

Sockets

Introducción

- Un socket es un punto final de comunicación
 - (socket => dirección IP + puerto + protocolo)
- Aparecieron en 1981 en el UNIX BSD 4.2
 - Intento de incluir TCP/IP en UNIX
 - El diseño de los sockets es independiente de
 - El protocolo de comunicación (soportan unos 25 protocolos)
 - El lenguaje de programación y
 - son independientes puedo hablar con sockets python y go
 - el sistema operativo
- Un socket es una abstracción que:
 - ofrece interfaz de acceso a los servicios de red en el nivel de transporte
 - Representa un extremo de una comunicación bidireccional con una dirección asociada
 - Ejemplo el socket TCP [172.25.4.4/1521](http://172.25.4.4:1521) se refiere a puerto TCP 1521 en la dirección IP 172.25.4.4
 - La comunicación se produce siempre entre dos sockets

Nivel de transporte

- Mueve bits entre procesos

Socket es como un enchufe punto final de comunicación

Sockets pertenecen a los procesos

El kernel se divide en dos subsistemas

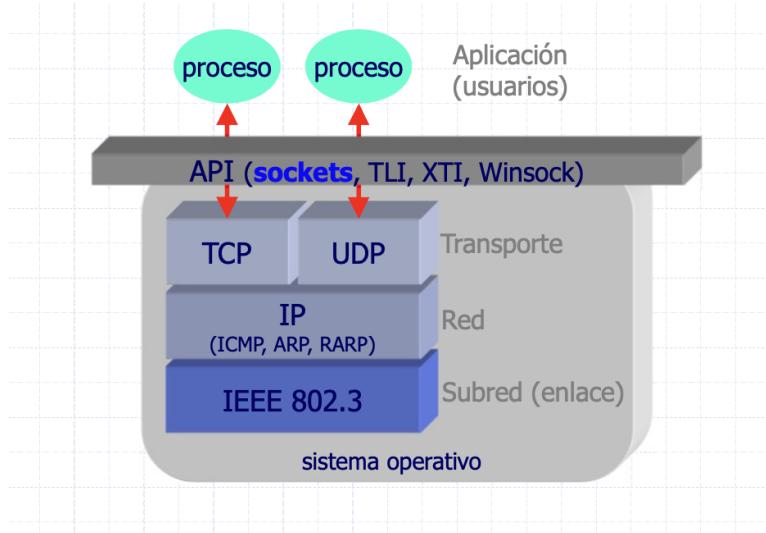
- Ficheros y procesos
- Todo se representa con ficheros en un sistema unix

Sockets BSD

- Sujetos a proceso de estandarización dentro de POSIX
 - Portable Operating System Interface Unix
 - Portable lo pueden usar cualquier sistema Unix que este basado en POSIX
 - linux es el kernel más posix
 - MacOS tiene un kernel llamado Darwin
 - Posix están más currado usan enteros de 32 bits sin signo
 - Dos modos
 - modo usuario y privilegiado
 - llamadas al sistema
 - kernel el único que corre en modo privilegiado
 - El dueño de todos los recursos de la máquina es el kernel
 - Interfaz sockets son llamas al sistema
 - Actualmente

- Disponibles en casi todos los sistemas Unix
- En prácticamente todos los sistemas operativos
- Competidores
 - TDI
 - XTI
- El 99.99999% de las comunicaciones en internet se realizan mediante sockets

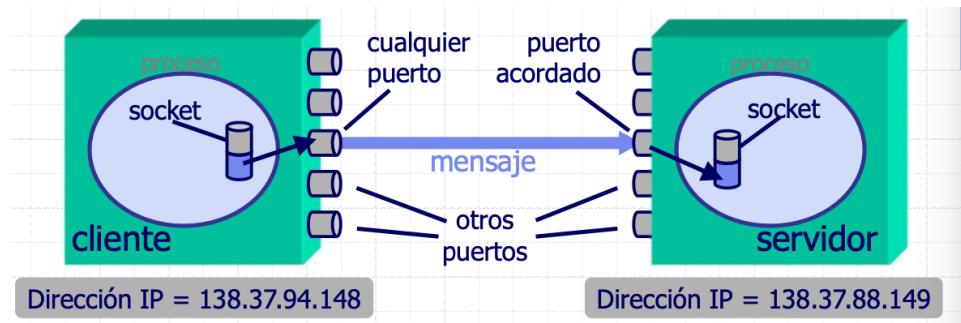
APIs de acceso al nivel de transporte



- IEEE 802.3 = Ethernet
- Las de ethernet son las mac
- IP mi tarjeta de red habla con la tuya
 - interfaz con interfaz
- Kernel es dueño de todo el sistema operativo
- Proceso es un programa en ejecución
- IP
 - hablar entre máquinas
 - hablan tarjetas de red
- UDP
 - Mando paquetes
 - duplicar
 - ruido
 - no llegar
 - llegar en desorden
 - multiplexar
 - transporte procesos son capaces de hablar
 - Puertos distinguen a los diferentes procesos
- TCP
 - circuitos
 - establece un circuito virtual
 - cable
- Subred sin interfaces tiene la loopback 127.0.0.1 localhost
 - send packet y receive packet entre ip y ieee

- Parte de arriba de la subred es un driver controlador
 - software

Sockets y puertos



Socket = conector

- Ligado a una pareja (dirección IP + Puerto) y a un protocolo (TCP, UDP)
 - El socket es un elemento del proceso
 - El puerto es un elemento del sistema operativo
- Hay 2^{16} puertos posibles (algunos reservados)
- No se puede reabrir un puerto ya asignado a otro proceso

Modelo Cliente-Servidor con Sockets

- Los dos procesos que se comunican no son iguales
 - Cliente habla primero
 - responde al cliente
- Servidor son procesos que están esperando peticiones durmiendo
- Servidor
 - proceso que se ejecuta en un nodo de la red y que proporciona un determinado recurso o servicio (está continuamente esperando peticiones)
 - Servidor interactivo
 - El mismo servidor recibe la petición y la atiende
 - Si el servidor es lento en atender a los clientes, se pueden producir grandes esperas
 - Responde el propio servidor
 - Servidor concurrente
 - El servidor recibe las peticiones y genera por cada una un proceso que se encarga de atenderlas
 - Solo aplicable en sistemas multiproceso como UNIX
 - Responde otro
- Cliente
 - El proceso que se ejecuta en otro nodo (o el mismo) y realiza peticiones al servidor
 - Es el proceso que inicia y termina el diálogo

Implementación socket cliente-servidor

- Servidor
 - Creación y enlace al puerto de servicio de un socket
 - Desconexión del servidor de su terminal de lanzamiento o ejecución
 - Bucle infinito en el cual el servidor
 - Espera una petición
 - La trata
 - Calcula y formatea la respuesta
 - Envía la respuesta
- Cliente
 - Creación del socket local
 - Preparación de la dirección del servidor
 - Envío del mensaje
 - Espera del resultado
 - Explotación del resultado // Imprimirlo

UDP

- DNS
- Hacer una petición y tiene respuesta
- si se pierde la vuelvo a hacer

Orden

- Timestamps
- Tengan un id los paquetes

Un socket representa un extremo de una comunicación de datos

- Procesos y ficheros
 - Sockets tiene que ver con ficheros tiene un fd
 - read
 - write
 - close

Sockets son bidireccionales

- Escribir y leer a la vez
 - autovia dos sentidos

Conceptos básicos sobre sockets

- Dominios de comunicación
 - Un dominio representa una familia de protocolos
 - Un socket está asociado a un dominio desde su creación
 - Cada dominio tiene su propio formato de direcciones
 - Cuando dos protocolos son familia

- Cuando usamos un protocolo solo usamos ese protocolo para hablar entre si
- Veo transporte abajo soy aplicación
- Son de la misma familia si usan el mismo formato de direcciones
 - UDP y TCP son de la misma familia
 - TCP y bluetoth no son de la misma familia
- Los servicios de sockets son independientes del dominio
 - Solo se pueden comunicar sockets del mismo dominio
 - segundo parametro es el tipo de protocolo que se usa
- Algunos ejemplos (presentes en el Unix 4.3 BSD)
 - Dominio PF_UNIX o PF_LOCAL dentro de una máquina
 - procesos dentro de una. máquina
 - super tuberías
 - Las direcciones son nombres de ficheros
 - Dominio PF_INET comunicación usando protocolos TCP/IP (válido para internet usado universalmente)
 - Dominio PF_APPLETALK protocolo de AppleTalk de Apple
- No son la misma familia Ip4 y ip6 son diferentes dominios de comunicación

Token Ring

Internetwork

- hablamos diferentes redes de diferente tipo heterogeneo

Subnetwork

- hablamos la misma red homogeneo

PF = ProtocolFormat

AF = addressFormat

- AF inet

Valen lo mismo

Tipos o estilos de sockets

- El tipo o estilo de un socket recoge el conjunto de propiedades del servicio que se desean
 - De forma independiente del dominio de comunicaciones

Los tipos más populares e importantes son:

- Sockets tipo stream
 - circuitos
- Sockets tipo datagrama
 - conmutación paquetes

Otro tipos de sockets

- Sockets tipo crudo (raw)
- Otros tipos

- Sequenced packet sockets
- Reliably delivered message sockets

Sockets tipo stream

- Stream
 - reserva un camino cuando se hace la conexión
- Stream (SOCK_STREAM)
 - Representa un circuito virtual u orientado a conexión: al conectar se realiza una búsqueda de un camino libre entre origen y destino y se mantiene el camino en toda la conexión
- Propiedades
 - Orientado a conexión
 - De una conexión full duplex
 - le digo la ip a la que quiero hablar y se crea un camino que se mantienen hasta que se acabe la conexión
 - Fiable
 - Asegura que no se pierden ni se duplican datos
 - Secuencial
 - Asegura el orden de entrega de los datos
 - No mantiene separación entre mensajes (byte stream)
 - Permite el envío de mensajes fuera de banda (out of band)
- En el dominio PF_INET se corresponden con el protocolo TCP
- En el dominio PF_UNIX son como una FIFO full-duplex
- Stream
 - es un flujo constante

Sockets tipo Datagrama

- Datagrama
 - red basada en paquetes
 - No existe una reserva de camino
 - Paquete va por donde quiere
 - Mandas cartas
- Datagrama (SOCK_DGRAM)
 - Representa una red basada en datagramas (no orientada a conexión) no se fija un camino; cada paquete podrá ir por cualquier sitio
- Propiedades
 - best effort
 - hacer el mejor esfuerzo para cumplir
 - Sin conexión (posible "pseudoconexión" gracias al uso de direcciones implícitas)
 - No fiable, no se asegura el orden en la entrega
 - No se garantiza la recepción secuencial (es decir, el orden) de los datos
 - Mantiene la separación entre mensajes
- En el dominio PF_INET se corresponden con el protocolo UDP

Otros tipos de sockets

- Raw(SOCK_RAW)
 - Túneles de servicio
 - Wireshark
 - monitor modo promiscuo
 - veo todo sea para mi o no
 - Permite el acceso a los protocolos de niveles más bajos (IP, la propia ethernet, etc)
 - Requieren ser superusuarios (root) para su utilización
 - Pueden usarse para implementar nuevos protocolos de transporte RDP
- Otros tipos de sockets
 - Sequenced packet sockets (SOCK_SEQPACKET):
 - Solo presentes en el dominio PF_NS
 - Como los sockets stream pero con límites entre mensajes
 - Reliably delivered message sockets (SOCK_RDM):
 - Supuestamente transfieren mensajes garantizando su entrega
 - Nunca se han implementado

Direcciones de sockets

- Un socket debe tener asignada una dirección única
 - sockets tienen direccionamiento directo
 - los dos extremos tienen dirección origen y destino
 - Las direcciones son dependientes del dominio
 - Usos
 - Asignar una dirección local a un socket (bind())
 - bind
 - asignar una dirección al socket
 - le paso la dirección
 - Especificar una dirección remota(connect(), sendto())
- Formatos de direcciones: cada dominio usa una estructura específica
 - Direcciones en PF_INET: dirección IP + puerto
 - Formato: socket.AF_INET
 - ejemplo ("172.25.3.33", 4567)
 - te convierte la ip dotted decimal
 - Direcciones en PF_UNIX: path de un fichero
 - Formato: socket.AF_UNIX
 - ejemplo: "/tmp/socket"
- Buffer dirección más tamaño

TCP es big endian

intel little endian

arm bi endian pero lo configuran little endian

Direcciones de sockets en PF_INET

- Una dirección de Internet viene determinada por:

- Dirección IP del host 32 | 128 bits
- Puerto de servicio 16 bits
- Una transmisión se caracteriza por 5 parámetros únicos
 - (Dirección host + puerto) origen
 - (Dirección host + puerto) destino
 - Protocolo de transporte (UDP o TCP)
- Al pasar las direcciones a la pila TCP/IP
 - Han de codificarse siempre en el orden de bytes de red (big endian)
 - Constantes
 - INADDR_LOOPBACK("127.0.0.1") - Interfaz loopback
 - INADDR_ANY("", "[0.0.0.0](#)") - cualquier dirección entrante
 - [0.0.0.0](#) comodín entrante
 - cualquiera que tenga activada en mi máquina
 - mínima 2
 - loopback
 - la de wifi
 - no sirve para salir es para entrar
 - INADDR_BROADCAST("<broadcast>") - envío de broadcast
 - INADDR_NONE("255.255.255.255") - indica un error
 - -1
 - invalida
 - no puede configurar una interfaz a esta dirección
 - reservada
 - DHCP
 - Dynamic Host Configuration Protocol
 - lo arranco y le dan una ip mascara broadcast dns

Gráfico de estructuras de direcciones

- Comparación visual de las estructuras en C de direcciones de las diferentes familias de sockets
 - type tag
 - constante que dice el tipo
- Un socket AF_INET solo trabaja con direcciones af_inet

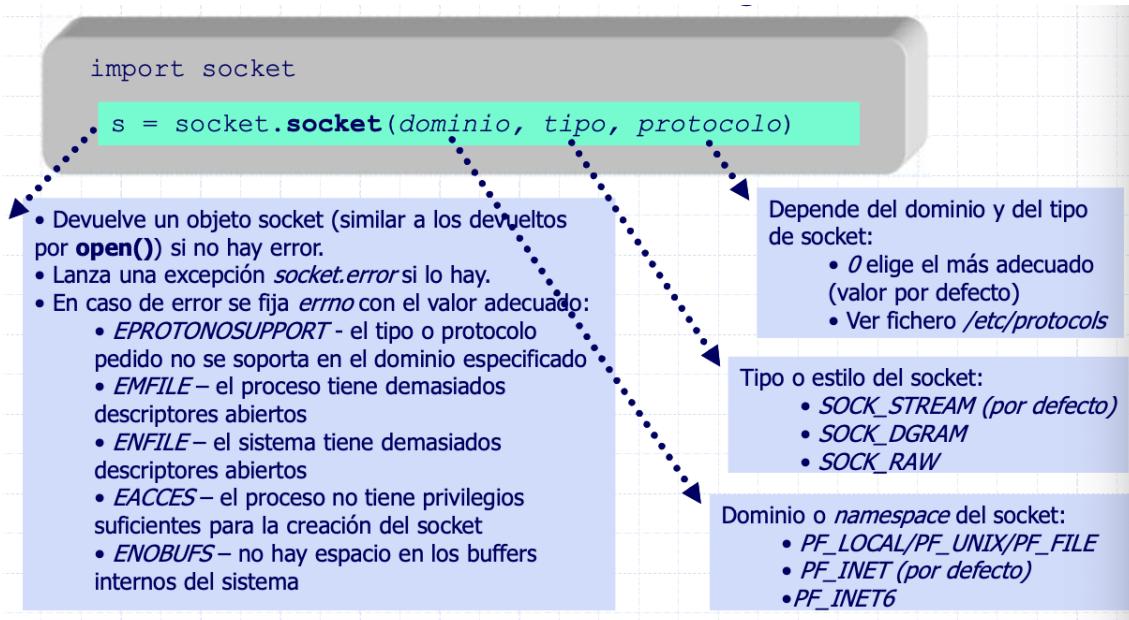
Mapa de la API de sockets

- API de sockets
 - Creación
 - Socket(): crear un socket
 - bind(): asignar una dirección a un socket
 - Preparación de la conexión
 - listen(): esperar conexiones en modo pasivo
 - connect(): iniciar una conexión a otro socket
 - accept(): aceptar una conexión
 - Transferencia de datos
 - write()/send()/sendto(): escribir datos en un socket

- `read()/recv()/recvfrom()`: leer datos de un socket
- Despedida y cierre
 - `close() / shutdown()` cerrar un socket eliminando la conexión
- Otras llamadas de interés

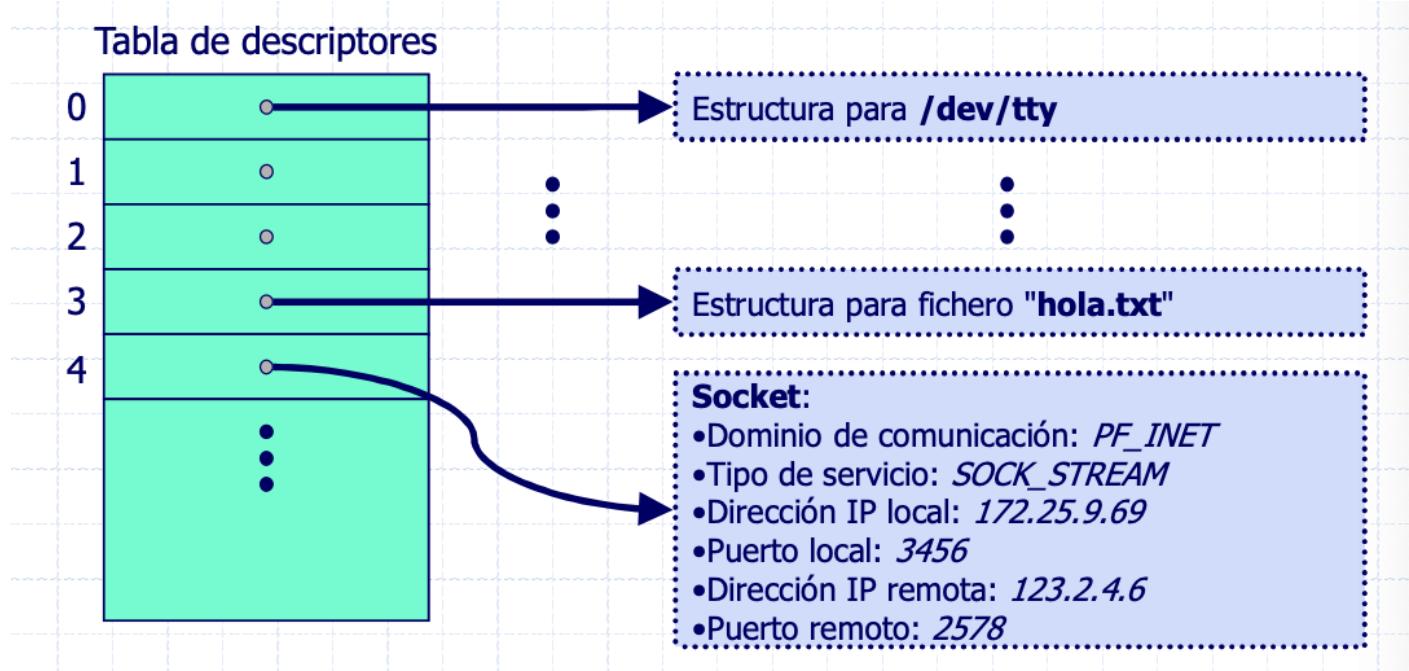
Creación de un socket

- La llamada `socket()` crea un socket
 - El socket creado no tiene dirección asignada



Socket como descriptor de fichero

- Un objeto socket una vez creado contiene un socket (abierto) del sistema operativo
- Dicho socket no es otra cosa que un descriptor de fichero de unix (entero pequeño)



Asignación de direcciones

- Asignación de una dirección a un socket ya creado

- Si no se asigna dirección / puerto (típico de clientes) se le dara automáticamente en su primer uso

```
import socket
s.bind(direccion)
```

- En caso de error lanza una excepción `socket.error` y fija `errno` con el valor adecuado:
 - `EBADF` – s no es un descriptor de fichero válido
 - `ENOTSOCK` – s no es un socket
 - `EADDRNOTAVAIL` – la dirección no está disponible en esta máquina
 - `EADDRINUSE` – la dirección está siendo usada por otro socket
 - `EINVAL` – el socket ya tiene dirección asignada
 - `EACCES` – el socket no tiene privilegios suficientes para acceder a la dirección pedida (sólo root puede acceder a los puertos de 0 a `IPPORT_RESERVED` en el dominio `PF_INET`)

- Dirección a asignar al socket.
- Formato dependiente del dominio.
- `INADDR_ANY` pone la dir. por defecto.
- Puerto `0` elige un puerto efímero.

Socket ya creado.

- Si soy un servidor
 - tengo que saberme mi puerto
 - tiene la dirección any
 - 0.0.0.0
 - puerto específico donde escucha el servidor
- si soy el cliente
 - pongo mi ip o 0.0.0.0
 - y el puerto que este libre
 - puerto 0 elige cualquier puerto
 - bind no hace el cliente
- No es lo mismo ser el cliente que el servidor

Asignación de direcciones en PF_INET

- Direcciones en dominio PF_INET
 - Host una dirección IP de la máquina local
 - `INADDR_ANY` elige cualquiera de la maquina
 - Puertos
 - El rango de puertos es 65535
 - Si se les indica el puerto 0, el sistema elige uno adecuado
 - Si el puerto solicitado esta ya asignado la llamada `bind()` devuelve un valor negativo
 - El espacio de puertos para streams (tcp) y datagramas (udp) es independiente
 - son protocolos diferentes

Tipo de puertos	Rangos	Descripción
<i>Reservados</i> 0 .. 1023 ↑ (<code>IPPORT_RESERVED</code>)	0 .. 255	aplicaciones públicas: <ul style="list-style-type: none"> ftp: 20 y 21 telnet: 23 SMTP: 25 www-http: 80
	255 .. 1023	aplicaciones que necesiten privilegios de superusuario
<i>No reservados</i>	1024 .. 4999	usados por procesos de usuario y del sistema

NO RESERVADOS	1024...1000	usados por procesos de usuario y del sistema
1024..65535	> 5000	usados sólo por procesos de usuario

Solicitud de conexión

- Realizada en el cliente con la llamada connect():
 - solo se usa en sockets tipo stream
 - connect conecta
 - connect le doy la dirección del servidor
 - lanzo una petición de conexión
 - puede fallar
 - tiene que devolver un ACK para confirmar que estamos bien
 - Si el cliente no ha asignado dirección/puerto local al socket (ver bind()), se le asigna una automáticamente
 - Normalmente se usa con sockets tipo stream (bloqueante)
 - Connection refused
 - remoto rechaza la conexión
 - llego hasta la máquina remota
 - me equivoco de puerto
 - Timeout
 - Si no recibo nada durante un tiempo
 - Inalcanzable (host unreachable)
 - Máquina apagada
 - A nivel local se si la máquina esta apagada o prendida
 - Network Unreachable
 - estar en otra red
 - router nos dice que no se puede llegar a la red
 - firewall
 - interfaz entrada y salida
 - motor de reglas pasa o no pasa
 - decide que entra y que no basado en reglas
 - drop tirar el paquete
 - best effort

```
import socket
```

```
s.connect(direccion)
```

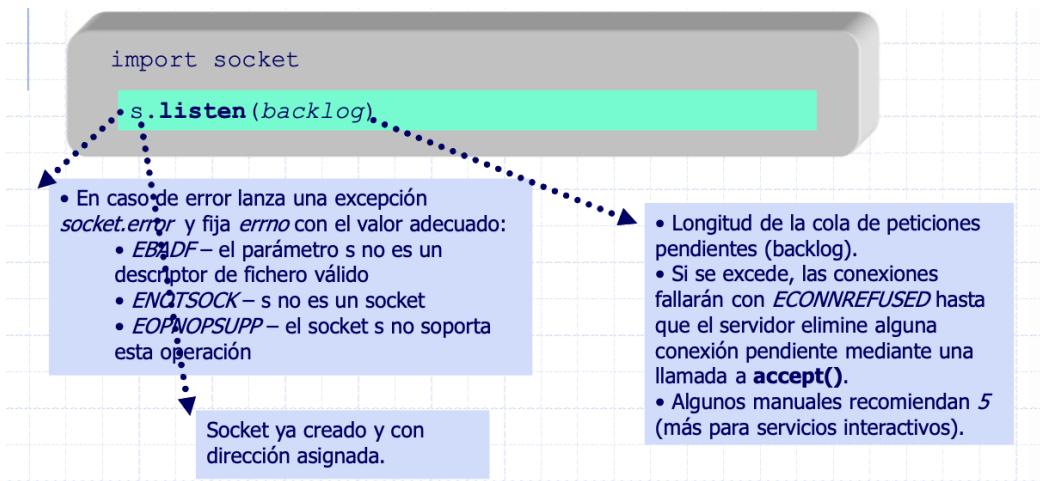
• En caso de error lanza una excepción `socket.error` y fija el error con el valor adecuado.

- `EBAUD`, `ENOTSOCK`, `EADDRINUSE`, `EADDRNOTAVAIL` (dirección remota no disponible), `EAFNOSUPPORT` (tipo de socket no soportado), `EISCONN` (ya conectado), `ETIMEOUT` (time out), `ECONNREFUSED` (remoto rechaza la conexión), `EHOSTUNREACH` (host inalcanzable), `ENETUNREACH` (red remota inalcanzable).
- `EINPROGRESS` – s es no bloqueante y la conexión no puede establecerse de inmediato; `select()` permite saber si la conexión está por fin establecida o no; otra llamada a `connect` fallará con `EALREADY`
- `EALREADY` – s es no bloqueante y ya hay una conexión pendiente en progreso

- Dirección del socket remoto.
- Su formato es dependiente del dominio.

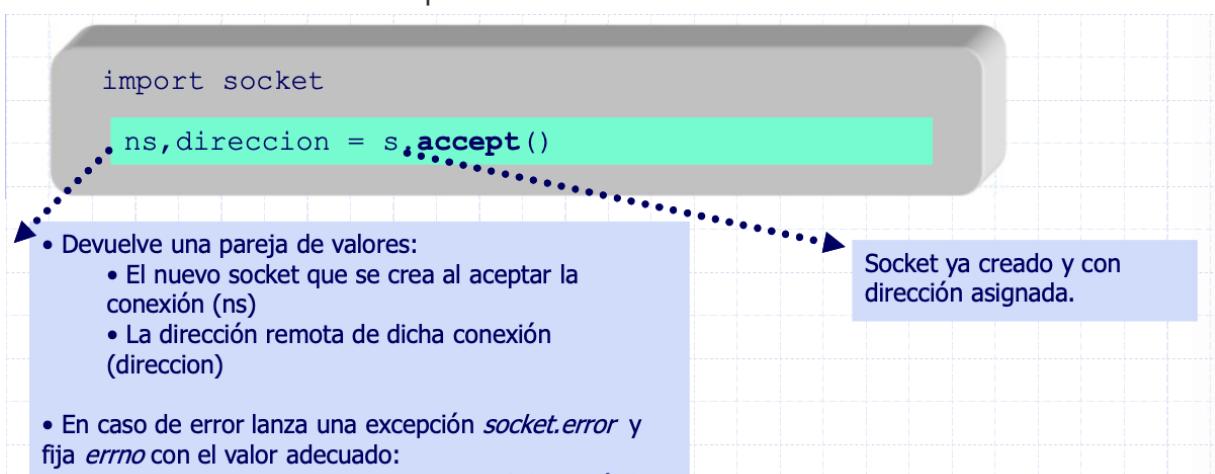
Socket ya creado.

- La llamada listen() deja al socket preparado para aceptar conexiones
 - Realizada en un servidor de sockets stream tras haberlo creado (socket()) y reservado dirección (bind())
 - Convierte el socket en un socket de escucha
 - Queda preparado para recibir paquetes
 - backlog
 - bajos
 - 1
 - 0
 - no tiene cola
 - servidor esta libre o no
 - Solo hay tráfico de conexiones



Aceptar una conexión

- Realizada con accept() en un servidor de sockets stream tras preparar la conexión (listen())
- accept devuelve la dirección con la que estoy hablando
 - crea un nuevo socket
 - tenemos el socket escuchando 2
 - 1 de escucha y 1 de ns
 - cada socket nuevo está concentrado en su cliente
 - Servidor es concurrente
 - crea un fork para el nuevo socket



- *EBADE* – s no es un descriptor de fichero válido
- *ENOTSOCK* – s no es un socket
- *EOPNOPSUPP* – el socket s no soporta esta operación
- *EWOULDBLOCK* – s es no bloqueante y no hay conexiones pendientes

Semántica de la aceptación de conexión

- La semántica de la llamada `accept()` es:
 - Sólo se usa en el lado servidor, en sockets tipo stream
 - Extrae la primera petición de la cola creada con `listen()`
 - Cuando se produce la conexión, el servidor obtiene como valor de retorno de la llamada:
 - La dirección del socket remoto del cliente
 - Un nuevo descriptor (nuevo socket local) que queda conectado al socket del cliente cuya petición se ha extraído
 - Al retornar `accpet()` quedan activos dos sockets en el servidor
 - El original para aceptar nuevas conexiones
 - El nuevo para enviar/recibir datos por la conexión que se acaba de establecer
 - Gracias a este sistema se pueden plantear servidores multiproceso o multithread para servicios concurrentes

Envío y recepción de datos

- Envío
 - Llamadas al sistema para escribir datos en un socket
 - `Write()`, `send()`, `sendto()`
 - La llamada `write()`
 - Es la misma que se usa con ficheros (módulo os)
 - El descriptor a usar aquí sera el descriptor del socket
 - Puede escribir en cualquier descriptor, sea fichero o socket
- Recepción
 - Llamadas al sistema para leer datos de un socket `read()`, `recv()`, `recvfrom()`
 - La llamada `read()`
 - Es la misma que se usa con ficheros
 - El descriptor a usar aquí será el descriptor del socket
 - Puede leer de cualquier descriptor, sea fichero o socket

Envío y recepción en sockets stream

- Envío

» Envío:

```
import socket
```

```
s.send(datos, flags=0)
```

- Devuelve el nº de bytes enviados
- En caso de error lanza una excepción `socket.error` y fija `errno` con el valor adecuado

Aspectos avanzados,
p.ej. `MSG_OOB` (datos
fuera de banda).

- Con `flags=0, send()` es equivalente a `write()` en
objetos de tipo fichero (los devueltos por `open()`)

- Send es mas que un write

Recepción

```
import socket
    datos = s.recv(maxtam, flags=0)
```

- Con flags=0, recv() es equivalente a read() en objetos de tipo fichero (los devueltos por open())

- Receive es básicamente un read
 - dar tamaño máximo que voy a leer

Envío y recepción en sockets datagrama

- No se establece conexión (connect() / accept())
 - Para usar un socket para transferir basta con crear el sockets y reservar la dirección bind()

Envío

» Envío:

```
import socket
    s.sendto(datos, direccion, flags=0)
```

- En *direccion* irá la dirección del destinatario.
- Por lo demás, exactamente igual que send()

» Recepción:

- enviar unos datos a una dirección
 - Send to

Recepción

```
import socket
    datos,direccion = s.recvfrom(maxtam, flags=0)
```

- En *direccion* aparece la dirección del remitente.
- Por lo demás, exactamente igual que recv()

- Puede usarse recv() si no importa la dirección del destinatario, e incluso read() si no se usan los flags.

- Recibir un paquete from
 - Si te mando 3 sentTo tenemos que hacer 3 receiveFrom
 - como es udp capaz ni te llegue
 - cuanto tiempo tengo que esperar cuando lo haya recibido
 - nada no sabe si le llega o no lo manda y ya
 - Mando una carta
 - El tiene que mandar uno de respuesta
 - mayor apps udp
 - sendto
 - receive from
 - y luego lo mismo pero al revés para decir que ya lo recibí
 - si recvfrom no le llega ningún paquete
 - se puede quedar bloqueado para siempre

Cierre un socket

- Puede usarse la clásica llamada close():
 - Cierra ambos sentidos de la conexión en sockets streams

```
import socket
s.close()
```

- servidor tiene que tragarse los close
- La de udp no hay conexiones
 - no hay close

Otra opción es usar shutdown()

```
import socket
s.shutdown(cómo)
```

- En caso de error lanza una excepción `socket.error` y fija `errno` con el valor adecuado:
 - `EBADF` – s no es un descriptor válido
 - `ENOTSOCK` – s no es un socket
 - `ENOTCONN` – s no está conectado
- El parámetro `como` dice cómo se cierra:
 - `SHUT_RD` – dejar de recibir datos en el socket; si llegan más, se rechazan.
 - `SHUT_WR` – dejar de transmitir datos, descartando todos los que quedasen por transmitir y reconocimientos o retransmisiones de los ya transmitidos.
 - `SHUT_RDWR` – `SHUT_RD + SHUT_WR`, es decir, igual a cerrar la conexión con `close()`.

Sockets como ficheros

- Un socket puede tratarse como un fichero con la API de streams del lenguaje makefile():

```
import socket
fichero = s.makefile([modo], [tambuffer])
```

- Devuelve un objeto fichero

Mismos parámetros que la llamada de apertura de ficheros: `open()`.

- También puede fijarse el timeout de una conexión:

```
import socket
s.settimeout(segundos)
```

- Fija el `timeout` de un intento de conexión (en segundos).

- Si ponemos un timeout muy bajo capaz no llega y vuelve

Sockets en lugar de pipes

- Un par de sockets es una estructura similar a una tubería sin nombre pipe; lo crea socketpair()
 - Crear una pareja de sockets sin nombre y conectados
 - Diferencia ambos sockets son bidireccionales, no como los descriptores de la llamada pipe ()

```
import socket
```

```
s1,s2 = socket.socketpair(dominio, tipo,  
                           protocolo)
```

- Devuelve una pareja de sockets (s1,s2) conectados entre sí, igual que en **pipe()**
- En caso de error lanza una excepción *socket.error* y fija *errno* con el valor adecuado:
 - *EMFILE* – el proceso tiene demasiados descriptores abiertos
 - *EAFNOSUPPORT* – dominio no soportado
 - *EPROTONOSUPPORT* – protocolo no soportado
 - *EOPNOTSUPP* – el protocolo no permite crear parejas de sockets

Sólo tiene sentido 0

Soportados todos pero ambos sockets se conocen incluso con tipo no orientado a conexión.

Ha de ser *PF_LOCAL*

Direcciones local y remota de un socket

- Las funciones *getsockname()* y *getpeername()* obtiene información a partir del descriptor socket:
 - *get sockname()* obtiene la dirección asociada al socket local
 - *get peername()* obtiene la dirección del socket remoto conectado al socket local (si existe tal conexión)

```
import socket  
s1,s2 = socket.socketpair(dominio, tipo,  
                           protocolo)
```

- Devuelve una pareja de sockets (s1,s2) conectados entre sí, igual que en **pipe()**
- En caso de error lanza una excepción *socket.error* y fija *errno* con el valor adecuado:
 - *EMFILE* – el proceso tiene demasiados descriptores abiertos
 - *EAFNOSUPPORT* – dominio no soportado
 - *EPROTONOSUPPORT* – protocolo no soportado
 - *EOPNOTSUPP* – el protocolo no permite crear parejas de sockets

Sólo tiene sentido 0

Soportados todos pero ambos sockets se conocen incluso con tipo no orientado a conexión.

Ha de ser *PF_LOCAL*

Comandos y llamadas de utilidad

- Existen algunos comandos Unix que son de en la programación con sockets:
 - ver conexiones de sockets abiertas en el sistema
 - *netstat -a*
 - Ver las llamadas al sistema que realiza un proceso
 - *strace proceso*
 - *-f*
 - *follow forks*
- También existen algunas llamadas de utilidad en el módulo os
 - sincronizan un descriptor de fichero
 - *os.fsync(fd)*
 - *sincronizar*
 - *poner al día*
 - *limpiar buffers intermedios*
 - *fflush*

Configuración de opciones en sockets

- Existen varios niveles de opciones dependiendo del protocolo afectado como parámetro:
 - Opciones independientes del protocolo socket.SOL_SOCKET
 - nivel de protocolo TCP socket.IPPROTO_TCP
 - nivel de protocolo IP socket.IPPROTO_IP
- Consulta de opciones asociadas a un socket
 - Socket.getsockopt()
- Modificación de opciones asociados a un socket
 - socket.setsockopt()
 - Ejemplos (nivel socket.SOL_SOCKET)
 - Para reutilizar direcciones: socket.SO_REUSEADDR

Gestión de direcciones

- Una dirección física por que se especifica
 - no es transparente
 - puede cambiar
 - no la gestionas tu
 - te la dan
- Gestión de direcciones en sockets
 - Direccionamiento en sockets
 - Orden de bytes
 - Direccionamiento físico: conversiones de formato
 - Decimal-punto a binario
 - Binario a decimal - punto
 - Direccionamiento lógico: la librería del resolver
 - Conversiones de nombres de host en la librería
 - La estructura struct hostent
 - Manejo de puertos y servicios en la librería
 - la estructura struct servent
 - otras funciones
 - nombre de host local
 - dirección local y remota de un socket

Direccionamiento en sockets

- Los usuarios manejan direcciones como textos
 - Formato decimal-punto 172.24.9.200
 - Formato dominio-punto servidor.ceu.es
 - dominio si cambia la ip sigue siendo el mismo
 - dns nos ayuda a encontrarlo
 - decimal-punto y dominio-punto son strings
 - la torre de protocolo esta esperando bytes
 - 32 bits 4 bytes
- Las llamadas de sockets manejan direcciones en binario
 - en orden de bytes de red
- Necesidad de conversión entre ambos formatos:

- Direccionamiento físico
 - Decimal-punto a binario: `inet_pton()`
 - binario a decimal-punto: `inet_ntop()`
- Direccionamiento lógico (librería del resikver)
 - Dominio-punto a binario
 - `gethostbyname()`
 - binario a dominio-punto
 - `gethostbyaddr`

Orden de los bytes

- En TCP/IP los números se usan siempre en orden de red (network byte order), que es big-endian
 - un host puede usar su propio orden (little-endian)
- Funciones que transforman el orden de los bits

```
import socket
valor_red16bits = socket.htons(valor_host16bits)
valor_red32bits = socket htonl(valor_host32bits)

valor_host16bits = socket ntohs(valor_red16bits)
valor_host32bits = socket ntohl(valor_red32bits)
```

» En todas ellas:

- La letra `h` significa “host”
- La letra `n` significa “network”
- La letra `s` significa “short” (16 bits)
- La letra `l` significa “long” (32 bits)

network to host short

Conversiones decimal-punto a/de binario

- Decimal-punto a binario: `inet_pton()`
 - funciona tanto para IPv4 como para IPv6

```
import socket
dirbinaria = socket.inet_pton(familia, direccion)
```

» Binario a decimal-punto: `inet_ntop()`

```
import socket
direccion = socket.inet_ntop(familia, dirbinaria)
```

- En caso de error lanzan una excepción `socket.error` y fijan `errno` con el valor adecuado:
 - `EAFNOSUPPORT` – dominio no soportado
 - `ENOSPC` – el parámetro de dirección no tiene el formato adecuado

Familia de dirección. Sólo:

- `AF_INET`
- `AF_INET6`

Dirección a convertir.

Descripción de la librería del resolver

- La librería del resolver realiza las conversiones anteriores mediante varios sistemas:
 - fichero /etc/hosts
 - DNS
 - bases de datos
 - base de datos jerárquica
 - árbol
 - .co.uk
 - www.test.co.uk
 - ac
 - academic
 - co
 - comercial
 - es
 - rediris
 - ceu
 - julian romea
 - monteprincipe
 - eps
 - monteprincipe
 - git
 - nombre de la máquina
 - alias
 - si la cambias funciona
 - www.marca.com
 - dentro de com
 - marca
 - www.
 - NIS/ NIS+
 - servicio de estilo DNS
 - sirve para red de área local
 - ...
- Las llamadas de la librería del resolver operan tanto con formato dominio-punto como decimal-punto
- El resolver también maneja puertos y servicios

Nombres de host: Librería del resolver

- Dominio-punto a binario
 - `gethostbyname()` y `gethostbyname_ex()`

```
import socket
direccion_ipv4 = socket.gethostbyname(nombre)
print nombre, lista_alias, lista_dinamicos
```

```
nombre, lista_alias, lista_diripv4 =  
socket.gethostbyname_ex(nombre)
```

- En caso de error lanzan una excepción `socket.error` y fijan `h_errno` con el valor adecuado:
 - Valores:
`HOST_NOT_FOUND`,
`NO_DATA`, `TRY AGAIN`,
`NO RECOVERY`

Lista de direcciones IP del host
Generalmente una sola

Lista de nombres alternativos (alias)
Generalmente vacía

Nombre canónico del host

- Devuelve una lista
 - una máquina puede tener más de una interfaz
- Binario a dominio-punto: `gethostbyaddr()`

```
import socket  
nombre, lista_alias, lista_diripv4 =  
socket.gethostbyaddr(direccion_ip)
```

- En caso de error lanza una excepción `socket.error` y fija `h_errno` con el valor adecuado:
 - Valores:
`HOST_NOT_FOUND`,
`NO_DATA`, `TRY AGAIN`,
`NO RECOVERY`

Lista de direcciones IP del host
Generalmente una sola

Lista de nombres alternativos (alias)
Generalmente vacía

Nombre canónico del host

- como se llama una máquina

Arma definitiva: Librería del resolver

- Información de direcciones: `getaddrinfo()`

```
import socket  
[(familia, tipo, protocolo, nombre, direccion)] =  
socket.getaddrinfo(nombre, puerto,  
[familia, tipo, protocolo, flags])
```

- En caso de error lanza una excepción `socket.error` y fija `h_errno` con el valor adecuado:
 - Valores:
`HOST_NOT_FOUND`,
`NO_DATA`, `TRY AGAIN`,
`NO RECOVERY`

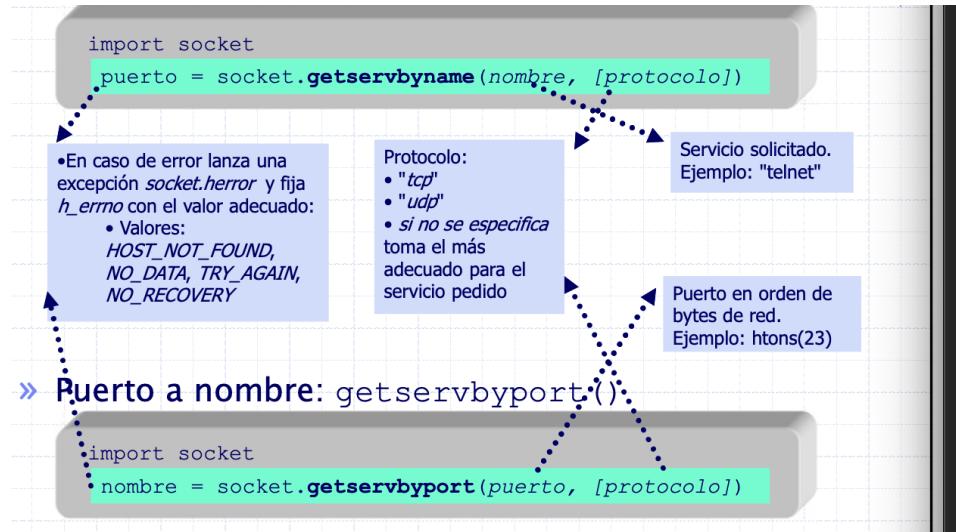
Puerto. Si se deja vacío sólo se usará el nombre

Nombre o dirección IP. Si se deja vacío sólo se usará el puerto

Ver constantes `AI_*`

- familia, tipo, protocolo son los parámetros que necesita la creación de un socket (ver `socket.socket()` para explicaciones)
- nombre es el nombre canónico del host si se ha especificado AI_CANONNAME en los flags
- dirección es (dir, puerto) para AF_INET o (dir, puerto, flujo, ámbito) para AF_INET6

Nombre a puerto: `getservbyname()`



Nombre de host local

» Nombre del host local: `gethostname()`

- Es una llamada local; no usa la librería del *resolver*.

```

import socket
nombre = socket.gethostname()
    
```

Devuelve el nombre de la máquina tal y como está configurado en el sistema operativo

» El nombre de un host bien configurado suele coincidir con el nombre canónico del DNS.

```

import socket
nombre_canonico = socket.getfqdn(nombre="")
    
```

Devuelve el nombre canónico asociado a `nombre`

- `gethostname()`
 - nombre de tu máquina
- `getfqdn`
 - fully qualifies domain name
 - con todos los puntos
 - mipc.eps.ceu.es.

ERRNO

- `error number`

error manager

- poder ver porque ha fallado
- cuando falla devuelvan valores negativos
- perror
 - print error

