argv[]) {
     int sockfd;
     struct sockaddr client;
     char data[1024];

     sockfd = socket(AF_UNIX, SOCK_STREAM, 0);
     if (sockfd < 0) {
        error("Falla socket);
     }

     client.sa_family = AF_UNIX;
     strcpy(client.sa_data, argv[1]);

     if (connect(sockfd, &client, sizeof(client)) < 0) {
         close(sockfd);
         error("Falla connect");
     }

     while (fgets(data, 1024, stdin)) {
        if (send(sockfd, data, strlen(data), 0) == -1) {
          error("Falla send");
        }
      }

     close(sockfd);

     return 0;
}
```

CLIENT
AF_UNIX
$ ./client my-socket
data…. Ctrl+D

Funciones para leer desde un socket

Son las siguientes:

- **read**, **readv**: pueden leer desde un file descriptor, para files o sockets.

- **recv**, **recvfrom**

```
#include <sys/socket.h>
int recv (int sockfd, void *buff, int len, int flags);
int recvfrom (int sockfd, void *buff, int len, int flags, struct sockaddr *from, int* fromlen);
```

buff: puntero al espacio de memoria donde de almacenan los datos leidos
len: número máximo de bytes que se pueden escribir en **buff**
flags: se forma mediante Ors (algunas son MSG_PEEK y MSG_OOB)

Ambas funciones retornan el total de bytes leídos.
Si el mensaje supera el tamaño del buffer los bytes excesivos de descartan (*datagram*).
Si no se reciben mensajes, estas funciones quedan a la espera. (excepción sockets nobloqueantes, fnctl). Hay funciones para conocer si llegan más datos a un socket (**select**, **poll**, **epoll**).

recvfrom es como recv pero retorna la dirección del remitente