

Tomás Fernández Pena  
José Carlos Cabaleiro Domínguez  
Oscar García Lorenzo

Grao en Robótica

Universidade de Santiago de Compostela

Redes e Comunicaci3ns, 3º Curso GrR

[citius.usc.es](http://citius.usc.es)



Centro de Investigación en  
Tecnología de Información

# Índice

- 1 Direccións IP
  - Direccións IPv4
  - Orden de host e orden de rede
  - Direccións IPv6
- 2 Portos
- 3 Sockets
- 4 Conversión do orden dos bytes
- 5 Servizo de Nomes de Dominio (DNS)
- 6 Outras funcións de conversión



# Índice

- 1 Direccións IP
  - Direccións IPv4
  - Orden de host e orden de rede
  - Direccións IPv6

- 2 Portos

- 3 Sockets

- 4 Conversión do orden dos bytes

- 5 Servizo de Nomes de Dominio (DNS)

- 6 Outras funcións de conversión



# Protocolo e direccións IP

- Protocolo de Internet (*Internet protocol*, IP): protocolo usado na Internet para encamiñar paquetes de datos entre **computadores** conectados a Internet
  - ▷ É o principal protocolo de comunicacións na Internet
- Dirección IP (*IP address*): etiqueta numérica asignada a cada dispositivo conectado a unha rede que usa o protocolo IP
  - ▷ Todo dispositivo (ordenador, móbil, consola de videoxogos, electrodoméstico intelixente, etc.) que se conecte a Internet ten que ter asignada unha dirección IP
  - ▷ Esa dirección pode ser fixa ou ir cambiando (pe, cada vez que se reinicia o dispositivo)
  - ▷ As direccións teñen que ser únicas, aunque hai direccións especiais que se poden repetir (máis sobre isto na clase de teoría)

# Versións das direccións IP

Dúas versións:

- IPv4 (clásica): a máis usada en equipos finais (ordenadores, móbiles, etc.)
  - ▷ a dirección é un número enteiro de 32 bits sen signo
- IPv6: reemplaza á anterior, vense introducindo en routers e dispositivos intermedios
  - ▷ a dirección é un número enteiro de 128 bits sen signo

Compatibilidade cara arriba:

- Toda dirección IPv4 ten unha IPv6 asociada (o contrario non é certo)

# Formato das direccións IPv4

Unha dirección IP pódese representar de dúas diferentes formas:

- Formato binario: representación interna do dispositivo

- ▷ Enteiro sen signo de 32 bits (4 bytes) (en C `in_addr_t` ou `uint32_t`<sup>1</sup>)

- ▷ Exemplo en hexadecimal:

```
in_addr_t ip = 0xC8806EC1;
```

- Formato textual: representación *lexible* para humanos

- ▷ Cadea de caracteres de lonxitude máxima `INET_ADDRSTRLEN`<sup>2</sup> (en C `char ip[INET_ADDRSTRLEN]`).

- ▷ As cadeas válidas como direccións teñen un formato fixo:

- 4 campos numéricos con valores entre 0 y 255 separados por punto

- ▷ Exemplo: `char ip[]="193.110.128.200";`

---

<sup>1</sup>Tipos definidos no ficheiro de cabeceira `inttypes.h`

<sup>2</sup>Macro definida en `netinet/in.h` e que vale 16

# Formato das direccións IPv4

Formato adicional:

- Representar a IP como un array de 4 enteiros sen signo de 8 bits (1 byte):

```
uint8_t ip1[4] = {193u, 110u, 128u, 200u};  
uint8_t ip2[4] = {0xC1, 0x6E, 0x80, 0xC8};
```

# Formato de las direcciones IPv4 en C

Moitas funcións C piden a IP formato binario encapsulado nunha estrutura `struct in_addr`<sup>3</sup>

- Contén un único campo `s_addr` de tipo `in_addr_t`

Exemplo:

```
struct in_addr ip;  
ip.s_addr=0xC8806EC1;
```

Macros predefinidas<sup>4</sup> para certas direccións IPv4 especiais:

- `INADDR_LOOPBACK`: lazo de volta ou *loopback* (127.0.0.1)
- `INADDR_ANY`: cualquier dirección válida (0.0.0.0)
- `INADDR_BROADCAST`: IP de difusión (255.255.255.255)
- `INADDR_NONE`: usado en funcións que devuelven unha IP para indicar un erro

---

<sup>3</sup>Declarada en `netinet/in.h`

<sup>4</sup>Definidas en `netinet/in.h`



# Orden de host e orden de rede

Dirección `in_addr_t ip = 0xC8806EC1`; en memoria

Dir. Mem.

0	B(0)
1	B(1)
2	B(2)
3	B(3)
	⋮
	C1 (193)
	6E (110)
	80 (128)
	C8 (200)
	⋮
N-1	

8 bits

Little-endian

Dir. Mem.

0	B(0)
1	B(1)
2	B(2)
3	B(3)
	⋮
	C8 (200)
	80 (128)
	6E (110)
	C1 (193)
	⋮
N-1	

8 bits

Big-endian

- En Internet sempre se usa big-endian (**orden de rede**)
- Os dispositivos poden usar little o big-endian (**orden de host**)

# Orden de host e orden de rede

Nun host little-endian, os seguintes valores representan a mesma IP:

- `in_addr_t ip1 = 0xC8806EC1;`
- `uint8_t ip2[4] = {0xC1, 0x6E, 0x80, 0xC8};`
- `uint8_t ip3[4] = {193u, 110u, 128u, 200u};`
- `char ip[]="193.110.128.200";`

Tense entón que:

$$\begin{aligned} "193.110.128.200" &= 193 \times 2^0 + 110 \times 2^8 + 128 \times 2^{16} + 200 \times 2^{24} = \\ &= 3363850161 = 0xC8806EC1 \end{aligned}$$

- En C hai funcións para realizar esa conversión (e a inversa)

# Direccións IPv6

## Problema coas direccións IPv4:

- Van da  $0 \times 0$  (0.0.0.0) á  $0 \times \text{FFFFFFFF}$  (255.255.255.255)
- Son un total de  $2^{32}$  valores posibles (máis de 4 mil millones)
- Non tódalas direccións son válidas (direccións reservadas)
- Xa se acabaron

## Solucións

- Permitir direccións repetidas (direccións privadas e NAT<sup>5</sup>)
- Migrar a IPv6 con direccións de 128 bits (16 bytes)
  - ▷  $2^{128} = 3,4 \times 10^{38}$  valores posibles

# Formato das direccións IPv6

## Formato binario: representación interna do dispositivo

- Enteiro sen signo de 128 bits (16 bytes)
- En C almacénase sempre encapsulado nunha estrutura de tipo `struct in6_addr`<sup>6</sup>
  - ▶ Permite acceder de diferentes formas, a través dunha `union`

```
/* Formato de la estructura declarada en netinet/in.h
struct in6_addr
{ union
{
    uint8_t s6_addr[16];
    uint16_t s6_addr16[8];
    uint32_t s6_addr32[4];
}; } */
struct in6_addr ipv6;
ipv6.s6_addr32[0] = 0x12345678;
printf("%x\n", ipv6.s6_addr[3]); // Imprime 12
printf("%x\n", ipv6.s6_addr16[0]); // Imprime 5678
```

# Formato das direccións IPv6

Formato textual: cadea de caracteres de lonxitude máxima

INET6\_ADDRSTRLEN<sup>7</sup>

- Representada mediante 8 grupos de ata 4 díxitos hexadecimais cada un, separados por :

```
char ip6[]="1080:0:0:0:8:800:200C:417A";
```

- As secuencias de zeros consecutivos pódense abreviar con ::,
  - ▷ Exemplo 1: a dirección anterior pódese abreviar como "1080::8:800:200C:417A"
  - ▷ Exemplo 2: a dirección do lazo de volta é "0:0:0:0:0:0:0:1", que se abrevia a "::1"
- Para facilitar o paso de IPv4 a IPv6 unha dirección IPv4 pode escribirse en IPv6 como "::FFFF:193.110.128.200"

# Funcións de conversión de formatos

C proporciona funcións para converter direccións IPv4 e IPv6 de formato textual a binario

- Dúas son válidas para IPv4 e IPv6 (definidas en `arpa/inet.h`):
  - ▷ `inet_pton`: de textual a binario
  - ▷ `inet_ntop`: de binario a textual

# Función `inet_pton`: textual a binario

```
int inet_pton(int af, const char *src, void *dst)
```

## ■ Parámetros:

- ▷ `af` enteiro que debe valer `AF_INET` para direcciónes IPv4 o `AF_INET6` para IPv6<sup>8</sup>
- ▷ `src` punteiro a un string coa dirección IP a converter
- ▷ `dst` punteiro ao resultado, debe apuntar a unha `struct in_addr` para IPv4 ou a unha `struct in6_addr` para IPv6<sup>9</sup>
- ▷ A IP gardase en orden de rede

## ■ Valor devolto: 1 en caso de éxito, 0 ou -1 en caso de erro

---

<sup>8</sup>As siglas AF significan *address family*. Por razóns históricas, as veces usase PF (*protocol family*) no seu lugar, pero son equivalentes.

**CITIUS** <sup>9</sup>Debe apuntar a un área reservada co tamaño adecuado

# Función `inet_pton`: textual a binario

Exemplo de uso:

```
struct in_addr miip;  
if(inet_pton(AF_INET, "193.110.128.200", (void *) &miip) != 1) {  
    fprintf(stderr, "Formato de direccion incorrecto");  
    exit(EXIT_FAILURE);  
}  
printf("%X\n", miip.s_addr); // Imprime C8806EC1
```



# Función `inet_ntop`: binario a textual

```
const char *inet_ntop(int af, const void *src,  
char *dst, socklen_t size)
```

## ■ Parámetros:

- ▷ `af` igual que antes
- ▷ `src` punteiro a unha struct `in_addr` para IPv4 ou a unha struct `in6_addr` para IPv6
- ▷ `dst` punteiro á cadea no que se gardará o resultado<sup>10</sup>
- ▷ `size` enteiro indicando o tamaño en bytes da cadea destino<sup>11</sup>

## ■ Valor devolto: punteiro a `dst`, NULL en caso de erro

---

<sup>10</sup> Debe apuntar a un área reservada co tamaño adecuado

**CiTIUS** <sup>11</sup> `socklen_t` é un tipo de dato enteiro sen signo de tamaño suficiente

# Función `inet_ntop`: binario a textual

Exemplo de uso:

```
struct in_addr miip;  
miip.s_addr = 0xC8806EC1;  
char miiptext[INET_ADDRSTRLEN];  
if(inet_ntop(AF_INET, (const void *) &miip, miiptext, INET_ADDRSTRLEN)  
    != NULL) {  
    printf("%s\n", miiptext); // Imprime 193.110.128.200  
}
```

# Outras funcións para IPv4

## Outras funcións (so IPv4)

- `inet_addr`: toma como entrada unha dirección IPv4 en formato textual e a devolve en formato binario sen encapsular (`in_addr_t`)
  - ▶ En caso de erro devolve `INADDR_NONE`
- `inet_aton` (`inet_ntoa`): a partir dunha dirección IPv4 en formato textual obtén o formato binario encapsulado (`struct in_addr`) (e viceversa)<sup>12</sup>
- `inet_network`: similar a `inet_addr`, devolve o formato binario en *orden de host*

# Python

## Libraría Socket

- Acceso de baixo nivel á interface BSD de sockets, a que usamos
- Automatiza moito a xestión de memoria
- Direccións IP:
  - ▷ Formato binario empaquetado
  - ▷ Cadea de caracteres
- Moitas funcións e constantes equivalentes ás de C, para compatibilidade

# Python

- `socket.AF_INET, socket.AF_INET6`
- `socket.inet_pton(address_family, ip_string)`
- `socket.inet_ntop(address_family, packed_ip)`

# Python

## Exemplo:

```
>>> import socket
>>> import binascii
>>> res = socket.inet_pton(socket.AF_INET, "193.110.128.200")
>>> res
b'\xc1n\x80\xc8'
>>> binascii.hexlify(res)
b'c16e80c8'
>>> socket.inet_ntop(socket.AF_INET, b'\xc1n\x80\xc8')
'193.110.128.200'
```

# Índice

- 1 Direccións IP
  - Direccións IPv4
  - Orden de host e orden de rede
  - Direccións IPv6
- 2 Portos
- 3 Sockets
- 4 Conversión do orden dos bytes
- 5 Servizo de Nomes de Dominio (DNS)
- 6 Outras funcións de conversión



# Portos e servizos

Porto: número enteiro que identifica de forma única un servizo (ou aplicación) nun nodo final da rede

- Por exemplo, computador executando un servidor web e un servidor de e-mail (SMTP)
  - ▷ Servizo web: identificado polo porto 80 (HTTP) ou 443 (HTTPS)
  - ▷ Servizo e-mail: identificado polo porto 25
- Un paquete de datos destinado a ese computador debe especificar o puerto (servizo) ao que vai dirixido

Existen números de portos asignados a servizos concretos e outros de uso libre

- Os números van do 0 ao  $2^{16} - 1 = 65535$



# Portos

Macros de interese (definidas en `netinet/in.h` e `netdb.h`) e en `socket` en Python

- `IPPORT_RESERVED`: portos menores que ese valor (1024) están reservados para o superusuario e servizos estándares
- `IPPORT_USERRESERVED`: portos maiores que este valor (5000) poden ser usados polos usuarios
- Cando se usa un `socket` sen especificar o seu porto, o sistema xeneralle automaticamente un porto comprendido entre `IPPORT_RESERVED` e `IPPORT_USERRESERVED`
- `NI_MAXSERV`: lonxitude máxima da cadea de texto que representa a un servizo

Unha lista de servizos y o seu porto asociado pódese ver no ficheiro `/etc/services`

# Índice

- 1 Direccións IP
  - Direccións IPv4
  - Orden de host e orden de rede
  - Direccións IPv6
- 2 Portos
- 3 Sockets
- 4 Conversión do orden dos bytes
- 5 Servizo de Nomes de Dominio (DNS)
- 6 Outras funcións de conversión



# Sockets

Un **socket** é unha estrutura software nun nodo que serve como punto final para enviar e recibir datos

- Establecen una interface entre a aplicación e a capa de transporte
- Analogía: servizo de correo postal
  - ▷ Aplicación: usuario que deposita a carta nunha caixa de correo
  - ▷ Capa de transporte: carteiro que recolle a carta da caixa de correo
  - ▷ Socket: a caixa de correo
- Hai diferentes tipos de sockets: locais, TCP, UDP, raw

Para establecer unha comunicación entre dous puntos necesitamos dous sockets, un en cada punto da mesma

- Socket orixen
- Socket destino

# Sockets

Para crear un socket hai que especificar:

- O dominio de comunicación (AF\_LOCAL, AF\_INET, AF\_INET6, etc.)
  - ▷ Indica a familia da dirección a usar na comunicación
- O tipo de socket (SOCK\_STREAM, SOCK\_DGRAM, SOCK\_RAW, etc.)
- O protocolo a usar na comunicación (IPPROTO\_TCP, IPPROTO\_UDP, IPPROTO\_ICMP, etc.)
  - ▷ Normalmente, para cada dominio/tipo hai un único protocolo (pe, AF\_INET/SOCK\_STREAM → IPPROTO\_TCP)
  - ▷ Pódese usar 0 como “calquer protocolo”
  - ▷ A lista de protocolos e o seu número correspondente pódese ver no ficheiro `/etc/protocols` ou en <https://www.iana.org/assignments/protocol-numbers/>

# Sockets de rede

Traballaremos con sockets coas seguintes especificacións:

- Dominio: `AF_INET` ou `AF_INET6`
- Tipo: `SOCK_STREAM` (orientado a conexión) ou `SOCK_DGRAM` (non orientado a conexión)
- Protocolo: por defecto para cada par dominio/tipo (protocolo 0)

É necesario ligar (*bind*) cada socket a unha dirección IP e a un porto:

- Dirección IP do ordenador orixen e porto do proceso orixen
- Dirección IP do ordenador destino e porto do proceso destino
- Valores encapsulados nunha `struct sockaddr` (definida en `sys/socket.h` y `netinet/in.h`)

# Estructura `struct sockaddr`

Un par dirección IP/porto encapsulase nunha estrutura de tipo `struct sockaddr`

- Esta estrutura usase na especificación de *sockets* e noutras funcións

A `struct sockaddr` é unha estrutura xenérica

- Usanse versións particulares da mesma según o dominio de conexión (pe `AF_INET` ou `AF_INET6`)
- Faise un *cast* de esas versións á xenérica cando se usan en funcións
  - ▷ Evita ter que usar funcións distintas para cada tipo

# Estructuras particulares para IPv4 y IPv6

## ■ Direcciones IPv4

struct sockaddr_in	
sa_family_t sin_family	dominio (AF_INET)
struct in_addr sin_addr	a dirección IPv4
uint16_t sin_port	o número de puerto

Tamaño = `sizeof(struct sockaddr_in)`

## ■ Direcciones IPv6

struct sockaddr_in6	
sa_family_t sin6_family	dominio (AF_INET6)
struct in6_addr sin6_addr	a dirección IPv6
uint32_t sin6_flowinfo	campo no usado
uint16_t sin6_port	o número de puerto

Tamaño = `sizeof(struct sockaddr_in6)`.

# Índice

- 1 Direccións IP
  - Direccións IPv4
  - Orden de host e orden de rede
  - Direccións IPv6
- 2 Portos
- 3 Sockets
- 4 Conversión do orden dos bytes
- 5 Servizo de Nomes de Dominio (DNS)
- 6 Outras funcións de conversión





# Conversión do orden dos bytes

Tanto ás IPs como aos portos lles afecta o orden no que se almacenan os bytes:

- Orden de rede: debe ser sempre big-endian
- Orden de host: pode ser little o big-endian según o host

Funcións:

- `uint16_t htons(uint16_t hs)` convirte o enteiro de 16 bits `hs` do orden de host ao orden de rede.
- `uint16_t ntohs(uint16_t ns)` convirte o enteiro de 16 bits `ns` do orden de rede ao orden de host.
- `uint32_t htonl(uint32_t hl)` convirte o enteiro de 32 bits `hl` do orden de host ao orden de rede.
- `uint32_t ntohl(uint32_t nl)` convirte o enteiro de 32 bits `nl` do orden de rede ao orden de host.

# Conversión do orden dos bytes

En Python úsase por compatibilidade con C

Funcións:

- `socket.htons(x)` convirte o enteiro de 16 bits `x` do orden de host ao orden de rede.
- `socket.ntohs(x)` convirte o enteiro de 16 bits `x` do orden de rede ao orden de host.
- `socket.htonl(x)` convirte o enteiro de 32 bits `x` do orden de host ao orden de rede.
- `socket.ntohl(x)` convirte o enteiro de 32 bits `x` do orden de rede ao orden de host.

# Conversión do orden dos bytes

Por exemplo, a estrutura `struct sockaddr` require orde de rede

```
struct sockaddr_in addr4;  
uint16_t porto
```

- para introducir un porto:

```
porto = 13000;  
addr4.sin_port = htons((uint16_t) porto);
```

- para ler un porto:

```
porto = ((struct sockaddr_in *)  
info->ai_addr)->sin_port;  
printf("%hu", ntohs(porto));
```

# Índice

- 1 Direccións IP
  - Direccións IPv4
  - Orden de host e orden de rede
  - Direccións IPv6
- 2 Portos
- 3 Sockets
- 4 Conversión do orden dos bytes
- 5 Servizo de Nomes de Dominio (DNS)**
- 6 Outras funcións de conversión



# Servizo de Nomes de Dominio (DNS)

## Nomes de equipos

- Nomes que identifican ordenadores na Internet (pe [www.usc.es](http://www.usc.es))
  - ▷ Máis fácil de recordar que unha IP
  - ▷ Son cadeas de caracteres de lonxitude máxima `NI_MAXHOST`
- Un nome pode ter varias direccións IP
- Un ordenador pode ter varios nomes

# Servizo de Nomes de Dominio (DNS)

O Domain Name System permite converter nomes a direccións IPs e viceversa

- Existen funcións en C que permiten facer consultas ao DNS

Funcións xenéricas:

- `getaddrinfo`
  - ▷ Devolve a información de DNS dun nome de dominio en forma dunha lista enlazada de direccións
  - ▷ Resolve un nome de servizo ao porto asociado
- `getnameinfo`
  - ▷ Resolución inversa da anterior

## getaddrinfo: nomes a direccións

```
int getaddrinfo(const char *node, const char *srv,  
                const struct addrinfo *hints,  
                struct addrinfo **res)
```

- `node`: nome do host a converter (pode ser NULL se `srv` non o é)
- `srv`: nome do servizo a converter (pode ser NULL se `node` non o é)
- `hints`: estrutura `addrinfo` que permite especificar o que se quere obter
- `res`: lista enlazada de estruturas `addrinfo` co resultado

Valor devuelto: 0 en caso de éxito, non 0 en caso de erro

- Ver `man getaddrinfo` para máis información dos erros

# Estrutura addrinfo

Estrutura usada para representar direccións

struct addrinfo	
struct sockaddr *ai_addr	Punteiro á dirección resolta
socklen_t ai_addrlen	Lonxitude da dirección
char *ai_canonname	Nome canónico do host consultado
int ai_family	Familia da dirección
int ai_socktype	Tipo de socket
int ai_protocol	Tipo de protocolo da conexión
int ai_flags	Opcións adicionais
struct addrinfo *ai_next	Punteiro ao seguinte elemento na lista



# Estrutura `addrinfo`

- `ai_family`: valores `AF_INET`, `AF_INET6` o `AF_UNSPEC` (obtéñ direccións IPv4 y IPv6)
- `ai_socktype`: valores `SOCK_STREAM` ou `SOCK_DGRAM` (si 0 aceptanse ambos)
- `ai_protocol`: código do protocolo, se 0 o protocolo por defecto para a familia/tipo
- `ai_flags`: múltiples opcións (combinables usando `|`):
  - ▷ `AI_CANONNAME`: o nome canónico do dominio aparece no campo `ai_canonname` do primeiro elemento da lista de direccións
  - ▷ `AI_V4MAPPED`: se se especifica `AF_INET6` e non existen direccións IPv6 devólvense as IPv4 en formato IPv6
  - ▷ `AI_V4MAPPED | AI_ALL`: se se especifica `AF_INET6` devuelvense as IPv6 e as IPv4 en formato IPv6

# Exemplo de getaddrinfo

```
struct addrinfo *result, *rp, hints;  
struct sockaddr_in *socka;  
struct in_addr *ip;  
char iptext[INET_ADDRSTRLEN];  
int error;  
  
memset(&hints, 0, sizeof(hints));  
hints.ai_family = AF_INET;  
hints.ai_socktype = SOCK_STREAM;  
hints.ai_protocol = 0;  
hints.ai_flags = AI_CANONNAME;  
  
error = getaddrinfo("www.elpais.es", NULL, &hints, &result);  
if (error != 0) {  
    fprintf(stderr, "Error en getaddrinfo: %s\n", gai_strerror(error));  
    exit(EXIT_FAILURE);  
}
```

## Exemplo de getaddrinfo (cont.)

```
printf("Nombre canonico: %s\n", result->ai_canonname);  
for(rp = result; rp != NULL; rp = rp->ai_next) {  
    socka = (struct sockaddr_in *) rp->ai_addr;  
    ip = &(socka->sin_addr);  
    if(inet_ntop(rp->ai_family, ip, iptext, sizeof(iptext)) == NULL) {  
        perror("Error en inet_ntop");  
        exit(EXIT_FAILURE);  
    }  
    printf("Direccion IP: %s\n", iptext);  
}  
freeaddrinfo(result);
```

Salida da execución:

```
$ ./a.out
```

```
Nombre canónico: lb-redireccionesweb-pro-407952733.eu-west-1.elb
```

```
Dirección IP: 108.128.37.20
```

```
Dirección IP: 34.249.78.60
```

# Python `socket.getaddrinfo`: nomes a direccións

```
res = socket.getaddrinfo(host, port, family=0,  
type=0, proto=0, flags=0)
```

- Similar a C
- `host`: pode ser un nome de dominio, IP ou `None`
- `port`: pode ser un nome de servizo, porto numérico ou `None`
- `family`, `type`, `proto`, `flags`: como as hints en C, limitan que mostrar á saída, ver manual
- `res`: lista de ata 5 valores (se se pide `AI_CANONNAME` en flags)
- `(family, type, proto, canonname, sockaddr)`
- `sockaddr`:
  - ▷ `(address, port)` para IPv4
  - ▷ `(address, port, flowinfo, scope_id)` para IPv6

# Exemplo de getaddrinfo en Python

```
>>> res = socket.getaddrinfo(None, "https")
>>> len(res)
4
>>> res[0]
(<AddressFamily.AF_INET6: 30>, <SocketKind.SOCK_DGRAM: 2>, 17, '', ('::1', 443, 0, 0))
>>> res[1]
(<AddressFamily.AF_INET6: 30>, <SocketKind.SOCK_STREAM: 1>, 6, '', ('::1', 443, 0, 0))
>>> res[2]
(<AddressFamily.AF_INET: 2>, <SocketKind.SOCK_DGRAM: 2>, 17, '', ('127.0.0.1', 443))
>>> res[3]
(<AddressFamily.AF_INET: 2>, <SocketKind.SOCK_STREAM: 1>, 6, '', ('127.0.0.1', 443))
>>> res[3][4][1]
443
```

# Exemplo de getaddrinfo en Python. Tratamento de erros

```
try:
    res = socket.getaddrinfo("googs", None)
    print(res)
except socket.gaierror as err: #para os erros connecidos propios da
    funcion, ver manual
    print(err)
except OSError as err: #para os erros xenericos do SO, ver manual
    print(err)
```

# getnameinfo: direccións a nomes

```
int getnameinfo(const struct sockaddr *addr,  
socklen_t addrlen, char *host, socklen_t  
hostlen, char *srv, socklen_t srrlen, int flags)
```

- `addr`: dirección a converter
- `addrlen`: tamaño da estrutura `addr`
- `host`: cadea co nome do host devolto
- `hostlen`: lonxitude máxima da cadea `host`
- `srv`: cadea co nome do servizo devolto
- `srrlen`: lonxitude máxima da cadea `srv`
- `flags`: opcións que modifican o comportamento

Valor devolto: 0 en caso de éxito, non 0 en caso de erro

- Ver `man getnameinfo` para máis información dos erros

# getnameinfo: algúns flags

Algúns valores de flags:

- NI\_NAMEREQD: devólvese un erro se falla a traducción
- NI\_NOFQDN: devólvese o nome do host sen o dominio completo
- NI\_IDN: converte os nomes en formato IDN (*Internationalized Domain Name*) á codificación local<sup>13</sup>



# socket.getnameinfo: direccións a nomes en Python

`socket.getnameinfo(sockaddr, flags)`

- `sockaddr`: dirección a converter, como tupla (`host`, `port`)
- `flags`: opcións que modifican o comportamento, análogas a C

Valor devolto: tupla (`host`, `port`), con información adicional en `host` para IPv6 se é necesario

```
>>> sockaddr = ('193.110.128.200', 22)
>>> socket.getnameinfo(sockaddr, 0)
('raw.elmundo.es', 'ssh')
>>> sockaddr = ('193.110.128.200', 80)
>>> socket.getnameinfo(sockaddr, 0)
('raw.elmundo.es', 'http')
```

# Funcións auxiliares para `getaddrinfo` y `getnameinfo`

Dúas funcións auxiliares:

- `void freeaddrinfo(struct addrinfo *res)`
  - ▷ Libera a memoria da estrutura `res`
- `const char *gai_strerror(int errcode)`
  - ▷ Xera unha mensaxe para os códigos de erro de `getaddrinfo` e `getnameinfo`

# Índice

- 1 Direccións IP
  - Direccións IPv4
  - Orden de host e orden de rede
  - Direccións IPv6
- 2 Portos
- 3 Sockets
- 4 Conversión do orden dos bytes
- 5 Servizo de Nomes de Dominio (DNS)
- 6 Outras funcións de conversión**



# Outras funcións de conversión de nomes a IPs

Dúas funcións (**obsoletas**) para acceder ao DNS

```
struct hostent *gethostbyname(const char *name)
```

- Devolve a información do DNS sobre o host name

```
struct hostent *gethostbyaddr(const void *addr,  
socklen_t len, int type)
```

- Devolve a información do DNS sobre o host de dirección IP addr
- addr: punteiro a unha dirección IPv4 ou IPv6 en formato binario
- len: lonxitude da dirección anterior
- type: AF\_INET o AF\_INET6

NOTA: estas funcións devolven un punteiro a un buffer estático, e que, polo tanto, sobreescribese en sucesivas chamadas á función. Ademais, as funcións non son reentrantes, é dicir, que o buffer pódese sobreescribir se varios fíos dun mesmo programa chaman á función.

# Estrutura hostent

Almacena unha entrada del DNS

struct hostent	
char *h_name	o nome canónico do host
char **h_aliases	lista de alias do host, acabada en NULL
int h_addrtype	o tipo de dirección, AF_INET o AF_INET6
int h_length	a lonxitude en bytes de cada dirección
char **h_addr_list	lista de IPs do host, acabada cun punteiro nulo
char *h_addr	sinónimo para h_addr_list[0]

# Outras funcións de conversión de servizos a portos

```
struct servent *getservbyname(const char *name,  
const char *proto)
```

- Devolve a información do porto asociado ao servizo name
- proto o protocolo (por exemplo, "tcp" o "udp"), se NULL buscase calquer protocolo

```
struct servent *getservbyport(int port, const  
char *proto)
```

- Devolve a información do servizo asociado ao porto port

# Estrutura servent

Almacena información asociada a un servizo

struct servent	
char *s_name	o nome canónico do servizo
char **s_aliases	lista de alias do servizo, acabada en NULL
int s_port	o número de porto (en orden de rede)
char *s_proto	o nome do protocolo que usa este servizo

# Funcións de obtención de información de protocolos

O número asociado a un protocolo obtense no ficheiro */etc/protocols*.

- `struct protoent *getprotobyname(const char *name)`
  - ▷ Devolve a información sobre o protocolo denominado `name`
- `struct protoent *getprotobynumber(int proto)`
  - ▷ Devolve a información sobre o protocolo de número `proto`



# Estructura protoent

Almacena información asociada a un protocolo

struct protoent	
char *p_name	o nome canónico do protocolo
char **p_aliases	lista de alias do protocolo, acabada en NULL
int p_port	o número do protocolo (en orden de host)