Curso de programación en C moderno

(II Edición)

Neira Ayuso, Pablo Falgueras García, Carlos

Tema 15

Sockets

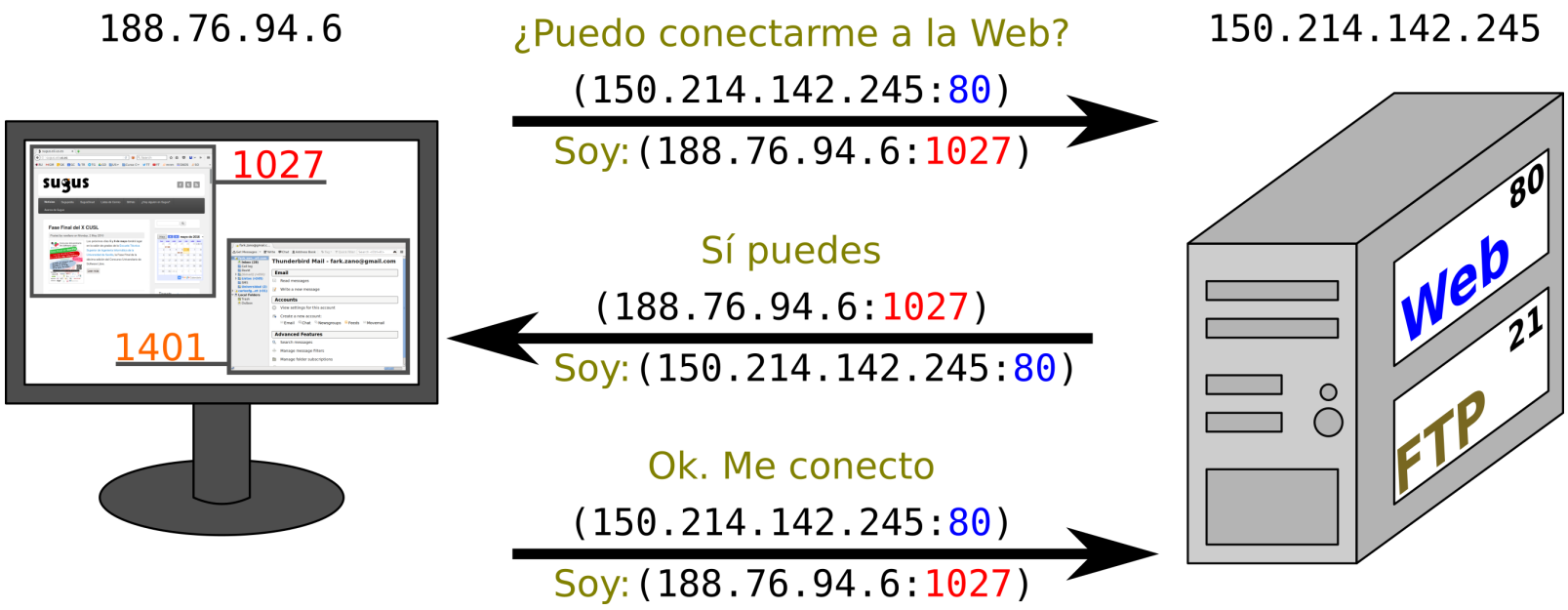
Índice

Un poco de redes

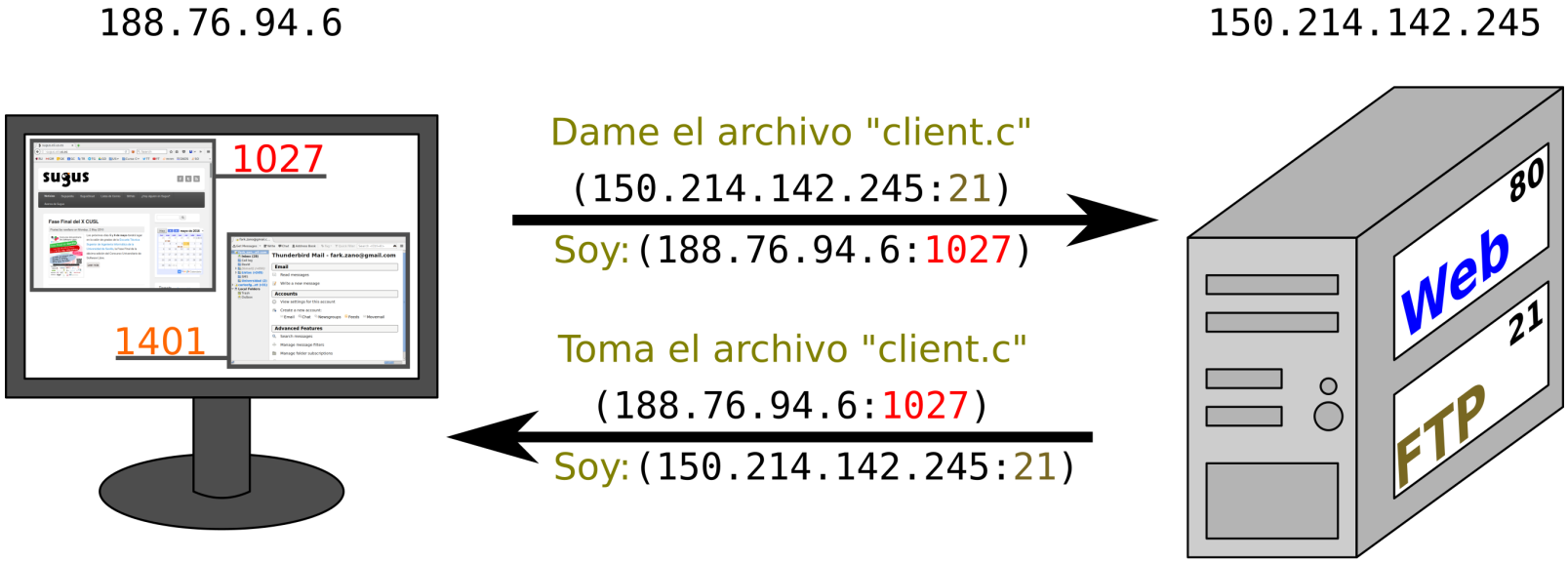
- **IP**: Dirección de cada máquina. Por ejemplo: 192.168.0.1
- **TCP**: Protocolo **con conexión**. Garantiza la llegada de los paquetes en orden
- **UDP**: Protocolo **sin conexión**. No garantiza ni la llegada, ni el orden de los paquetes; pero si llegan, llegan sin errores
- **Puerto**: Proceso/servicio/cliente de cada máquina

Conexión TCP

Apretón de manos en tres pasos (3-way handshake)



Sin conexión



Sockets

- Un socket es una manera de hablar con otros procesos a través de descriptores de ficheros Unix
- Estos programas no tienen por qué estar en la misma máquina. La comunicación a través de internet se hace de forma transparente
- Puedes enviar y recibir cualquier información de otro proceso en otra máquina, en la otra punta del mundo, tan solo escribiendo y leyendo de un fichero

"*En Unix todo es un fichero*"

Byte order

"There really is no easy way to say this, so I'll just blurt it out: your computer might have been storing bytes in reverse order behind your back. I know! No one wanted to have to tell you."

— Beej Jorgensen Guide to Network Programming

- No todas las máquinas almacenan la información en memoria en el mismo orden
- La comunicación entre máquinas con distinto *endianess* es un problema
- El programador tiene que ser consciente de esto al enviar y recibir información de una máquina remota
- Las siguientes funciones nos facilitan la tarea:
  - `uint32_t htonl(uint32_t hostlong)`
  - `uint16_t htons(uint16_t hostshort)`
  - `uint32_t ntohl(uint32_t netlong)`
  - `uint16_t ntohs(uint16_t netshort)`

Estructuras para direcciones

Por temas de herencia histórica y retrocompatibilidad, las estructuras para representar una dirección de red son un poco caóticas:

`struct addrinfo`: Información sobre direccion, socket, protocolo, . . . (es una lista)

`struct sockaddr_storage`: Estructura genérica para direcciones (cabe IPv6)

- `struct sockaddr`: Estructura genérica para direcciones (no cabe IPv6)
  - `struct sockaddr_in`: Estructura para direcciones IPv4. (puerto + dirección)
    - `struct in_addr`: Estructura para direcciones IPv4
  - `struct sockaddr_in6`: Estructura para direcciones IPv6. (puerto + dirección + info extra)
    - `struct in6_addr`: Estructura para direcciones IPv6

Estructuras para direcciones


```
1  struct sockaddr_storage {
2      sa_family_t  ss_family;     // address family
3  };
4  struct sockaddr {
5      unsigned short    sa_family;    // address family, AF_xxx
6      char              sa_data[14];  // 14 bytes of protocol address
7  };
8
9  struct sockaddr_in {
10     short int          sin_family;  // Address family, AF_INET
11     unsigned short int sin_port;    // Port number
12     struct in_addr     sin_addr;    // Internet address
13     unsigned char      sin_zero[8]; // Same size as struct sockaddr
14 };
15 struct in_addr {
16     uint32_t s_addr; // that's a 32-bit int (4 bytes)
17 };
18
19 struct sockaddr_in6 {
20     u_int16_t       sin6_family;   // address family, AF_INET6
21     u_int16_t       sin6_port;     // port number, Network Byte
           Order
22     u_int32_t       sin6_flowinfo; // IPv6 flow information
23     struct in6_addr sin6_addr;     // IPv6 address
24     u_int32_t       sin6_scope_id; // Scope ID
25 };
26 struct in6_addr {
27     unsigned char   s6_addr[16];   // IPv6 address
28 };
```

Estructuras para direcciones


```
1  struct addrinfo {
2      int              ai_flags;      // AI_PASSIVE, AI_CANONNAME, etc.
3      int              ai_family;     // AF_INET, AF_INET6, AF_UNSPEC
4      int              ai_socktype;   // SOCK_STREAM, SOCK_DGRAM
5      int              ai_protocol;   // use 0 for "any"
6      size_t           ai_addrlen;    // size of ai_addr in bytes
7      struct sockaddr *ai_addr;       // struct sockaddr_in or _in6
8      char            *ai_canonname;  // full canonical hostname
9
10     struct addrinfo *ai_next;       // linked list, next node
11 };
```

Direcciones y cadenas

Esta par de funciones nos permiten convertir entre el número binario de una dirección y su representación como cadena de texto:

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

Mediante el parámetro `int af`, indicamos el tipo de dirección que estamos tratando: (`AF_INET` ó `AF_INET6`)

Estas funciones están obsoletas por no tratar bien con IPv6, por lo que se desaconseja su uso:

- `gethostbyname()`
- `gethostbyaddr()`

Funciones principales

TCP		UDP	
Server	**Client**	**Server**	**Client**
socket()	socket()	socket()	socket()
bind()		bind()	
listen()			
accept()	connect()		
recv() send()	recv() send()	recvfrom() sendto()	recvfrom() sendto()
close()	close()	close()	close()

Estructura cliente-servidor

- **socket()**: Crea el socket (devuelve un descriptor de fichero)
- **bind()**: Asocia una dirección al socket
- **listen()**: Pone el servidor en modo escucha para aceptar peticiones de conexión
- **accept()**: Acepta la primera petición de la cola o se queda esperando hasta que llegue una
- **connect()**: Intenta establecer una conexión con un servidor
- **send()/sendto**: Envía información
- **recv()/recvfrom**: Recibe información
- **close()**: Cierra un descriptor de fichero (en nuestro caso, el socket)

Estudiar código cliente-servidor