Redes de Computadores

Nivel de Aplicación: Programación con sockets I

Área de Ingeniería Telemática
Dpto. Automática y Computación
http://www.tlm.unavarra.es/

En clases anteriores...

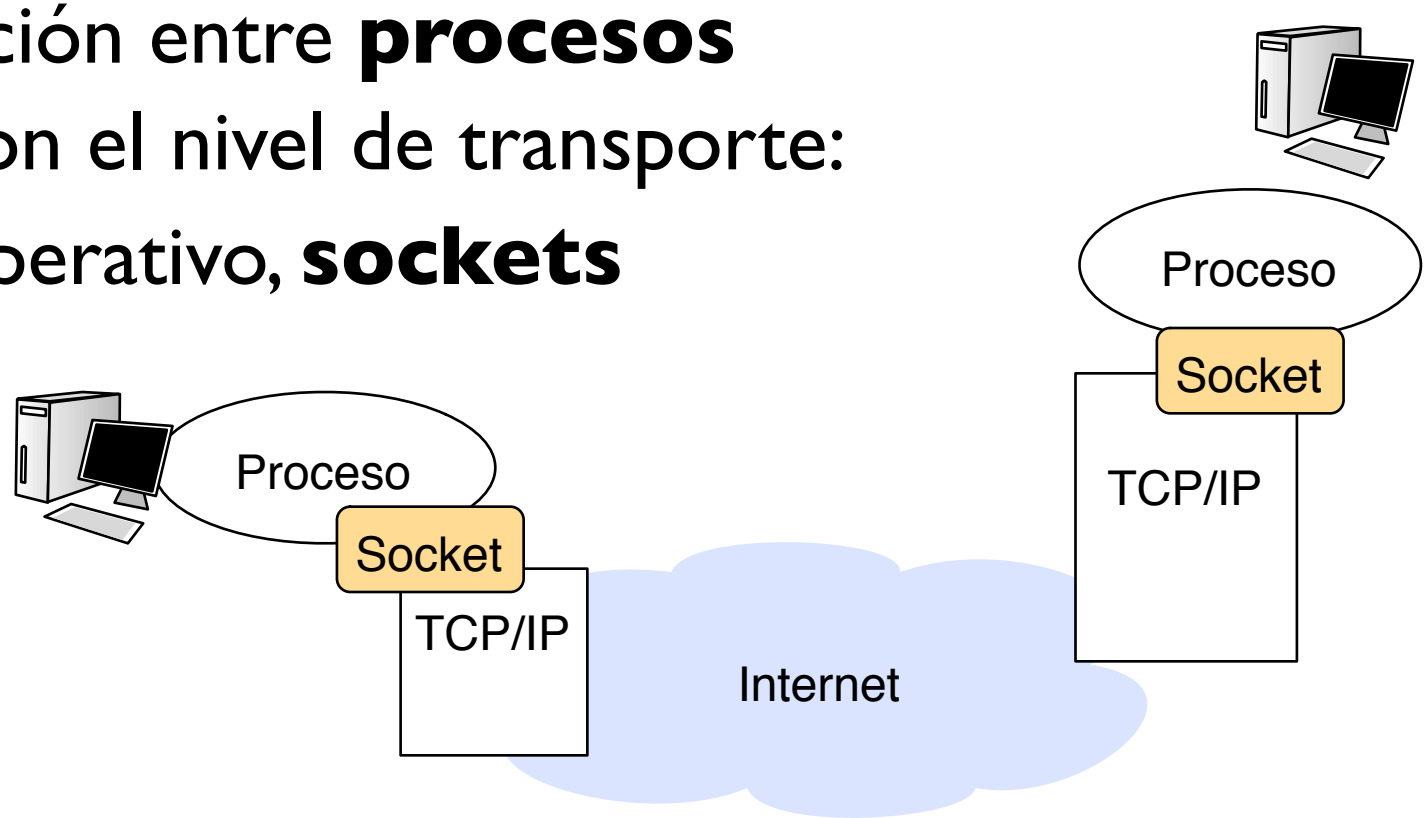
- El nivel de aplicación en Internet. Las aplicaciones usan sus protocolos de nivel de aplicación que usan los servicios de TCP/UDP proporcionados por el sistema operativo
- A grandes rasgos hemos visto como funcionan las aplicaciones más usuales.

Hoy:

- Cómo usar los servicios de TCP/UDP del sistema operativo: el **API de Sockets**

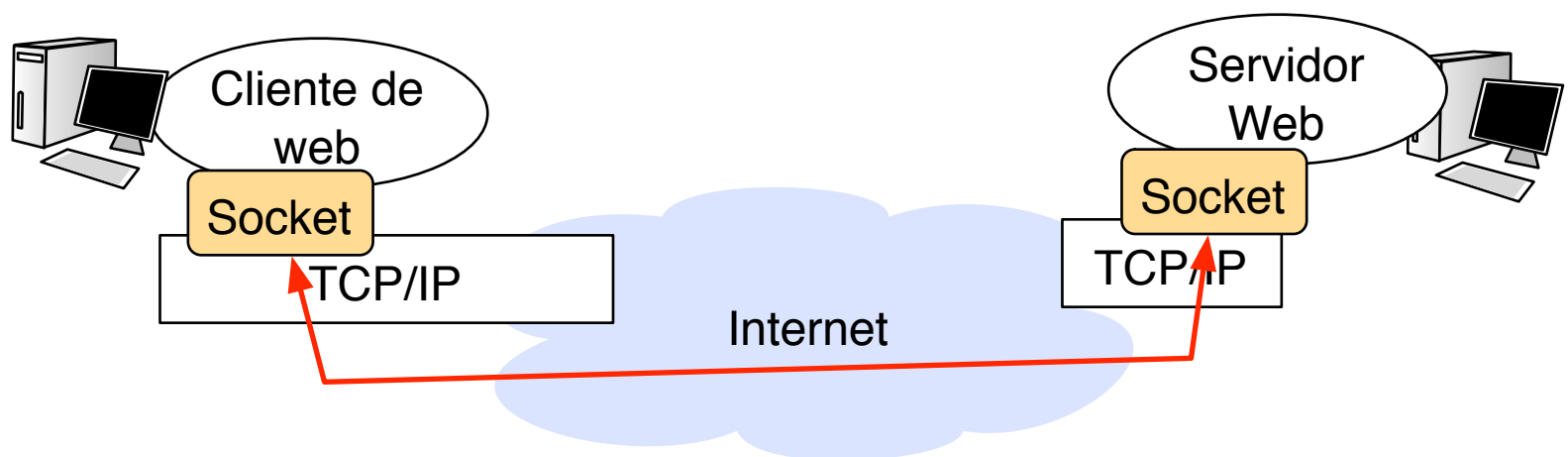
Comunicación entre procesos

- Función del nivel de red: comunicación entre hosts
- Función del nivel de transporte: comunicación entre **procesos**
- Interfaz con el nivel de transporte: Sistema operativo, **sockets**



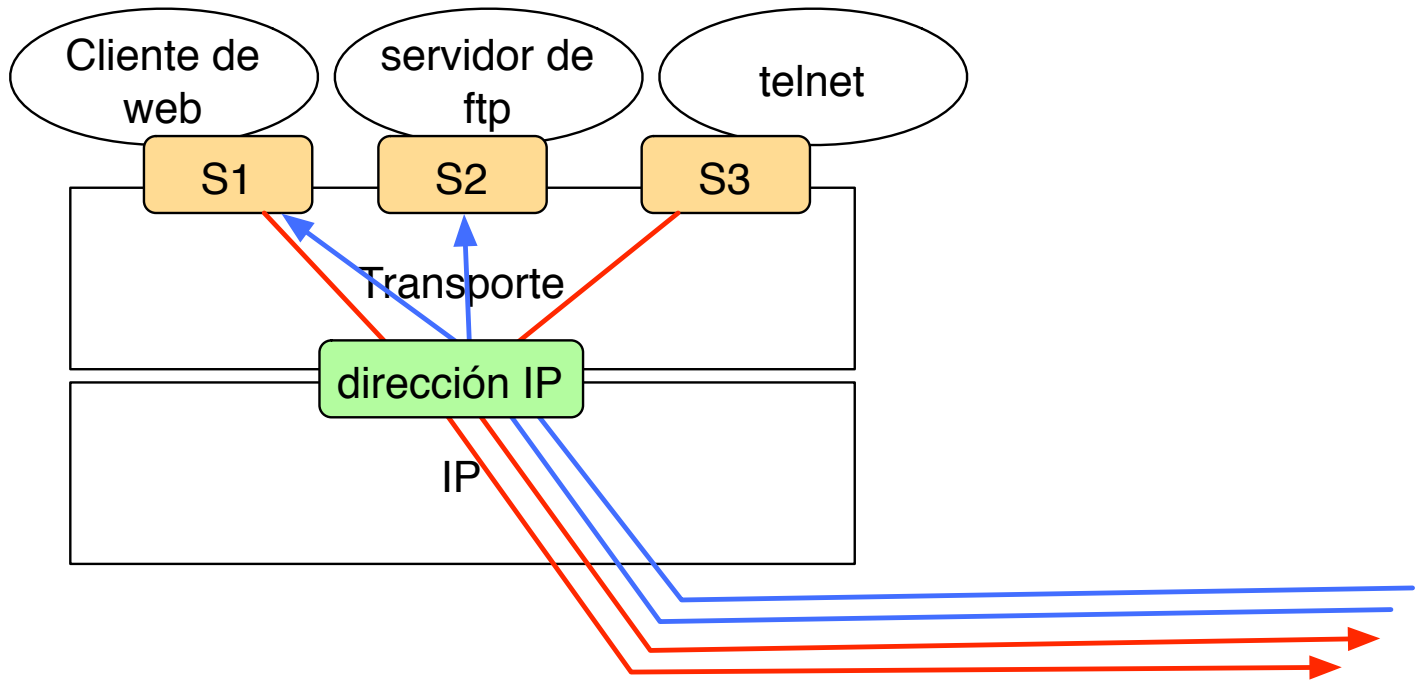
Sockets

- Interfaz con el nivel de transporte proporcionado por el sistema operativo
- Los procesos pueden pedir un socket
  - para usar transporte UDP
  - para usar transporte TCP
  - Se identifican por un descriptor de ficheros
  - Un proceso puede pedir varios sockets



Multiplexación

- El socket es la dirección del proceso
  multiplexación de aplicaciones

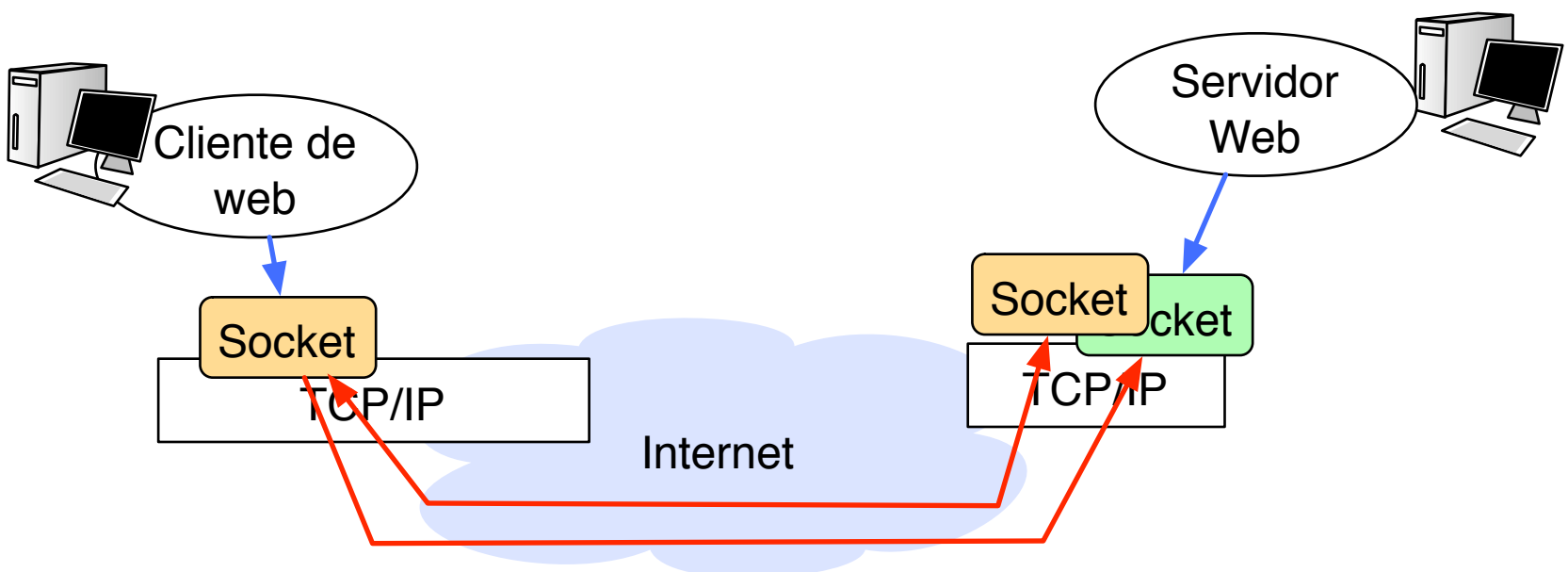


- Al socket se le asocia un dirección (0-65535)
  El **puerto**

Sockets TCP

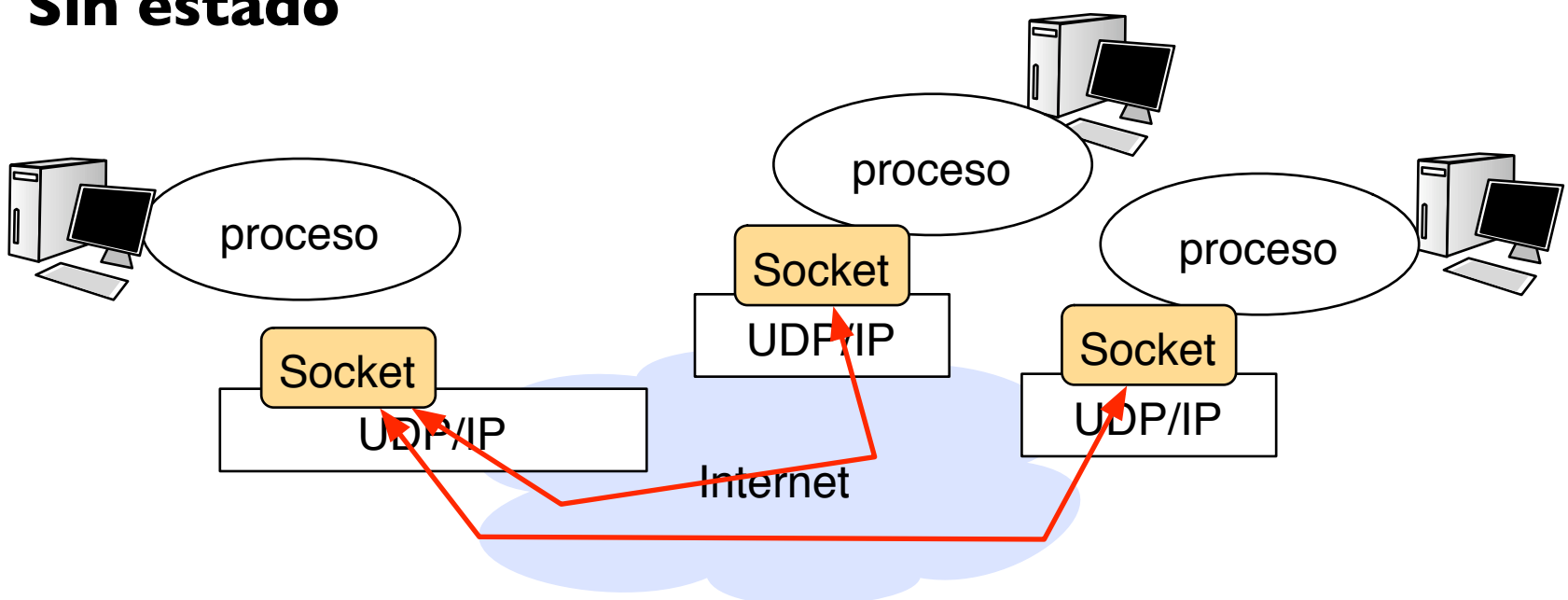
- Orientado a conexión

  Comportamiento de cliente o servidor
  - socket servidor para esperar conexiones
  - socket cliente que inicia una conexión hacia un socket servidor
  - el socket servidor construye un nuevo socket con cada cliente



Sockets UDP

- Orientado a datagramas
  - Un socket ligado a un puerto
  - puede mandar a cualquier otro socket UDP
  - puede recibir de cualquier otro socket UDP
  - **Sin estado**



El API de sockets

- API: Application Program Interface

  Un conjunto de funciones, protocolos y herramientas para ayudar al programador de aplicaciones a utilizar un determinado recurso, en nuestro caso la red
- El API de sockets viene del UNIX de Berkeley BSD 4.2
- Hay otros APIs para usar la red
  - XTI/TLI de System V
  - RPCs (remote procedure calls)
- Pero hoy en día sockets es el más usado

  De hecho es el estandar en Linux, MacOSX y Windows

El API de sockets

Construyendo un socket

- Incluir prototipos para usar sockets

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

- Función `socket()`

```
int socket(int domain, int type, int protocol);
```

protocolo
normalmente no se elige
`0: IP`

descriptor de fichero
-1 : error
>0 : descriptor

dominio
para usar diferentes pilas de protocolos (protocol family)
`PF_INET` (Internet)
`PF_INET6` (IPv6)
`PF_UNIX` (local)
`...`

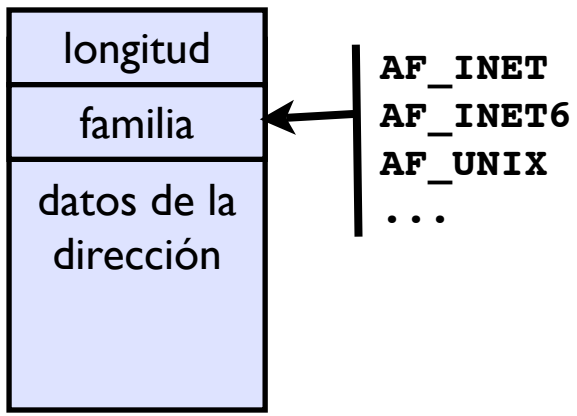
tipo
`SOCK_STREAM` (conexión = TCP)
`SOCK_DGRAM` (datagramas = UDP)
`SOCK_RAW`
`...`

Mas información: `man socket`

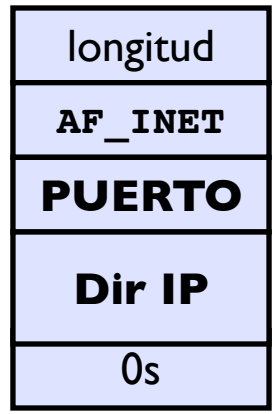
El API de sockets

Manejando direcciones

- La dirección de un socket Inet es { direcciónIP, puerto }
- Las funciones del API de sockets almacenan direcciones con la estructura `struct sockaddr` direcciones de cualquier protocolo




```
struct sockaddr {
    __uint8_t    sa_len;
    sa_family_t  sa_family;
    char         sa_data[14];
};
```




```
struct sockaddr_in {
    __uint8_t    sin_len;
    sa_family_t  sin_family;
    in_port_t    sin_port;
    struct  in_addr sin_addr;
    char         sin_zero[8];
};
```

```
struct in_addr {
    __uint32_t s_addr;
};
```

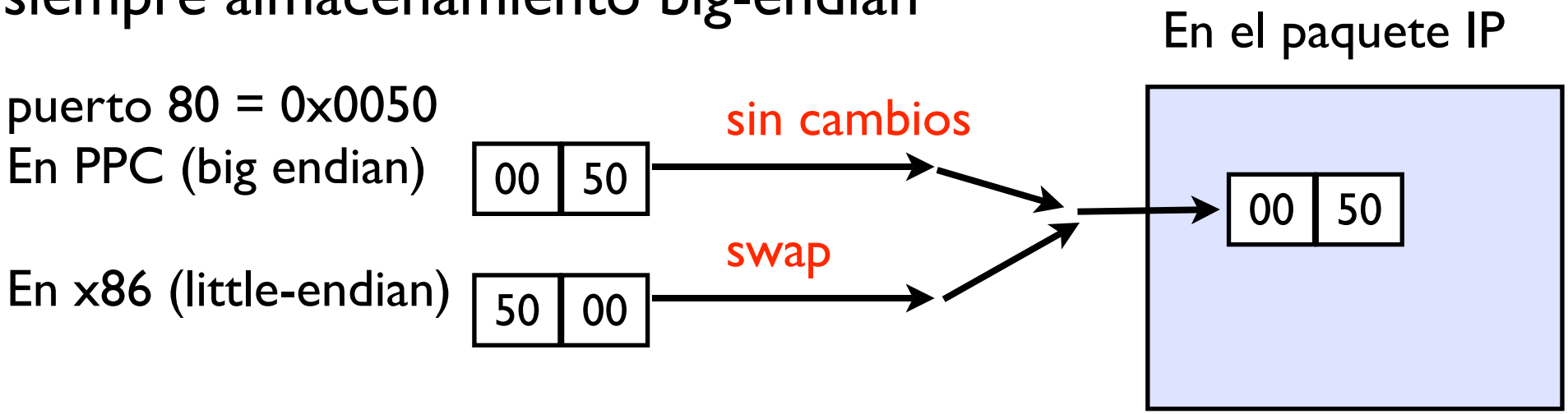
En Linux ni siquiera hay campo **`sin_len`**

La dirección IP es un entero de 32 bits

El API de sockets

Manejando direcciones

- Hay que tener en cuenta que los protocolos de red usan siempre almacenamiento big-endian



- Funciones para cambiar el almacenamiento de un dato
  - `ntohs(), ntohl()` network to host [short o long]

    Usar siempre que leamos un dato que provenga de la red
  - `htons(), htonl()` host to network [short o long]

    Usar siempre que escribamos un dato para la red

El API de sockets

Manejando direcciones

- Ejemplo

  Para rellenar una estructura de dirección para comunicarme con el servidor web en

  { 130.206.160.215, 80 } 130.206.160.215=0x82CEA0D7

```
struct sockaddr_in servidorDir;

servidorDir.sin_family = AF_INET;
servidorDir.sin_port = htons( 80 );
servidorDir.sin_addr.s_addr = htonl( 0x82CEA0D7 );
(struct sockaddr *)servidorDir /* si necesitamos sockaddr */
```

- Veremos métodos más cómodos para manejar direcciones IP y también nombres
- Direcciones especiales INADDR_ANY

El API de sockets

- Antes de enviar datos a otro ordenador con un socket orientado a conexón (TCP) deberemos establecer la conexión
- Función `connect()`

```
int connect(int s, const struct sockaddr *name, socklen_t namelen);
```

error?
-1 : error
0 : ok

socket
establece conexión en este socket

destino
destino de la conexión
Estructura con una dirección { IP , puerto }

convertida a `sockaddr`
`(struct sockaddr *)servidorDir`

destino_len
tamaño de la estructura con el destino

El API de sockets

- Utilizando la conexión
- Una vez establecida la conexión, podemos enviar datos al otro extremo y recibir
- El socket es un descriptor de fichero

  Sólo hay que usar
  - **`read( socket, buffer, tamaño )`** para recibir
  - **`write( socket, buffer, tamaño )`** para enviar
- Con sockets UDP podemos usar
  - **`sendto( ) y recvfrom( )`** sin necesidad de establecer conexión
  - **`connect( ) y read( ) y write( )`** para simular conexiones

Ejemplos: el servicio DAYTIME

- El servicio DAYTIME está disponible en UDP y TCP
  - Un servidor escucha en TCP y UDP en el puerto 13
  - Si recibe una conexión,

    la acepta, envía la fecha y hora actual sin esperar a recibir nada y cierra la conexión
  - Si recibe un datagrama,

    contesta con un datagrama que contiene una cadena de texto con la feche y hora actual al origen del datagrama recibido
- Construyamos con sockets un cliente de DAYTIME...

Ejemplo: cliente TCP

- Abrimos un socket de tipo SOCK_STREAM

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

int main(int argc,char *argv[]) {
    int sock;
     int err;
    struct sockaddr_in servidor;
    char buf[2000];
    int leidos;

    /* Abrimos el socket */
    sock=socket(PF_INET,SOCK_STREAM,0);
    if (sock==-1) {
        printf("Error no puedo abrir el socket\n");
        exit(-1);
    }
```

Necesario para usar sockets

Necesario para sockaddr_in

```
#include <stdio.h>
```

Ejemplo: cliente TCP

- Rellenamos la estructura con la dirección del servidor de DAYTIME, en este caso 10.1.1.22 (0x 0A 01 01 16)
- Hacemos un connect() con la estructura

```
/* Abrimos el socket */
sock=socket(PF_INET,SOCK_STREAM,0);
if (sock==-1) {
    printf("Error no puedo abrir el socket\n");
    exit(-1);
                              }

/* Rellenamos la estructura de la direccion */
servidor.sin_family=AF_INET;
servidor.sin_port=htons(13);
servidor.sin_addr.s_addr=htonl(0x0A010116);

/* Conexion al servidor TCP */
  err=connect(sock,
       (struct sockaddr *)&servidor,
                              sizeof(servidor));
 if (err==-1) {
    printf("Error no puedo establecer la conexion\n");
    exit(-1);
 }
/* No hace falta escribir nada porque el servidor TCP sabe
```

Ejemplo: cliente TCP

- Leemos lo que se recibe por la conexión y lo imprimimos

```
/* Abrimos el socket */
...
        (struct sockaddr *)&servidor,
                                sizeof(servidor));
    if (err==-1) {
       printf("Error no puedo establecer la conexion\n");
       exit(-1);
    }
    /* No hace falta escribir nada porque el servidor TCP sabe
    que ha aceptado una conexion */
    write(sock,"eoo",3);

    /* Esperamos la respuesta */
    leidos=read(sock,buf,2000);
    if (leidos>0) {
       /* Terminamos la cadena y la imprimimos */
       buf[leidos]=0;
       printf("He leido: [%d]: _%s_\n",strlen(buf),buf);
     }
}
```

Escribir

Leer

Ejemplo: cliente UDP

- Para hacerlo en UDP se cambia a tipo `SOCK_DGRAM`
- `connect( ) read( ) y write( )` se pueden usar igual

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

int main(int argc,char *argv[]) {
    int sock;
    struct sockaddr_in servidor;
    char buf[2000];
    int leidos;

    /* Abrimos el socket */
    sock=socket(PF_INET,SOCK_DGRAM,0);
    if (sock==-1) {
        printf("Error no puedo abrir el socket\n");
        exit(-1);
     }
```

Ejemplo: otro cliente UDP

- O bien usamos las funciones

  `sendto()` y `rcvfrom()` y no usamos `connect()`

```
    servidor.sin_port=htons(13);
    servidor.sin_addr.s_addr=htonl(0x0A010116);

    printf("Preguntemos la hora al servidor de DAYTIME\n");

    sendto(sock,"eoo",3,0,(struct sockaddr *)&servidor,sizeof(servidor));
    /* El servicio de daytime contesta a cualquier cosa*/

    /* Esperamos la respuesta */
        quienl=sizeof(quien);
    leidos=recvfrom(sock,buf,2000,0,(struct sockaddr
      *)&quien,&quienl);
    if (leidos>0) {
       /* Terminamos la cadena y la imprimimos */
       buf[leidos]=0;
       printf("He leido: [%d]: _%s_\n",strlen(buf),buf);
     }
}
```

Enviar

Recibir

Conclusiones

- Para usar TCP o UDP desde una aplicación usamos el API de sockets
- Sabemos:

  Crear sockets

  Manejar direcciones

  Hacer conexiones (como cliente)

  Enviar y recibir por las conexiones
- Próxima clase:

  Usando sockets TCP y UDP (cliente y servidor)