



Sockets

Programación en Red / Entornos Distribuidos

Curso 2024/2025

Universidad San Pablo-CEU

Escuela Politécnica Superior


Campus de Montepríncipe



porque son interfaz de llamadas al sistema

Enchufe a nivel de transporte, que llevan BITS ENTRE PROCESOS

TCP 1521 es el listener de Oracle



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. *metieron la pila de protocolos*
- El diseño de los sockets es independiente de:
 - > el protocolo de comunicación (soportan unos 25 protocolos),
 - > el lenguaje de programación, y
 - > 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 se refiere al puerto TCP 1521 en la dirección IP 172.25.4.4
 - > La comunicación se produce **siempre entre dos sockets**.

18-ene.-25

Programación en Red / Entornos Distribuidos

página 2

Los sockets son la API del nivel de transporte de la pila TCP/IP

Aparecio la cola TP/IP, yo no solo eran ficheros y sistems de e/s

Se hizo independiente de TCP/IP. Otros protocolos como bluetooth tmb se pueden

Si hay un transporte entre dos procesos cada uno necesita un socket.



Sockets BSD

» Sujetos a proceso de estandarización dentro de POSIX (POSIX 1003.1g). Los sockets de hoy en día son POSIX, pero son practicamente iguales

» Actualmente:

- Disponibles en casi todos los sistemas Unix. mas bien todos
- En prácticamente todos los sistemas operativos:
 - › WinSock: API de sockets de Windows.
- En Java, Python, etc. aparecen como clase nativa.

» Competidores: estan muy extinguidos

- Transport Data Interface (TDI)
- X/Open Transport Interface (XTI)

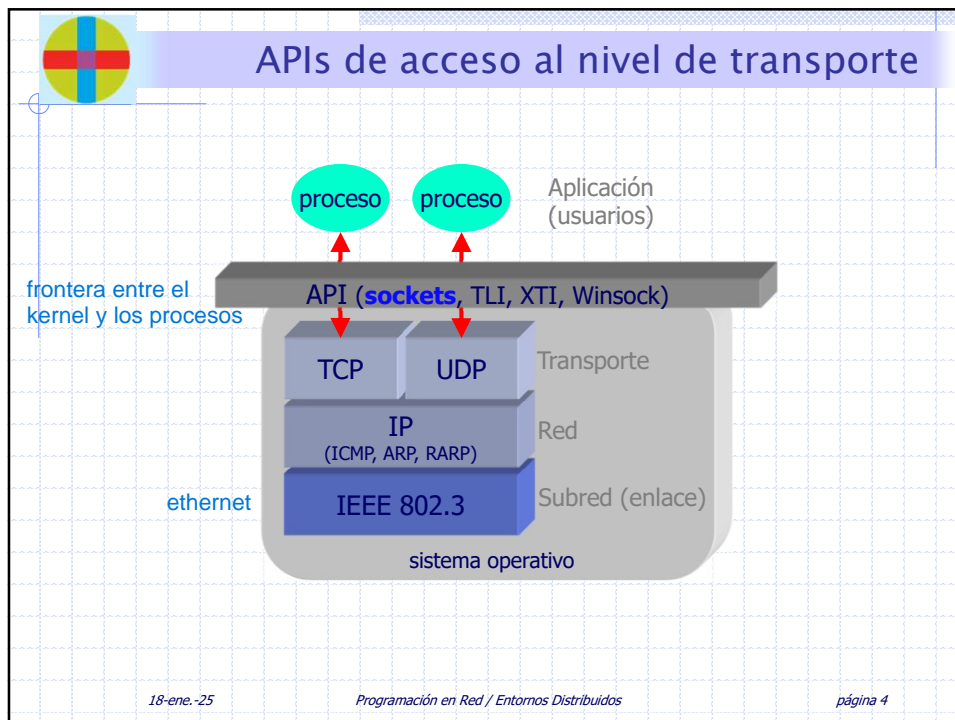
» El 99.9999999% de las comunicaciones en Internet se realizan mediante sockets.

Berkley Standard Distribution

En python se puede usar las llamadas directas o las clase socket

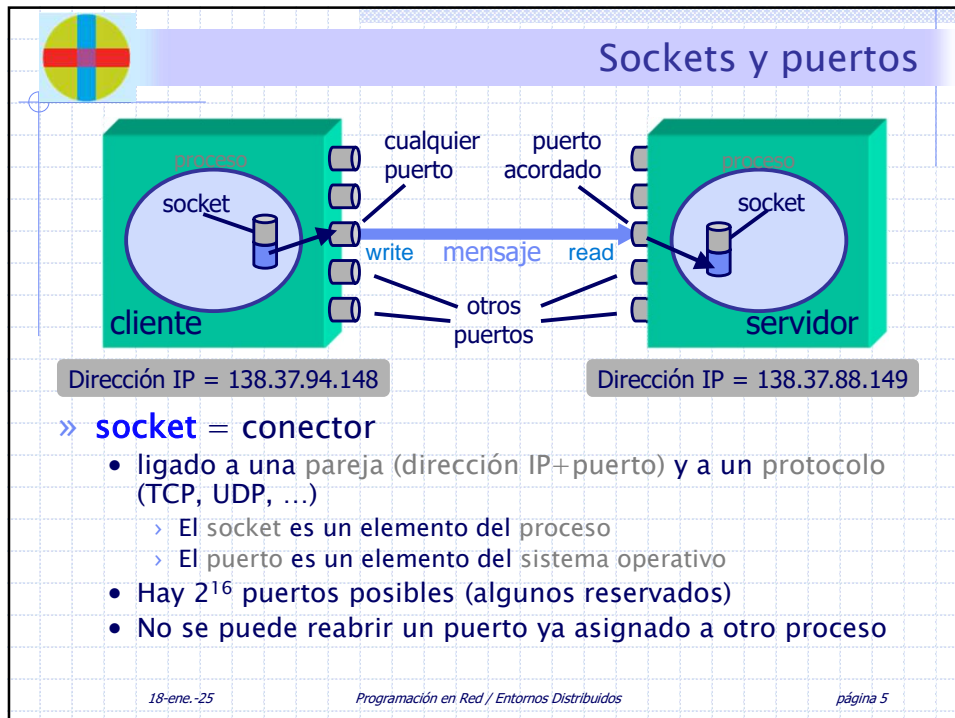
18-ene.-25 Programación en Red / Entornos Distribuidos página 3

3



4

El socket esta dentro del proceso, cada uno tiene el suyo. Se vincula los socket al puerto con una llamada al sistema (bind)



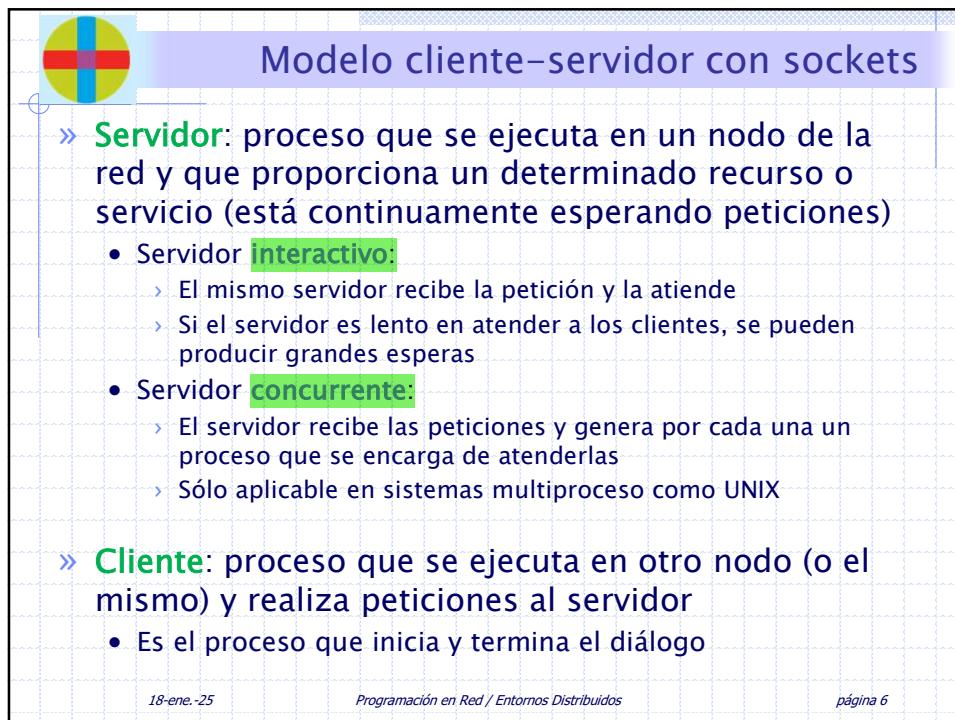
5

Tiene que cubrir la parte de sesion presentacion y aplicacion

Para solucionar Sesión: se hace un contrato definiendo quien es el cliente y servidor (se decide dependiendo de la conexión, quien la inicia es el cliente) Cliente es el activo y servidor el pasivo.

Espera preguntas, si no se le pregunta nada no dice nada, se duerme.

Siempre es el cliente el que manda peticiones y espera respuestas, el servidor espera preguntas y envía respuestas. Pero no se apagan nunca. Están esperando a que se les hagan preguntas.



El servidor atiende la petición personalmente.


El concurrente hace un hijo para atender a un cliente, después muere. Un proceso por cada cliente.

Depende de el uso para saber cual es mejor, para la hora interactivo va a ser mas rapido.

Pero si son cosas mas largas, el servidor no puede atender a otro cliente hasta que ha acabado con el otro cliente.

6


Lo que tarda es en crear el hijo, no en atender al cliente



Implementación socket cliente-servidor

» **Servidor:**

1. Creación y enlace al puerto de servicio de un socket.
2. Desconexión del servidor de su terminal de lanzamiento o ejecución.
3. Bucle infinito en el cual el servidor:



a. Espera una petición.

b. La trata.

c. Calcula y formatea la respuesta.

d. Envía la respuesta.

desde el punto de vista del servidor, no para nunca

La mayor parte del tiempo los servidores estan en la a. durmiendo

» **Cliente:**

1. Creación del socket local.
2. Preparación de la dirección del servidor.
3. Envío del mensaje.
4. Espera del resultado.
5. Explotación del resultado.

El servidor no se puede ir, se queda en el bucle.

Se puede repetir los pasos 3-4-5 pero cuando quiera se puede ir

La desconexion mata a todos los procesos hijos, a no ser que se desacople del proceso padre.

18-ene.-25


Programación en Red / Entornos Distribuidos

página 7

7

Comutacion de circuitos

Concurrente



Escenario de uso de sockets *streams*

TCP

Proceso cliente

socket()

connect()

error

send()/write()

recv()/read()

close()

Proceso servidor

socket() equivalente a open()

bind() abre la tienda

listen()

accept()

error / señal de fin

bloqueo

fork()

padre

hijo

ejecución en paralelo

recv()/read()

procesado

send()/write()

recv()/read()

close()

Abrir conexión

Petición

Respuesta

EOF

TCP es mas complicado de montar, pero mucho mas facil y fiable de usar.

concurrente porque crea hijos con el fork

18-ene.-25

Programación en Red / Entornos Distribuidos

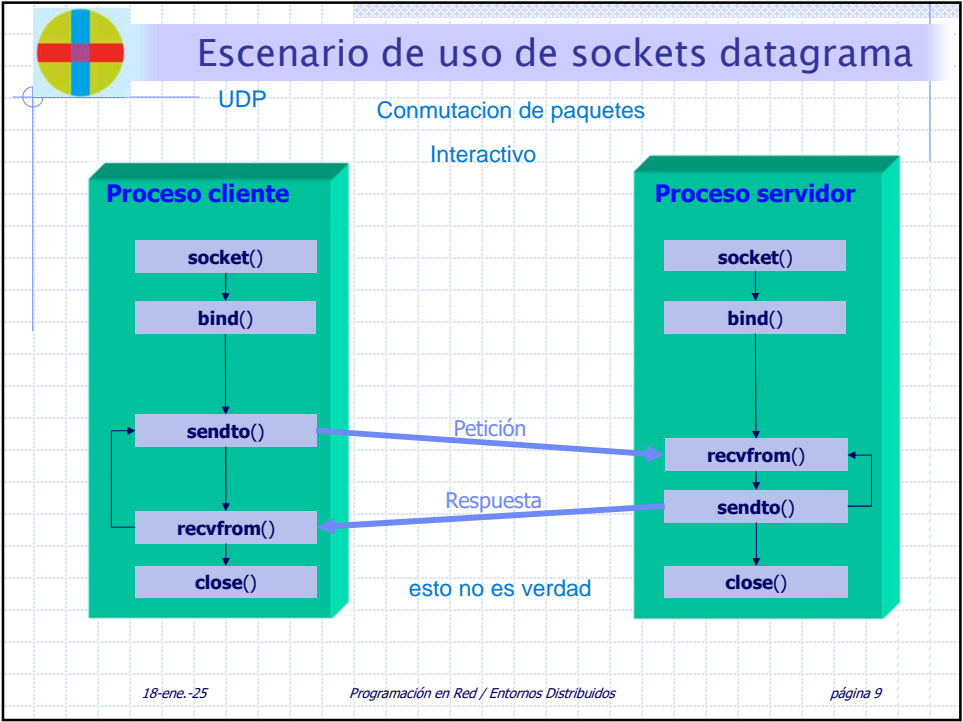
página 8

8

prac 4-6

Con datagramas es mas difícil que con streams porque UDP no tiene garantías. Muchas veces no hay respuesta, o llegan dos, o descolocadas, ...

Para programas muy sencillas donde hay basicamente solo una petición. Porque no se va a montar todo el fregado de TCP. Y porque solo siendo una petición, los llios de UDP son mas asequibles.



Aqui no hay realmente read() y write()

Conceptos básicos sobre sockets

1. Dominios de comunicación.
2. Tipos (estilos) de sockets.
3. Direcciones de sockets.
4. API de sockets:
 - Creación de un socket.
 - Asignación de direcciones.
 - Solicitud de conexión.
 - Preparar para aceptar conexiones.
 - Aceptar una conexión.
 - Transferencia de datos.
 - Cierre de la conexión.
 - Otras llamadas de interés.
5. Gestión de direcciones en sockets.

1.- Creación del socket

2.- Asignación de dirección

3.- Aceptación de conexión

18-ene.-25 Programación en Red / Entornos Distribuidos página 10

1. 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.
- » Los **servicios de sockets son independientes del dominio**.
 - Sólo se pueden comunicar sockets del mismo dominio.
- » Algunos **ejemplos** (presentes en el Unix 4.3 BSD):
 - Dominio `PF_UNIX` (ó `PF_LOCAL`): dentro de una máquina.
 - Dominio `PF_INET`: comunicación usando protocolos TCP/IP (válido para Internet, usado universalmente).
 - Dominio `PF_NS`: XEROX Network Service (NS). En desuso.
 - Dominio `PF_APPLETALK`: protocolo AppleTalk de Apple.
 - » Otros: `PF_802`, `PF_CCITT`, `PF_CHAOS`, `PF_DATAKIT`, `PF_DECNET`, `PF_DLI`, `PF_ECMA`, `PF_GOSIP`, `PF_HYLINK`, `PF_IMPLINK`, `PF_LAT`, `PF_MAX`, `PF_NIT`, `PF_OSI`, `PF_OSINET`, `PF_PUP`, `PF_SNA`, `PF_X25`, `PF_UNSPEC`

La llamada `socket()` crea un socket asociado a un dominio. Un socket del dominio X solo puede hablar con otros sockets del dominio X. Incompatible con los de otros dominios.

Se dice que dos protocolos pertenecen a la misma familia si usan el mismo formato de direcciones.

TCP y UDP usan IP+puerto, son de la misma familia.

Sin especificar

18-ene.-25

Programación en Red / Entornos Distribuidos

página 11

11

2. 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** TCP Stream es que es un flujo continuo.
 - Sockets tipo **datagrama** UDP Paquetes, como llegan depende del repartidor
- » Otros tipos de sockets:
 - Sockets tipo crudo (*raw*)
 - Otros tipos:
 - » *Sequenced packet sockets*
 - » *Reliably delivered message sockets*

18-ene.-25

Programación en Red / Entornos Distribuidos

página 12

12

ProtocolFamily_UNIX solo funciona en local. (pr 4). Formato de direcciones de las FIFO (archivos)

La familia de internet. Para IPv4. Un socket de IPv4 no habala con IPv6.

Para que do sockets puedan hablar tambien tienen que tener mismo conjunto de propiedades.



Sockets tipo *stream*

» **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; da una conexión *full-duplex*.
 - › 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. *Esto es remoto*
- En el dominio PF_UNIX son como una FIFO *full-duplex*. *Esto no es remoto, es local*

es full-duplex

Solo escribo por un lado de la autovia y leo por el otro lado de la autovia

Fiable es que no se pierde ni duplica

Lo que garantiza el orden es que sea secuencial.


Le pueden llegar dos mensajes de golpe

Son las propiedades del protocolo TCP.

Propiedades de las FIFO full-duplex

18-ene.-25 Programación en Red / Entornos Distribuidos página 13

13



Sockets tipo datagrama

» **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:
 - › 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.

Si manda dos paquetes (write()) tiene que hacer dos recogidas (read())

Cada paquete se busca su propio camino, no reserva camino y recursos como stream/TCP

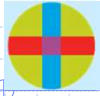
18-ene.-25 Programación en Red / Entornos Distribuidos página 14

14

Como WireShark que lee los paquetes de toda la red, aunque no sean los tuyos



Poner la interfaz de eth en modo promiscuo, sube todas las tramas aunque no sean para ti.



Otros tipos de sockets

» Raw (SOCK_RAW):

- Permiten el acceso a los protocolos de niveles más bajos (IP, la propia Ethernet, etc.).
- Requieren ser **superusuario (root)** para su utilización.
- Pueden usarse para implementar nuevos protocolos de transporte (RDP).

Esto solo se puede hacer por superusuarios porque permite acceder cualquier nivel de la torre de protocolos, lo que es muy peligroso. No se está produciendo encapsulación o desencapsulación

Se pueden tocar parámetros de protocolos de TCP/IP

» Otros tipos de sockets:

- **Sequenced packet sockets** (SOCK_SEQPACKET):
 - › Sólo 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

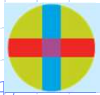
18-ene.-25

Programación en Red / Entornos Distribuidos

página 15

15

Si eres un proceso con un socket, tienes que tener asignada solo una dirección



3. Direcciones de sockets

» Un **socket** debe tener asignada una **dirección única**.

- Las direcciones son **dependientes del dominio**.
- Usos:
 - › Asignar una dirección local a un socket (**bind()**).
 - › Especificar una dirección remota (**connect()**, **sendto()**).

En un socket local no le puedes poner una IP+puerto ni a un socket internet le puedes poner una dirección de fichero.

» **Formatos** de direcciones: cada **dominio** usa una estructura específica:

- Direcciones en **PF_INET**: **dirección IP + puerto**
 - › formato: `socket.AF_INET` **AddressFormat**
 - › ejemplo: `('172.25.3.33', 4567)`
- Direcciones en **PF_UNIX**: **path de un fichero**
 - › formato: `socket.AF_UNIX`
 - › ejemplo: `"/tmp/socket"`

18-ene.-25

Programación en Red / Entornos Distribuidos


página 16

16

Loopback es una subred, nivel de enlace o físico. Se usa cuando no tiene una tarjeta de ethernet para que transmita. Loopback cuando le llega un paquete le da la vuelta al origen y destino y lo devuelve.

Garantiza que funcione la torre sin tarjeta física. Como una tarjeta virtual.

Así se pueden probar aplicaciones en local.



Direcciones de sockets en PF_INET

» Una **dirección de Internet** viene determinada por:

- Dirección IP del *host*: 32 / 128 bits. IPv4 IPv6
- Puerto de servicio: 16 bits. 2 oct
- 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*), que proviene del IBM System/370.
- Constantes:
 - > INADDR_LOOPBACK ('127.0.0.1') – interfaz loopback
 - > INADDR_ANY ('', '0.0.0.0') – cualquier dirección entrante
 - > INADDR_BROADCAST ('<broadcast>') – envío de broadcast
 - > INADDR_NONE ('255.255.255.255') – indica un error

18-ene.-25

Programación en Red / Entornos Distribuidos

página 17

Tienen que tener mismo protocolo de transporte!

Se vincula el protocolo desde el principio

Hay que ponerlo siempre en big endian, la conversión si necesaria ya se hará

No es una IP válida, sirve para decir que me vale cualquiera.

Envío a todos los de la red. Pero no pasa los routers, no lo encaminan.

17 Broadcast default si no sabes donde estas para que el server dhcp te envíe la configuración

Diferentes formatos de direcciones son de diferentes tamaños

Se pasa un puntero y size. (como un buffer).




Gráfico de estructuras de direcciones

» Comparación visual de las estructuras (en C) de direcciones de las diferentes familias de *sockets*:

struct sockaddr

sa_family

sa_data

14 bytes

"clase abstr"

struct sockaddr_in

AF_INET

sin_port 2

sin_addr 4

sin_zero 0s

14 bytes

struct sockaddr_un

AF_UNIX

sun_data

108 bytes

18-ene.-25

Programación en Red / Entornos Distribuidos

página 18

TypeTag

A día de hoy mucho más

18

4. Mapa de la API de sockets

4. API de sockets:

1. Creación
 - `socket()`: crear un socket
 - `bind()`: asignar una dirección a un socket
2. Preparación de la conexión
 - `listen()`: esperar conexiones en modo pasivo
 - `connect()`: iniciar una conexión a otro socket
 - `accept()`: aceptar una conexión
3. Transferencia de datos
 - `write()/send()/sendto()`: escribir datos en un socket
 - `read()/recv()/recvfrom()`: leer datos de un socket
4. Despedida y cierre
 - `close()/shutdown()`: cerrar un socket eliminando la conexión
5. Otras llamadas de interés

18-ene.-25 Programación en Red / Entornos Distribuidos página 19

19

Creación de un socket

» La llamada `socket()` crea un socket:

- El socket creado **no** tiene dirección asignada.

```
import socket    es un createSocket
```

```
s = socket.socket(dominio, tipo, protocolo)
```

- Devuelve un objeto socket (similar a los devueltos por `open()`) si no hay error.
- Lanza una excepción `socket.error` si lo hay.
- En caso de error se fija `errno` con el valor adecuado:
 - `EPROTONOSUPPORT` - el tipo o protocolo pedido no se soporta en el dominio especificado
 - `EMFILE` - el proceso tiene demasiados descriptors abiertos
 - `ENFILE` - el sistema tiene demasiados descriptors abiertos
 - `EACCES` - el proceso no tiene privilegios suficientes para la creación del socket
 - `ENOBUFS` - no hay espacio en los buffers internos del sistema

Depende del dominio y del tipo de socket:

- Elige el más adecuado (valor por defecto)
- Ver fichero `/etc/protocols`

Tipo o estilo del socket:

- `SOCK_STREAM` (por defecto)
- `SOCK_DGRAM`
- `SOCK_RAW`

Dominio o *namespace* del socket:

- `PF_LOCAL/PF_UNIX/PF_FILE`
- `PF_INET` (por defecto)
- `PF_INET6`

18-ene.-25 Programación en Red / Entornos Distribuidos página 20

Un socket es un descriptor de fichero (en lenguajes de alto nivel es un objeto)

Por si dentro del dominio y tipo quieres elegir propiedades. Casi siempre se pone 0 (por defecto)


La llamada `socket` puede fallar si no tienes implementado ese dominio o tipo de comm o porque no tenga espacio en buffers,...

La llamada `socket` asocia un socket a mi programa.

20

Todos los procesos de unix tiene abierto los descriptores de fichero 0,1,2 porque su padre los tiene abiertos.

Desvincular la terminal del servidor tambien significa eliminar esos descriptores, entre otras cosas



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):

Tabla de descriptores

0

1

2

3

4

...

→ Estructura para /dev/tty

→ Estructura para fichero "hola.txt"

→ Socket:

- Dominio de comunicación: PF_INET
- Tipo de servicio: SOCK_STREAM
- Dirección IP local: 172.25.9.69
- Puerto local: 3456
- Dirección IP remota: 123.2.4.6
- Puerto remoto: 2578

18-ene.-25

Programación en Red / Entornos Distribuidos


página 21

Estructura de los parametros del socket.

La llamda socket rellena los dos primeros params, dominio y servicio

21

Bind garantiza que una dir local queda unida a un solo proceso, los sockets garantiza acceso exclusivo, no como los pipes.



Asignación de direcciones

» Asignación de una dirección a un socket ya creado:

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

```
import socket
```

```
s.bind(direccion)
```

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

- EADDRNOTAVAIL – s no es un descriptor de fichero válido
- ENOTSOCK – s no es un socket
- EADDRINUSE – 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.

18-ene.-25

Programación en Red / Entornos Distribuidos

página 22

El cliente pone cualquier direccion y cualquier puerto (0). Efímero

Aqui es donde se le pasa una direccion, que tiene que coincidir con la familia del dominio.

La direccion que se elige suele ser la any (0.0.0.0)

Si se hace un bind al loopback, no atenderias ninguna petición.

Habitualmente solo tiene que saberse su puerto


El DNS al final lo que hace es coger el puerto.

22

El sistema no te deja usar un puerto de los 'oficiales' para que un usuario cualquiera no pueda 'secuestrar' un proceso oficial.

Si intentas usar un puerto ya en uso, te da un error de puerto ya en uso.

Se podría poner dos procesos a un puerto, pero pasaría que se pelarían por los datos.



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 máquina.
- **Puertos:** el rango de puertos es 0..65535.
 - » Si se le indica el puerto 0, el sistema elige uno adecuado.
 - » Si el puerto solicitado está ya asignado la llamada `bind()` devuelve un valor negativo.
 - » El espacio de puertos para *streams* (TCP) y datagramas (UDP) es independiente.

Tipo de puertos	Rangos	Descripción
Reservados 0..1023 ↑ (IPPORT_RESERVED)	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
No reservados 1024..65535	1024..4999	usados por procesos de usuario y del sistema
	> 5000	usados sólo por procesos de usuario

18-ene.-25

Programación en Red / Entornos Distribuidos

página 23

23

Connect, para conectar el cliente al servidor.

Se pone la dirección remota (del server), por lo que sale a la red.

Justo por esto, falla mucho más.

Si hay un firewall, te tira pa abajo el paquete y no suelven devolver error, así que te da un timeout


Si te equivocas de puerto, te da conexión refused

Si te equivocas de IP te da un host unreachable.

Los timeout son los más comunes

Como el cliente siempre hace el mismo bind (0.0.0.0:0) pues se lo hace automáticamente/explicito.

El servidor si que tiene que poner el puerto, así que no lo puede hacer explícitamente



Solicitud de conexión

» Realizada en el cliente con la llamada `connect()`:

- 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)

```
import socket  
s = socket.connect(direccion)
```

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

- `EBADF`, `ENOTSOCK`, `EADDRINUSE`, `EADDRNOTAVAIL` (dir. remota no disponible), `EINVAL` (tipo de socket no soportado), `EISCONN` (ya conectado), `ETIMEDOUT` (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.


18-ene.-25

Programación en Red / Entornos Distribuidos

página 24

24

En los servidores concurrentes el tiempo de servicio es el tiempo que se tarda en crear un hijo.



Preparar para aceptar conexiones

» 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()`)

```
import socket  
s.listen(backlog)
```

- En caso de error lanza una excepción `socket.error` y fija `errno` con el valor adecuado:
 - `EBADF` – el parámetro `s` no es un descriptor de fichero válido
 - `ENOTSOCK` – `s` no es un socket
 - `EOPNOSUPP` – el socket `s` no soporta esta operación

- Longitud de la cola de peticiones pendientes (`backlog`).
 - Si se excede, las conexiones fallarán con `ECONNREFUSED` hasta que el servidor elimine alguna conexión pendiente mediante una llamada a `accept()`.
 - Algunos manuales recomiendan 5 (más para servicios interactivos).

Socket ya creado y con dirección asignada.

18-ene.-25

Programación en Red / Entornos Distribuidos

página 25

Deja el socket preparado para aceptar conexiones.


Si el servidor es interactivo, solo puede atender ese cliente, y puede haber una cola, con un size predeterminado de clientes. Eso es el backlog

Mejor no tener una cola grande, porque es casi mejor que se rechaze la conn que dejar esperando a muchos clientes.

Acepta la conexion de un cliente.

Aqui es donde se queda durmiendo el servidor hasta que pase un cliente.

Si un serv concurrente tiene 15 conexiones con clientes, tendra 16 sockets activos



Aceptar una conexión

» Realizada con `accept()` en un servidor de sockets *stream* tras preparar la conexión (`listen()`)

```
import socket  
ns,direccion = s.accept()
```

- Devuelve una pareja de valores:
 - El nuevo socket que se crea al aceptar la conexión (`ns`)
 - La dirección remota de dicha conexión (`direccion`)

Socket ya creado y con dirección asignada.

- 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
 - `EOPNOSUPP` – el socket `s` no soporta esta operación
 - `EWOULDBLOCK` – `s` es no bloqueante y no hay conexiones pendientes

18-ene.-25

Programación en Red / Entornos Distribuidos

página 26

En Servidores concurrentes:


El socket original solo se usa para crear las conexiones

ns => nuevo socket

Se crea un nuevo socket para atender el cliente, y así liberar el otro socket para que puedas recibir nuevas peticiones.

En servidores interactivos, no hace falta porque solo atiende a uno a la vez.

26




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 `accept()` 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**.

18-ene.-25 Programación en Red / Entornos Distribuidos página 27

27



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 será 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 (módulo `os`)
 - El descriptor a usar aquí será el descriptor del socket
 - › puede leer de cualquier descriptor, sea fichero o socket

18-ene.-25 Programación en Red / Entornos Distribuidos página 28

28

Envío y recepción en sockets *stream*

» 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()`)

» 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()`)

18-ene.-25 Programación en Red / Entornos Distribuidos página 29

29

Envío y recepción en sockets datagrama

» No se establece **conexión** (`connect()` / `accept()`)

- Para usar un socket para transferir basta con crear el socket y reservar la dirección (`bind()`)

» Envío:

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

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

» 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.

18-ene.-25 Programación en Red / Entornos Distribuidos página 30

30

Cierre de un socket

» Puede usarse la clásica llamada `close()`:

- Cierra ambos sentidos de la conexión en sockets *stream*.

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

» Otra opción es usar `shutdown()`:

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

- 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()`.

18-ene.-25
Programación en Red / Entornos Distribuidos
página 31

31

Sockets como ficheros

» Un socket puede tratarse como un fichero (con la API de *streams* del lenguaje) mediante el método `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).

18-ene.-25
Programación en Red / Entornos Distribuidos
página 32

32

Sockets en lugar de *pipes*

» Un par de sockets es una estructura similar a una tubería sin nombre (*pipe*); lo crea `socketpair()`.

- Crea una pareja de sockets sin nombre y conectados.
- **Diferencia:** ambos sockets son bidireccionales, no como los descriptors 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 descriptors 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`

18-ene.-25 Programación en Red / Entornos Distribuidos página 33

33

Direcciones local y remota de un socket

» Las funciones `getsockname()` y `getpeername()` obtienen información a partir del descriptor socket:

`getsockname()` obtiene la dirección asociada al socket local.

`getpeername()` obtiene la dirección del socket remoto conectado al socket local (si existe tal conexión).


```
import socket
direccion_local = s.getsockname()
direccion_remota = s.getpeername()
```

- Devuelven una dirección IPv4 ó IPv6
- En caso de error lanzan una excepción `socket.error` y fijan `errno` con el valor adecuado:
 - `EBADF` – s no es un descriptor de fichero válido
 - `ENOTSOCK` – s no es un socket
 - `ENOTCONN` – socket no conectado
 - `ENOBUFS` – no hay suficientes buffers de entrada/salida para satisfacer la petición

Socket ya creado.

18-ene.-25 Programación en Red / Entornos Distribuidos página 34

34



Comandos y llamadas de utilidad


- » Existen algunos **comandos** Unix que son de utilidad en la programación con sockets:
 - Ver **conexiones** de sockets abiertas en el sistema:

```
unix> netstat -a
```
 - Ver las **llamadas** al sistema que realiza un proceso:

```
unix> strace proceso
```
- » También existen algunas **llamadas** de utilidad en el módulo **os**:
 - Sincronizar un descriptor de fichero: `os.fsync(fd)`

18-ene.-25 Programación en Red / Entornos Distribuidos página 35

35




Configuración de opciones en sockets

- » Existen varios niveles de **opciones** dependiendo del protocolo afectado como parámetro:

opciones independientes del protocolo	<code>socket.SOL_SOCKET</code>
nivel de protocolo TCP	<code>socket.IPPROTO_TCP</code>
nivel de protocolo IP	<code>socket.IPPROTO_IP</code>
- » **Consulta** de opciones asociadas a un socket:
`socket.getsockopt()`
- » **Modificación** de opciones asociadas a un socket:
`socket.setsockopt()`
 - Ejemplos (nivel `socket.SOL_SOCKET`):
 - › Para reutilizar direcciones: `socket.SO_REUSEADDR`

18-ene.-25 Programación en Red / Entornos Distribuidos página 36

36



5. Gestión de direcciones

5. Gestión de direcciones en sockets.

1. Direccionamiento en sockets
2. Orden de bytes
3. Direccionamiento físico: conversiones de formato
 - > Decimal-punto a binario
 - > binario a decimal-punto
4. 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`
5. Otras funciones
 - > Nombre de host local
 - > Dirección local y remota de un socket

18-ene.-25 Programación en Red / Entornos Distribuidos página 37

37



Direccionamiento en sockets

- » Los usuarios manejan direcciones como **textos**:
 - Formato decimal-punto: 172.24.9.200
 - Formato dominio-punto: servidor.ceu.es 13 bytes (12+1)
- » 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 *resolver*):
 - > Dominio-punto a binario: `gethostbyname()`
 - > Binario a dominio-punto: `gethostbyaddr()`

18-ene.-25 Programación en Red / Entornos Distribuidos página 38

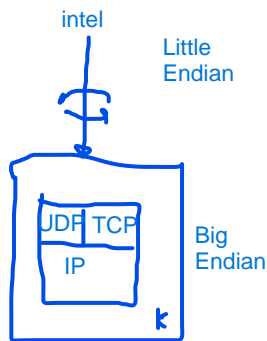
Manejamos direcciones como textos

Como traduzco esto a los 32 bits que necesito? DNS

Hay una librería de resolver

38

socket :127.0.0.1"



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)

Si meto el n de puerto desde little endian a tcp (big endian) hay que convertirlo. Si es maquina big endian desde el principio no tiene que hacer nada

Para imprimir las cosas que viene de la pila de protocolos hay que usar estas

Queremos que funcione en todo tipo de maquinas no solo big endian. Por eso hay que usar estas cosas en el código

18-ene.-25

Programación en Red / Entornos Distribuidos

página 39

39

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)
```

Coge una dir escrita en decimal punto y lo traduce a binaria.

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

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

Pero si se usa esto, no hay que hacer lo anterior porque ya lo hace

• 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.

18-ene.-25

Programación en Red / Entornos Distribuidos

página 40


40

La libreria resolver suele tener de manera predeterminada solo el hosts y el DNS

Todo lo anterir no hay que hacerlo si se usa la libreria resolver. Porque si le pones una IP directamente, algo que no tiene que resolver, hace todo lo del htos,htol,...

aaaa IPv6

a => dar la IP de un nombre
ptr => dar el nombre de una IP



Descripción de la librería del *resolver*

» La **librería del *resolver*** realiza las conversiones anteriores mediante varios sistemas:

- Fichero `/etc/hosts` primero mira en /etc/hosts, si no esta la respuesta pregunta al DNS
- DNS
- NIS/NIS+
- ...

» Las llamadas de la librería del *resolver* operan tanto con formato dominio-punto como decimal-punto.

» El resolver también maneja **puertos y servicios**.


18-ene.-25

Programación en Red / Entornos Distribuidos

página 41

Si me quiero conectar a una maquina exterior pero no usar la IP, por si le cambia. Para esto el DNS

La libreria del resolver usa varios sistemas para resolver. Mira a estos por prioridad



Nombres de host: librería del *resolver*

» **Dominio-punto a binario:** `gethostbyname()` y `gethostbyname_ex()` Como hacer preguntas a las bases de datos:

```
import socket
direccion_ipv4 = socket.gethostbyname(nombre)
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

18-ene.-25

Programación en Red / Entornos Distribuidos


página 42

le das el nombre y te da la IP

version extendida con mucha mas informacion como la familia (IPv?)

Por debajo es un select de nombre, list_...

Le pones la dir IP y devuelve el nombre



Direcciones de host: librería del *resolver*

» 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

18-ene.-25

Programación en Red / Entornos Distribuidos

página 43


43

44

La autentica que hace las queries es esta. Porque la 'tabla' son 5 campos.

Salen todas las familias, por lo que muchos registros

getaddrinfo => gai
los errores relacionados con esto son los gai_errors



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_*`

Si se especifican reducen la lista de resultados

• `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

• `direccion` es (dir, puerto) para `AF_INET` o (dir, puerto, flujo, ambito) para `AF_INET6`


18-ene.-25

Programación en Red / Entornos Distribuidos

página 44

En Windows se usa netbios que es un broadcast para localizar cuando no esta en 'hosts'

El firewall suele hacer un drop en vez de un reject, porque si se hace un reject sabes que si existe la direccion o si se ha rechazado, no da pistas. Con un drop da un timeout como si no existiera.



Servicios y puertos: librería del *resolver*

» Nombre a puerto: `getservbyname()`

```
import socket
puerto = socket.getservbyname(nombre, [protocolo])
```

• 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`

Protocolo:

- `"tcp"`
- `"udp"`
- si no se especifica toma el más adecuado para el servicio pedido

Servicio solicitado. Ejemplo: `"telnet"`

Puerto en orden de bytes de red. Ejemplo: `htons(23)`

» Puerto a nombre: `getservbyport()`

```
import socket
nombre = socket.getservbyport(puerto, [protocolo])
```

18-ene.-25

Programación en Red / Entornos Distribuidos

página 45

La otra parte de la direccion que podemos a resolver son los puertos, no se usa mucho porque la gente se aprende los puertos, pero se pueden sacar.


El resolver los llama service

El DNS tambien los sabe pero no los dice.

El fichero `/etc/service` es el que se configura para los puertos

Por esto se puede usar el nombre de un puerto en vez del numero

45



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

pc5

» 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*

pc5.git.eps.ce.es

18-ene.-25

Programación en Red / Entornos Distribuidos

página 46

Averiguar el nombre de mi maquina

El nombre que se le pone a la maquina es buena practica que coincida con el que se le da al DNS, pero no tiene porque.

Es una pregunta al sistemas operativo

46

En C hay una llamada global que la coloca en una variable global llamada `errno`, Pero es una var, así que cambia cada vez que hat un error nuevo.

Se puede usar el `perror()` para que lo imprima antes de que cambie la variable.

Posibles valores de <code>errno</code> (I)		
<code>errno</code>	<code>perror()</code>	Posible causa
EACCES	<i>permission denied</i>	El programa no tiene acceso a este socket.
EADDRINUSE	<i>socket address is already in use</i>	La dirección dada ya está siendo usada.
EADDRNOTAVAIL	<i>socket address not usable</i>	La dirección dada no está disponible en la máquina local.
EAFNOSUPPORT	<i>unsupported socket addressing family</i>	La familia de direcciones no está soportada o no es consistente con el tipo de socket dado.
EALREADY	<i>previous connection not yet completed</i>	El socket tiene activada la opción de entrada/salida no bloqueante, y aún está pendiente de completarse una llamada anterior.
EBADF	<i>file or socket not open or unsuitable</i>	El descriptor de fichero dado no se corresponde con un socket (o un fichero) abierto.
ECONNABORTED	<i>connection aborted by local network software</i>	El software local de comunicaciones ha abortado la conexión.
ECONNREFUSED	<i>destination host refused socket connection</i>	La máquina destino ha rechazado la conexión del socket.
ECONNRESET	<i>connection reset by peer</i>	El proceso remoto ha reiniciado la conexión.
EDESTADDRREQ	<i>socket operation requires destination address</i>	El socket necesita que se le suministre una dirección de destino.
EHOSTDOWN	<i>destination host is down</i>	El host de destino está apagado.
EHOSTUNREACH	<i>destination host is unreachable</i>	El host de destino es inalcanzable.

18-ene.-25

Programación en Red / Entornos Distribuidos

página 47

Si se escribe mal la dir, da un timeout, si no escucha ese puerto puede dar un refused

Posibles valores de <code>errno</code> (II)		
<code>errno</code>	<code>perror()</code>	Posible causa
EINPROGRESS	<i>socket connection in progress</i>	La conexión se ha iniciado, pero se devuelve control de forma que la llamada no bloquee al proceso que la ha realizado. La conexión estará completa cuando una llamada <code>select()</code> indique que el descriptor de fichero está listo para escritura. Este código no es un error.
EISCONN	<i>socket is already connected</i>	El proceso llamó a <code>connect()</code> en un socket ya conectado.
EMSGSIZE	<i>message too large for datagram socket</i>	El mensaje es demasiado largo para acomodarlo en un socket de tipo datagrama.
ENETDOWN	<i>local host's network down or inaccessible</i>	El programa no puede hablar con el software de red en la máquina local, o bien la red local está apagada.
ENETRESET	<i>remote host dropped network communications</i>	El host remoto no se está comunicando por la red en este momento.
ENETUNREACH	<i>destination network is unreachable</i>	El host local no puede encontrar una ruta a la red de destino.
ENOBUFS	<i>insufficient buffers in network software</i>	El sistema operativo no tiene suficiente memoria para realizar la operación pedida.
ENOPROTOOPT	<i>option not supported for protocol type</i>	La opción (o el nivel de la misma) pedida sobre el socket no es válida.
ENOTCONN	<i>socket is not connected</i>	El socket no está conectado.
ENOTSOCK	<i>file descriptor not associated with a socket</i>	El descriptor de fichero no corresponde a un socket (es un fichero, o bien está sin asignar).

18-ene.-25

Programación en Red / Entornos Distribuidos

página 48

Posibles valores de <code>errno</code> (III)		
errno	perror()	Posible causa
EOPNOTSUPP	<i>operation not supported on socket</i>	La operación pedida no está soportada en el tipo de socket dado.
EPFNOSUPPORT	<i>unsupported socket protocol family</i>	La familia de protocolos dada es desconocida o bien el software de TCP/IP no la soporta.
EPIPE	<i>broken pipe or socket connection</i>	El proceso remoto ha cerrado el socket (o la tubería) antes de que el proceso local haya terminado de utilizarlo.
EPROTONOSUPPORT	<i>unsupported socket protocol</i>	El protocolo dado es desconocido o bien el software de TCP/IP no lo soporta.
EPROTOTYPE	<i>protocol inconsistent with socket type</i>	Al llamar a <code>socket()</code> el protocolo no era ni 0 ni un valor consistente con el tipo de socket pedido.
ESHUTDOWN	<i>connection has been shut down</i>	La conexión se ha cerrado.
ESOCKTNOSUPPORT	<i>socket type not allowed</i>	El proceso local ha pedido un tipo de socket no soportado por el software de TCP/IP.
ESYS	<i>operating system interface failure</i>	El software de TCP/IP, o el sistema operativo en el que se apoya, ha devuelto una condición anormal de fallo. Convendría contactar con el fabricante.
ETIMEDOUT	<i>socket connection attempt timed out</i>	El host de destino no ha contestado a una petición de conexión.
EWOULDBLOCK	<i>socket operation would block</i>	El socket tiene activada la opción de entrada/salida no bloqueante, y esta llamada debería haberse bloqueado. Este código no es un error.


18-ene.-25

Programación en Red / Entornos Distribuidos

página 49

49

¿Preguntas?



18-ene.-25

Programación en Red / Entornos Distribuidos

página 50

50